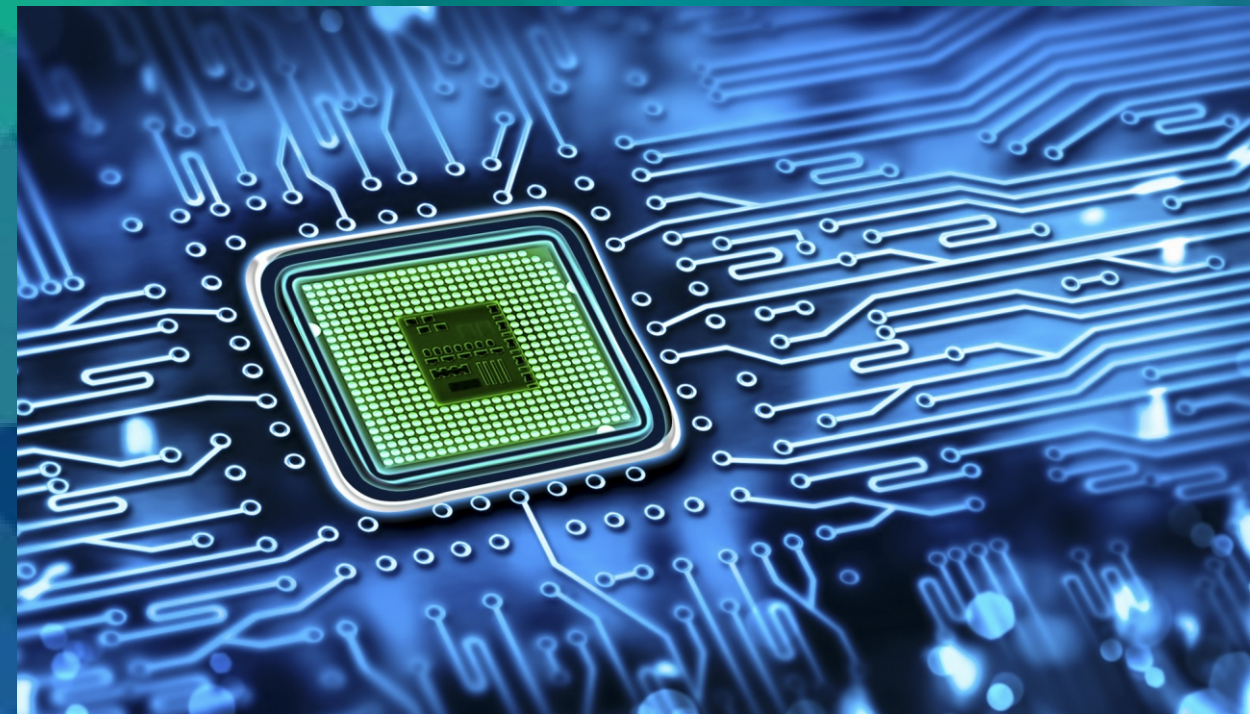


ЦИФРОВА СХЕМОТЕХНІКА ТА АРХІТЕКТУРА МІКРОПРОЦЕСОРІВ

*С.П. Євсєєв, Н.В. Дженюк, М.Ю. Охрименко,
С.О. Головашич, Д.А. Кудій*

ЦИФРОВА СХЕМОТЕХНІКА ТА АРХІТЕКТУРА МІКРОПРОЦЕСОРІВ

Навчальний посібник



Видавництво ПП «Новий Світ-2000»



Видавництво ПП «Новий Світ-2000»



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

*Євсєєв С. П., Дженюк Н.В., Охрименко М.Ю.,
Головашич С.О., Кудій Д.А.*

ЦИФРОВА СХЕМОТЕХНІКА ТА АРХІТЕКТУРА МІКРОПРОЦЕСОРІВ

Навчальний посібник

Харків – НТУ «ХП»
Львів – Видавництво ПП «Новий Світ – 2000»
2023

УДК 004.27:004.31
E25

Рекомендовано Вченою Радою Національного технічного університету
«Харківський політехнічний інститут» як навчальний посібник
для студентів галузей 12 – «Інформаційні технології» та 17 – «Електроніка,
автоматизація та електронні комунікації» (протокол № 6 від 07.07.2023 року).

Рецензенти:

Казакова Н.Ф., д.т.н., проф., завідувач кафедри інформаційних технологій
Одеський державний екологічний університет;

Остапов С.Е., д.ф-м.н., проф., завідувач кафедри програмного забезпечення
комп'ютерних систем Чернівецького національного університету
імені Ю.Федьковича.

Євсєєв С.П., Дженюк Н.В., Охрименко М.Ю.,
Головашич С.О., Кудій Д.А.

E25 Цифрова схемотехніка та архітектура мікропроцесорів:
навчальний посібник / Євсєєв С. П., Дженюк Н.В., Охрименко М.Ю.
та ін. – Харків, – Львів: Видавництво ПП «Новий Світ – 2000», 2023.
– 513 с. (Укр. мов.)

ISBN 978 – 966 – 418 – 396 – 0

У посібнику викладаються арифметичні та логічні основи мікроелектронних схем комп'ютерної схемотехніки. Розглядаються основи фізичних процесів та функціонування типових мікроелектронних схем сучасної комп'ютерної техніки. Наводяться принципи побудови основних аналогових та цифрових пристроїв, засоби їх математичного опису, а також основи аналізу та синтезу елементів комп'ютерної інженерії із заданими технічними характеристиками.

Книга розрахована на студентів вищих навчальних закладів III-IV рівня акредитації, які навчаються в галузях 12 – «Інформаційні технології» та 17 – «Електроніка, автоматизація та електронні комунікації».

УДК 004.27:004.31

© Національний технічний університет
«Харківський політехнічний інститут»

© Євсєєв С.П., Дженюк Н.В., Охрименко М.Ю.,
Головашич С.О., Кудій Д.А. 2023

© Видавництво ПП «Новий Світ – 2000»,
ФОП Піча С.В., 2023

ISBN 978 – 966 – 418 – 396 – 0

ВСТУП

Сучасний стан розвитку суспільства характеризується впровадженням інформаційних технологій у всі сфери діяльності людини. Вихід суспільства на новий рівень розвитку невід'ємно пов'язаний з вирішенням всіх проблем інформатизації, тобто з забезпеченням повсюдного використання автоматизованих методів збирання, передавання, обробки та зберігання інформації різноманітного призначення. Технічною основою вирішення вказаних проблем спеціалісти називають всебічне використання відповідних комп'ютеризованих засобів, практичне вивчення схемотехнічних основ яких і складає предмет даної книги.

Таким чином, комп'ютери стали невід'ємною частиною нашої цивілізації. І хоча перша електронна обчислювальна машина з'явилася трохи більше, ніж п'ятдесят років тому, на цей час вже нараховується п'ять поколінь обчислювальних машин. Такого бурного розвитку не знала, мабуть, жодна технологія.

Дійсно, якщо перша ЕОМ "Еніак" (1946 рік) займала площу близько 100 м² з вагою більш ніж 30 тон, то сучасний мікропроцесор, здатний вмістити все електронне обладнання такої машини, має площу всього 1,5 – 2 см², забезпечуючи при цьому таку ж обчислювальну потужність, яка перебільшує сумарну обчислювальну потужність всіх ЕОМ, що працювали в світі в середині 60-х років. Перша ЕОМ вміщувала близько 17000 електронних ламп, а зараз таку кількість електронних компонентів 0,15 мкм технології дозволяють розмістити в зрізі волосини людини.

Темпи розвитку ЕОМ спростовували всі найбільш сміливі припущення. Наприклад, президент та засновник однієї з провідних комп'ютерних фірм Digital Equipment Corp. Кен Олсон (Ken Olson) у 1977 році в одному з інтерв'ю сказав, що відсутні причини, за якими кому-небудь забажалося мати дома комп'ютер, а уже декілька років потому на ринку з'явився найбільш масовий та в цей час персональний комп'ютер: спочатку ПК фірми Apple, а у 1981 році і IBM PC.

Подібний стрімкий розвиток комп'ютерної техніки став можливим завдяки постійному швидкому удосконаленню комп'ютерної схемотехніки, тобто елементів, вузлів, пристроїв, які є складовими частинами будь-якого комп'ютера.

В даний час існує велика кількість різноманітних посібників з архітектури мікропроцесорів та компонентів ЕОМ. У них, як правило, наведена архітектура якихось окремих специфічних пристроїв і їх технічний опис, або приведена інформація платформ старого покоління, яка частково або повністю втратила

свою актуальність. Розгляд архітектури системи зі сторони основного компонента – процесора розглядається вкрай рідко. У цьому посібнику детально описано роботу процесора та його взаємодію з системою в цілому. Затронуті всі основні компоненти системи сучасної архітектури ЕОМ.

Вивчення архітектури ЕОМ дозволяє освоїти концептуальну структуру обчислювальної машини. Студент повинен урахувати специфіку апаратної частини й інші, часто унікальні, властивості системи в якій функціонує програма, використовувати ці властивості, наприклад, застосовуючи спеціально оптимізований для даної архітектури алгоритм.

Особлива увага приділяється особливостям роботи процесора в реальному та захищеному режимі та таким граничним питанням, як паралельні структури обчислювальних систем, багатопроцесорні й багатомашинні обчислювальні системи та т.і.

У навчальному посібнику застосовуються нові методики подачі матеріалу, системні підходи до вивчення дисципліни архітектури ЕОМ, які дозволяють освоїти концептуальну структуру обчислювальної машини. Враховувати специфіку апаратної частини, унікальні властивості системи в якій функціонує програма, використовувати ці властивості.

Представлений в навчальному посібнику матеріал надасть фахівцям практичні рекомендації і допоможе навчитися проектувати і розробляти оптимізоване системне програмне забезпечення.

Зміст навчального посібника побудований на відкритих матеріалах вітчизняних і зарубіжних літературних джерел, а також авторських досліджень операційних систем реального часу.

1. ФОРМИ ЗОБРАЖЕННЯ ІНФОРМАЦІЇ

1.1. Історія розвитку засобів комп'ютерної техніки

Комп'ютерна техніка – це сукупність засобів для автоматизації процесів обробки інформації, а також галузь техніки, яка займається розробленням, виготовленням та експлуатацією комп'ютерів.

Комплекс електронного устаткування, яке призначене для автоматизації процесів обробки дискретної інформації і має спільне керування, називають цифровою електронною обчислювальною машиною (ЕОМ). У наш час діапазон використання ЕОМ суттєво розширився: називають більше 20 тис. областей їх застосування – від науково інженерних задач до штучного інтелекту, математичного моделювання, робототехніки. Тому замість терміна ЕОМ використовують його більш сучасний синонім – комп'ютер.

Перші відомості про обчислювальні пристрої відносяться до 1500р., коли Леонардо да Вінчі розробив ескіз 13-розрядного підсумовуючого пристрою з десятизубцевими колесами. У 1642 р. Б. Паскаль у Франції створив першу діючу модель восьмирозрядного цифрового пристрою “Паскаліна”. Кожна цифра в ньому визначалася положенням десятисекторного колеса. Додавання здійснювалося поворотом колеса на відповідне число секторів. Німецький вчений Г. Лейбніц у 1670 р. описав механічну лічильну машину, яка виконувала операції додавання, віднімання, множення і ділення.

У 1833 р. англійський математик Ч. Бебідж спроектував і майже 30 років вдосконалював “аналітичну машину”, в яку були закладені фундаментальні для комп'ютерної техніки принципи:

- автоматичне (без зупинки) виконання операцій;
- необхідність пам'яті (“складу”) для зберігання чисел, набраних зубчастими колесами;
- робота за програмою, що вводиться “на ходу” з перфокарти в виконавчий пристрій (“фабрику”) зі швидкістю виконання операцій у машині.

Обмежені технічні можливості того часу не дозволили реалізувати ці прогресивні ідеї.

Автоматичні цифрові машини на механічних і релейних елементах створювали одночасно і незалежно в багатьох країнах. Так, у 1936 р. А. Тьюрінг в Англії і Е. Пост в США розробили концепцію абстрактної автоматичної цифрової машини, яка могла розв'язувати будь-яку алгоритмізовану задачу. У 1941 р. німецький інженер К. Зюс розробив дві електромеханічні машини з програмним керуванням “Зюс-1” і “Зюс-2”.

У 1944 р. в США під керівництвом Г. Айкена була розроблена електромеханічна машина “Марк-1”, де вперше були використані ідеї Ч. Бебіджа. У тому ж році в США під керівництвом Д. Еккерта, Д. Моучлі та Дж. фон Неймана був створений перший у світі комп’ютер з використанням електронних ламп і електромеханічних реле. В машині було задіяно 18 тис. ламп 11,5 тис. реле; вона виконувала за одну секунду 5 тис. операцій додавання, 300 – множення, споживала потужність 150 кВт. Ці перші машини мали такі недоліки:

- програми не зберігалися в пам’яті, а набиралися за допомогою зовнішніх складних і довготривалих комутацій;
- дані подавалися і оброблювалися в десятковій системі числення послідовно (розряд за розрядом).

У подальшому розвиток комп’ютерної техніки здійснювався шляхом нарощування їх функціональних можливостей та удосконалення елементної бази.

1.2. Характеристика електричних сигналів

При передачі інформації на відстані за допомогою радіотехнічних систем використовуються різноманітні види електричних (радіотехнічних) сигналів. З математичної точки зору, будь-який сигнал можна представити деякою функцією часу $u(t)$, яка характеризує зміну його миттєвих значень напруги (у більшості випадків) струму або потужності. Реальний електричний сигнал як об’єкт аналітичних і практичних досліджень достатньо складний, а це означає, що його математична модель може бути різноманітною. Це дозволяє класифікувати сучасні електричні сигнали за рядом специфічних ознак.

За вимогами математичних критеріїв все різноманіття сигналів прийнято ділити на дві основні групи: детерміновані (регулярні) і випадкові сигнали.

Детермінованими називають радіотехнічні сигнали, миттєві значення яких у будь-який момент часу достовірно відомі, тобто, їх можна передбачити з вірогідністю, що рівна одиниці. Прикладом детермінованого радіотехнічного сигналу є гармонічне коливання.

Випадкові сигнали – це сигнали, миттєві значення яких у будь-які моменти часу невідомі і не можуть бути передбачені.

Як не парадоксально на перший погляд, але сигналом, що несе корисну інформацію, може бути тільки випадковий сигнал. Інформація закладена в множині амплітудних, частотних (фазових) або кодових змін сигналу, що передається. Наприклад, випадковим сигналом є електрична напруга, отримана в результаті перетворення мови, музики, зображення та ін.

Окрім інформаційних (корисних) сигналів у просторі мають місце електромагнітні процеси, що не несуть в собі ніякої інформації – це шуми. Для систем обробки інформації ці сторонні електромагнітні процеси становлять перешкоди.

Відповідно до особливостей структури часового зображення всі електричні сигнали можна розділити на аналогові, дискретні (від лат. *discretus* – розділений, переривчастий) і цифрові.

Якщо фізичний процес, що породжує сигнал, можна представити безперервною функцією часу $u_a(t)$ (рис. 1.1, а), то такий сигнал прийнято називати аналоговим (безперервним або континуальним, тобто коли він має розриви по осі амплітуд).

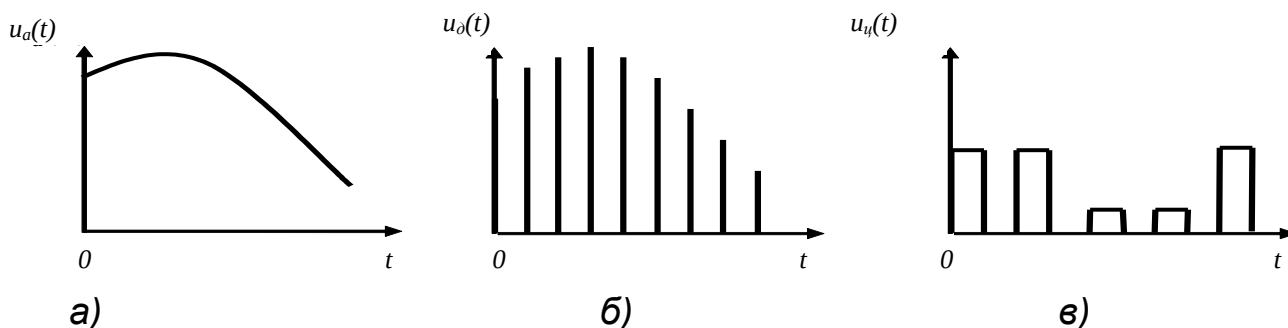


Рис. 1.1. Сигнали: а – аналоговий; б – дискретний; в – цифровий

У радіoeлектроніці широко застосовуються імпульсні системи, принцип дії яких заснований на використанні дискретних сигналів. Найпростіша *математична модель дискретного сигналу* – це послідовність точок на часовій осі, у кожній з яких задані значення відповідного безперервного сигналу (рис. 1.1б).

Одним з різновидів дискретних сигналів є *цифровий сигнал*. У ньому дискретні значення сигналу $u_q(t)$ замінюються числами, частіше всього реалізованими в двійковому коді, що представляється високим (одиниця) і низьким (нуль) рівнями потенціалів напруги (рис. 1.1 в). Електричні сигнали зручно розглядати у вигляді математичних функцій, заданих у часі, і фізичних координатах. Звідси сигнали поділяються на одномірні і багатомірні. Найбільш розповсюджені одномірні сигнали є функціями часу, а багатомірні, крім того, відображають їх положення в n -мірному просторі.

Щоб зробити аналіз проходження сигналу через пристрій обробки інформації, необхідно його представити в зручній математичній формі. У теорії сигналів широке застосування знайшли два способи їх математичного і фізичного зображення: часовий і спектральний. Це стало можливим завдяки принципу ду-

альності (подвійності, зв'язку) частоти і часу, адже $F = \frac{1}{T}$. Спектральний спосіб

оснований на зображенні сигналу у вигляді суми гармонійних складових різних, зазвичай кратних один одному частот. При часовому способі аналізу сигнал відображається безперервною функцією часу або сукупністю елементарних імпульсів, що йдуть один за одним через визначені інтервали часу. Обидва способи рівнозначні, і перевага надається більш зручним операціям над сигналом даної форми при проведенні потрібного перетворення або аналізу.

Одиночним імпульсом напруги (струму) називається напруга (струм), що діє протягом визначеного інтервалу часу, названого тривалістю імпульсу t_i (рис.1.2а). В електронних пристроях найбільш часто застосовуються імпульси з тривалістю від одиниць наносекунд до одиниць мілісекунд.

Однак частіше доводиться мати справу з послідовностями t -імпульсів (рис.1.2б), що періодично повторюються через проміжок часу T , названий періодом проходження. Інтервал часу між імпульсами t_n називається тривалістю паузи. Відношення періоду проходження до тривалості імпульсу називається скважністю $Q = \frac{T}{t_i}$.

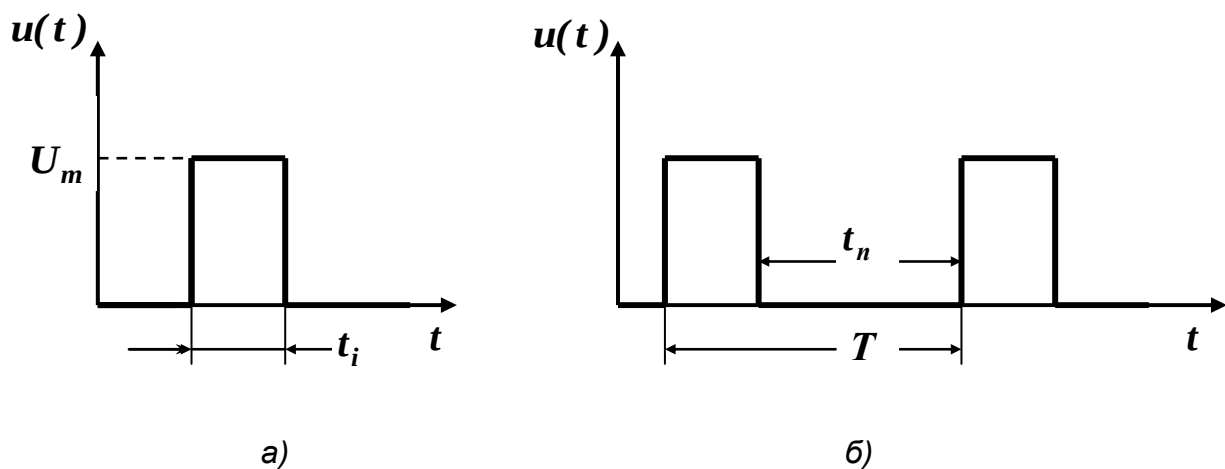


Рис. 1.2. а – одиночний імпульс, б – послідовність імпульсів

При імпульсному режимі роботи тривалість паузи найчастіше набагато більше тривалості імпульсу ($t_n \gg t_i$), тому фактично період проходження визначається тривалістю паузи, тобто $T \approx t_n$.

Характерною рисою всіх імпульсів є наявність у їх формі ділянок з різними швидкостями зміни напруг (струмів), що пояснюється інерційністю властивостей елементів схем. Типовий реальний імпульс представлений на (рис.1.3).

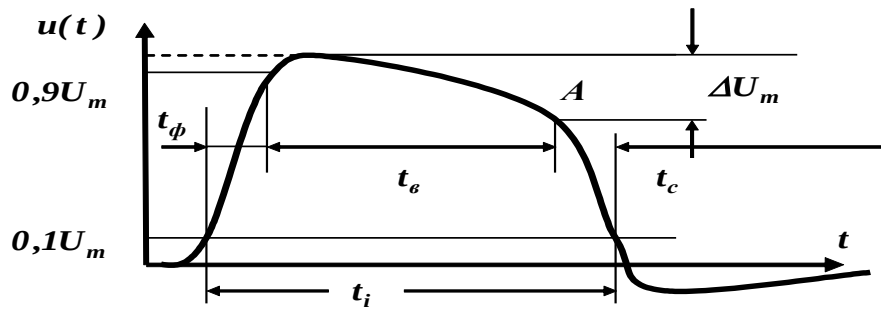


Рис. 1.3. Реальний імпульс

З рисунка видно, що вимірювання тривалості окремих інтервалів часу, протягом яких значення похідної $\frac{du(t)}{dt}$ сильно розрізняється, і залежить від чутливості вимірювача. Тому прийнято тривалість імпульсу t_i вимірювати на рівні $0,1U_m$, де U_m – амплітуда імпульсів.

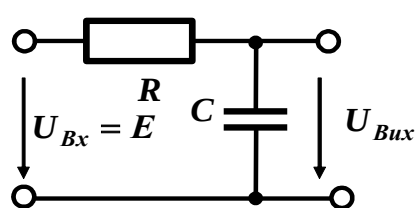
Перший в імпульсі інтервал часу, коли напруга змінюється швидко, називається фронтом імпульсу і вимірюється від рівня $0,1 U_m$ до рівня $0,9 U_m$. Другий інтервал часу швидкої зміни напруги називається спадом (або зрізом) імпульсу і вимірюється від моменту часу, що відповідає швидкому збільшенню похідної $\frac{du(t)}{dt}$ (А, до рівня $0,1 U_m$).

Частина сигналу, де похідна $\frac{du(t)}{dt}$ мала, називається вершиною імпульсу, тривалість якої $t_g = t_i - t_c - t_\phi$. Вершина імпульсу характеризується також абсолютним Δu і відносним $\lambda = \frac{\Delta u}{U_m}$ зниженнями. Наприкінці імпульсу існує викид зворотної полярності великої тривалості, іменований хвостом [1].

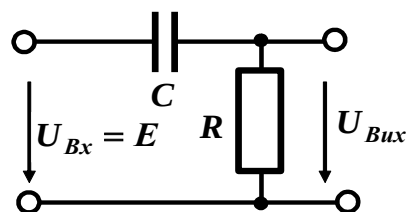
Крім перерахованих параметрів імпульсного сигналу важливими характеристиками його є енергія імпульсу $W = \int u^2(t)dt$, а для послідовності імпульсів – середнє значення $U_0 = \frac{1}{T} \int_0^T u(t)dt$, інакше назване постійною складовою .

1.3. Проходження прямокутних імпульсів через RC-ланцюжки

Для кращого розуміння фізичних процесів, що відбуваються з імпульсними сигналами під час їх обробки елементами та вузлами комп'ютерної схемотехніки, необхідно мати чітке уявлення впливу на імпульсний сигнал так званих RC-ланцюгів.



RC -ланцюг першого роду



RC -ланцюг другого роду

Рис. 1.4. Види RC-ланцюгів

Існує два різних типи RC-ланцюгів, що складаються з послідовно з'єднаних резистора і конденсатора. Вони отримали назву RC-ланцюгів першого і другого роду (рис. 1.4).

При підключенні до входу постійної напруги E для ланцюгів обох видів маємо відповідно до закону Кирхгофа рівняння:

$$u_C(t) + i(t)R = u_{Bx}(t). \quad (1.1)$$

Оскільки струм, що протікає через конденсатор і напруга на ньому пов'язані формулами $i(t) = C \frac{du_C(t)}{dt}$ та $u_C(t) = \frac{1}{C} \int_0^t i(t) dt$, то можна записати:

$$RC \frac{du_C(t)}{dt} + u_C(t) = u_{Bx}(t), \quad (1.2)$$

або

$$i(t) = \frac{u_{Bx}(t)}{R} - \frac{1}{RC} \int_0^t i(t) dt. \quad (1.3)$$

Прийнято позначати RC через τ , що має назву постійна (стала) часу RC-ланцюга.

Розглянемо фізичні процеси в RC-ланцюгах при проходженні через нього прямокутного імпульсу (рис.1.5).

З приходом вхідного імпульсу в RC-ланцюзі починають відбуватися перехідні процеси заряду конденсатора (ємності) C . В початковий момент часу ($t = 0$) вся вхідна напруга прикладається до резистора R , а потім конденсатор починає заряджатися за експоненціальним законом до величини вхідної напруги $u_{C\max}(t) = U_{Cm} = E$. По ходу заряду конденсатора струм $i_3(t)$ у ланцюзі змен-

шується, що спричиняє експоненціальне зменшення напруги яка прикладається до резистора (рис.1.5).

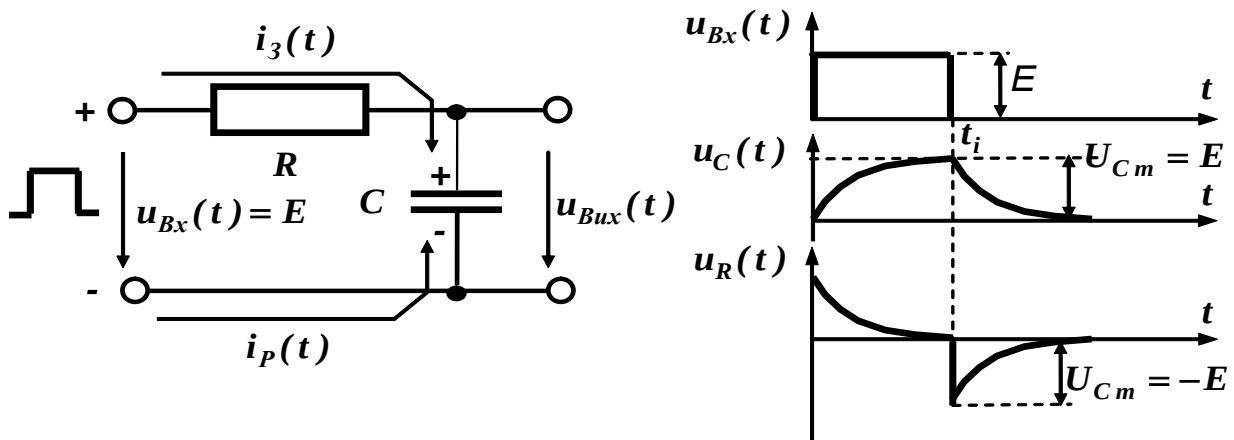


Рис. 1.5. Принцип функціонування RC-ланцюга

Після закінчення вхідного імпульсу в момент часу роль джерела напруги починає грати конденсатор. Він починає розряджатися. Струм розряду конденсатора $i_P(t)$ протікає шляхом, що протилежний шляху струму попереднього заряду конденсатора. В момент весь заряд ємності прикладається до опору зі зворотною полярністю. По ходу розряду конденсатора напруга на резисторі і конденсаторі зменшується до нуля (рис.1.5).

Для аналізу RC-ланцюгів важливе значення має постійна часу ланцюга. За час рівний 3τ в RC-ланцюзі закінчуються всі перехідні процеси, тобто конденсатор повністю заряджається або розряджається. У зв'язку з цим існують особливості функціонування RC-ланцюгів залежно від співвідношення тривалості вхідного імпульсу t_i , тривалості його фронту t_ϕ та спаду t_C , а також постійної часу ланцюга τ .

Ланцюги з дуже великою сталою часу ($\tau \gg t_i$)

У RC-ланцюзі першого роду під час впливу вхідної напруги в інтервалі часу $0 < t < t_i$ відбувається лінійна зміна напруги на конденсаторі $u_C(t)$. Така зміна відповідає інтегралу від постійної величини. Ці висновки можна підкріпити аналітичними виразами. Якщо виконується умова, що постійна часу $\tau = RC$ має достатньо велике значення, то з виразу (1.3) маємо $i(t) = \frac{u_{Bx}(t)}{R}$.

Напруга на конденсаторі є водночас і вихідною напругою, тому

$u_C(t) = u_{Bux}(t) = \frac{1}{C} \int_0^t i(t) dt = \frac{1}{RC} \int_0^t u_{Bx}(t) dt$, тобто вихідний сигнал становить проінтегрований вхідний сигнал помножений на коефіцієнт пропорційності $\frac{1}{RC} = \tau^{-1}$.

Такі RC -ланцюги прийнято називати інтегруючими RC -ланцюгами (рис.1.6).

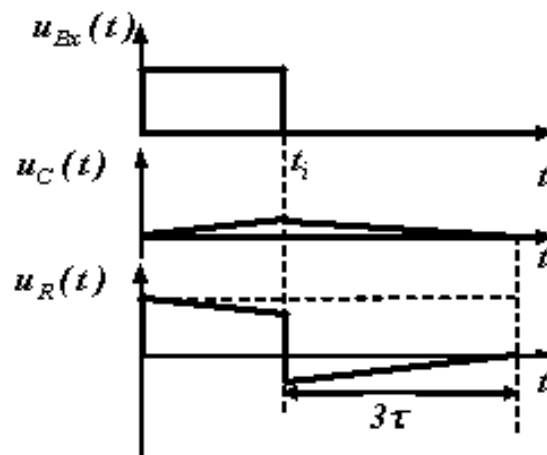


Рис. 1.6. Часові діаграми RC -ланцюга при $t \gg t_i$

У ланцюзі другого роду вихідна напруга $U_R(t)$ передає вхідну напругу без збільшення тривалості фронту і спаду. Чим суворіше виконується співвідношення $\tau \gg t_i$, тим менше викривлення вхідних сигналів. Тому такі RC -ланцюги прийнято називати передавальними (іноді розділювальними) і вони широко використовується на практиці для невикривленої передачі імпульсних сигналів.

Ланцюги з малою постійною часу ($\tau \ll t_i$)

Вихідна напруга RC -ланцюга першого роду $u_C(t)$ становить викривлений вхідний імпульс. Викривленням піддаються фронт і спад імпульсу, тривалість яких дорівнює 3τ ланцюга. Тому на практиці такий ланцюг не застосовується.

Вихідна напруга RC -ланцюга другого роду $u_R(t)$ становить два різнополярних імпульси, тривалість кожного з якого менше тривалості вхідного, а їх тимчасове положення відповідає фронту і спаду імпульсу. Ланцюг називається таким що скорочує і широко використовується на практиці (рис. 1.7).

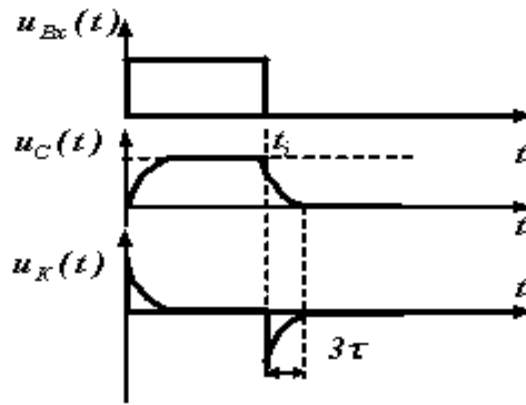


Рис. 1.7. Часові діаграми RC-ланцюга при $t \ll t_i$

Ланцюги з дуже малою постійною часу ($\tau \ll t_\phi$, $\tau \ll t_c$)

При виконанні нерівностей ($\tau \ll t_\phi$, $\tau \ll t_c$) фронт і спад вхідного імпульсу не сприймаються RC-ланцюгом як перепади напруги. За час фронту (або спаду) перехідний процес в RC-ланцюзі закінчується, що істотно позначається на формі напруг. При такій малій постійній часу вхідний імпульс по суті є трапецеїдальним, що і враховано при побудові часових діаграм (рис. 1.8).

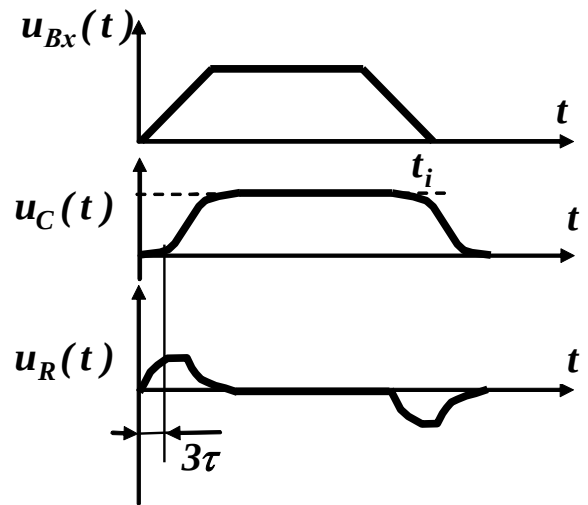


Рис. 1.8. Часові діаграми RC-ланцюга при $t \ll t_\phi$ та $t \ll t_c$

RC-ланцюг першого роду не знайшов використання на практиці.

Для RC-ланцюга другого роду при лінійній зміні вхідної напруги (під час фронту і спаду) напруга на виході ланцюга постійна за виключенням часу перехідних процесів 3τ . При постійній вхідній напрузі (вершина імпульсу) напруга на резисторі $u_R(t)$ дорівнює 0. Такими властивостями володіє похідна, тому такий RC-ланцюг називається RC-ланцюг, що диференціює. Дійсно, як слідує з виразу (1.1)

$$u_{Bx}(t) = u_C(t) + i(t)R = \frac{1}{C} \int_0^t i(t) dt.$$

Якщо опір резистора R малий, то член $i(t)R$ можна не брати до уваги, тоді струм у ланцюзі $i(t) \approx C \frac{du_{Bx}(t)}{dt}$ і вихідна напруга ланцюга, що знімається з ре-

зистора R , визначається формулою $u_{Bux}(t) = i(t)R = RC \frac{du_{Bx}(t)}{dt}$.

Таким чином, залежно від співвідношення між постійною часу RC -ланцюга та тривалістю імпульсного сигналу, RC -ланцюг може набувати різних властивостей, які широко використовуються в комп'ютерній схемотехніці.

КОНТРОЛЬНІ ПИТАННЯ

1. Назвіть основні характеристики аналогового і дискретного сигналів.
2. Дати визначення скважності.
3. Назвіть основні характеристики реального імпульсного сигналу.
4. Навести схему RC -ланцюга що розділяє, що диференціює, що інтегрує.
5. Яким чином змінюється напруга на елементах RC -ланцюга першого (другого) роду при впливі на нього прямокутного імпульсу?
6. Дати визначення сталої часу для RC -ланцюга.
7. Навести математичні вирази зміни напруги на елементах RC -ланцюга при впливі на нього імпульсу прямокутної форми.
8. Яким чином зміниться АЧХ RC -ланцюга першого (другого) роду якщо величину конденсатора (резистора) збільшити (зменшити) в декілька разів?
9. Яким чином повинні співвідноситися стала часу RC -ланцюга та тривалість імпульсу, щоб ланцюжок виконував функції інтегрування, скорочування та диференціювання?
10. Яким чином можна зменшити викривлення фронтів імпульсів для RC -ланцюга другого роду?
11. Пояснити вплив елементів ланцюга на форму АЧХ та ФЧХ.

2. ІНТЕГРОВАНІ СИСТЕМИ ЕЛЕМЕНТІВ

2.1. Принципи схемотехніки цифрових інтегральних схем

Згідно з булевою алгеброю будь-яку як завгодно складну логічну функцію можна виразити за допомогою набору найпростіших функцій. Такий набір називається *функціонально повним* або *логічним базисом*. Мінімальними логічними базисами можуть бути набори: **I** та **НІ**, **ЧИ** та **НІ**, **НІ-I**, **НІ-ЧИ** тощо. Це положення є основою для важливого принципу проектування, відповідно до якого майже всі цифрові інтегральні схеми у складі однієї серії виконуються на єдиному логічному елементі, який називається базовим і реалізує найчастіше операцію **НІ-ЧИ** або **НІ-ЧИ**. У ряді випадків у ролі базового елемента виступає інвертор.

Базовий логічний елемент (БЛЕ) має єдину принципову і топологічну схеми. У цьому сенсі він становить ту найпростішу “цеглинку”, з якої складаються “будинки” – всі інші цифрові ІС. Звідси випливає, що електричні параметри і властивості всіх ІС в одній серії однозначно визначаються параметрами і властивостями базового логічного елемента. Це *перший схемотехнічний принцип*.

Другий схемотехнічний принцип для цифрових пристроїв обумовлюється властивістю подвійності логічних елементів і може бути сформульований так: будь-який цифровий пристрій можна будувати тільки на елементах **НІ-I** або тільки на елементах **НІ-ЧИ** без зміни функціональних зв'язків між ними. Розходження є лише у тому, що логічні рівні напруги на всіх входах і виходах будуть протилежними. Таким чином, немає необхідності розглядати правила побудови цифрових ІС та пристроїв окремо на логічних елементах (ЛЕ) **НІ-I** й окремо на **ЛЕ НІ-ЧИ**. Вони єдині [1].

Потенційне кодування визначило третій принцип схемотехніки цифрових інтегральних схем – ключовий режим роботи їх активних елементів. При такому режимі струми та напруги в електричних колах визначаються в основному напругами джерел живлення й опорами резисторів і слабо залежать від параметрів транзисторів і діодів. Легко одержувана при ключовому режимі значна різниця та стабільність логічних рівнів вихідного сигналу забезпечують досить високу завадостійкість і функціональну надійність цифрових ІС.

2.2. Транзистор як технічна основа реалізації логічних функцій

Підсилювальні каскади на транзисторах із спільним емітером або спільним витоком зазвичай служать основою для побудови ключів. Усі такі пристрої працюють у режимі великого сигналу з обмеженням, тобто відсіченням і повним відмиканням транзисторів.

2.2.1. Ключ із спільним емітером

Найпростіша схема ключа складається з транзистора VT та резистора E_G , R_G (рис. 2.1). Джерело E_G керує ключем. Залежно від значення E_G (низький рівень E_G^0 або високий E_G^1) транзистор повинний бути в закритому (режим відсічення) або насиченому статичному стані. У першому випадку напруга на виході ключа близька до E_K , у другому – до 0. Таким чином, ключ є найпростішим логічним елементом **НЕ** – інвертором. На рис. 2.2 показане послідовне з'єднання ключів, яке часто використовують на практиці.

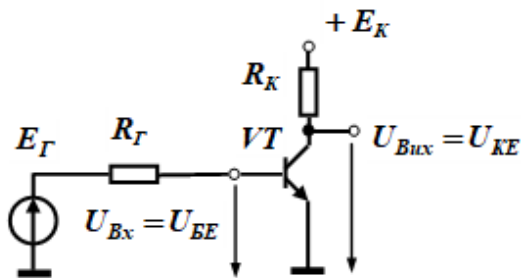


Рис. 2.1. Ключ із спільним емітером

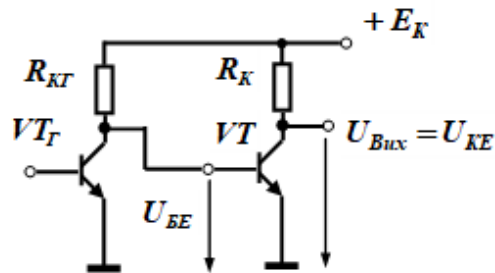


Рис. 2.2. Послідовне з'єднання ключів

Умови, за яких забезпечуються статичні стани, розглянемо за допомогою апроксимованих входних і вихідних характеристик транзистора (рис. 2.3).

З входньої характеристики випливає, що умовою запирання транзистора є:

$$U_{BE} < U_{BE_0}, \quad (2.1)$$

для транзисторів ІС $U_{BE_0} = 0,6...0,7 \text{ В}$.

Якщо як джерело керуючого сигналу використовується подібний ключ (на рис. 2.2 це VT_G і R_{KT}), то низький рівень $U_{BE} = U_{KE_H}$ не перевищує $0,2 \text{ В}$ і забезпечує надійне запирання транзистора VT .

В області насичення обидва переходи транзистора зміщені в прямому напрямку. При цьому $U_{BE} = U_{BE_H}$, $U_{BK} = U_{BK_H}$ і вони близькі один до одного (близько $0,7...0,8 \text{ В}$). Тому напруга $U_{KE} = U_{BE} - U_{BK} = U_{KE_H}$ дуже мала (у ряді практичних випадків її приймають рівною нулю). Струм колектора (рис. 2.3б), точка А) визначається виразом

$$I_{K_H} = \frac{E_K - U_{KE_H}}{R_K} \approx \frac{E_K}{R_K}. \quad (2.2)$$

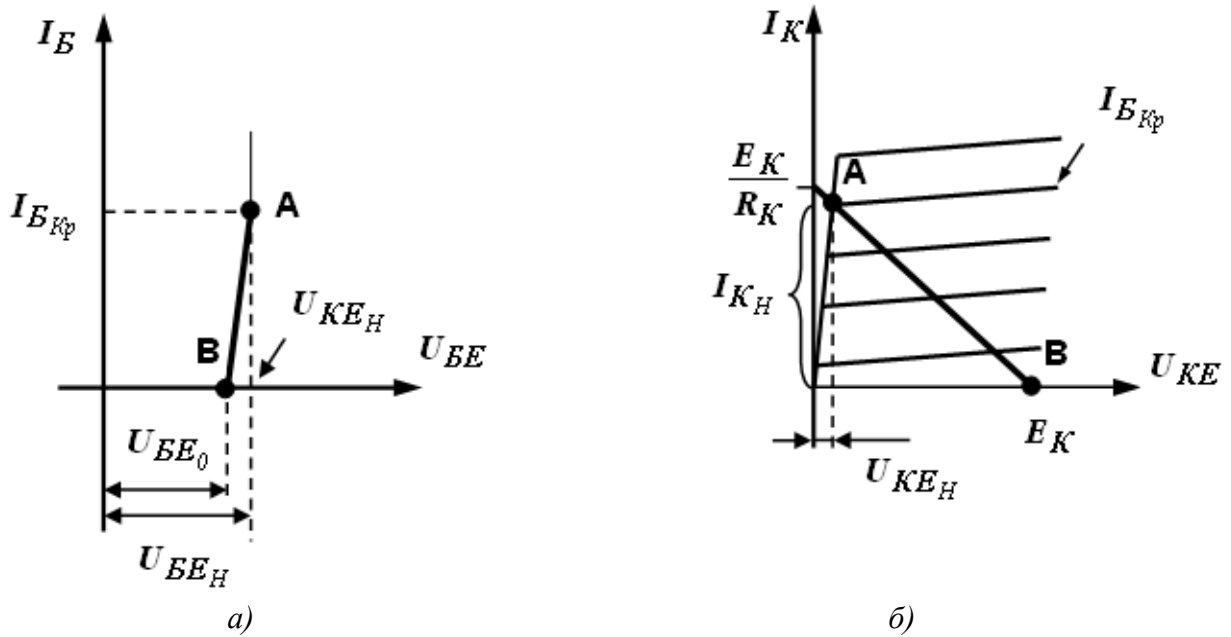


Рис. 2.3. Кусково-лінійна апроксимація вхідної (а) та вихідних (б) характеристик біполярного транзистора

Він відповідає так званому критичному струму бази $I_{B_{kp}}$, за якого транзистор знаходиться на границі активного та насиченого режимів. У цьому випадку справедливі відомі співвідношення між струмами в транзисторі, який працює в активному режимі. Зокрема,

$$I_{K_H} = \beta I_{B_{kp}}, \quad (2.3)$$

де β – статичний коефіцієнт передачі струму транзистора. Подальше збільшення струму бази практично вже не призводить до зростання колекторного струму. Таким чином, умовою насичення транзистора ключа є нерівність:

$$I_B \geq I_{B_{kp}} = \frac{I_{K_H}}{\beta}. \quad (2.4)$$

Для схеми на рис. 2.1 умова насичення, виражена через параметри елементів ключа, має вигляд:

$$\frac{E_I^1 - U_{BE_H}}{R_I} \geq \frac{E_K - U_{KE_H}}{\beta R_K}. \quad (2.5)$$

Для схеми на рис. 2.2 зазвичай $E_K \gg U_{BE_H}$, тому умова насичення спрощується: $\beta R_K \geq R_{KГ}$.

Ступінь насичення транзистора характеризується коефіцієнтом

$$S_H = \frac{I_B}{I_{B_{кр}}} = \frac{\beta I_B}{I_{K_H}}. \quad (2.6)$$

На границі насичення $I_B = I_{B_{кр}}$, $S_H = 1$. Зі збільшенням S_H збільшується навантажувальна здатність ключа (умова насичення задовольняється при більшому значенні струму I_{K_H}), зменшується вплив різних дестабілізуючих факторів на стан ключа, але, як буде показано нижче, погіршується його швидкодія. Тому ступінь насичення у всіх випадках вибирають з компромісних розумінь, виходячи з умов конкретної задачі. При цьому (у зв'язку зі значним технологічним розкидом параметра β , а також залежністю його від температури) умова насичення транзистора повинна бути виконана при мінімальному значенні β .

Швидкодія ключа визначається сумарним часом переходу з закритого (вимкненого) стану у відкритий (увімкнутий) і назад.

Для спрощення оцінки тривалості перехідних процесів при перемиканнях приймемо такі припущення: керуючий сигнал становить ідеальний прямокутний імпульс з низьким рівнем $E_G^0 = 0$, ключ не навантажений ($C_H = 0$, $R_H = \infty$) і $R_G \gg R_K$.

У вихідному стані, на інтервалі часу $0 \dots t_1$ (рис. 2.4), $E_G = E_G^0 = 0$, тому $U_{BE} = 0$, транзистор закритий і $U_{KE} = E_K$.

Вмикання ключа

При надходженні у момент часу t_1 перепаду $U_G = E_G^1$ напруга на базі транзистора наростає за експонентним законом зі сталою часу $\tau_3 \approx (C_E + C_K) R_G$ (C_E та C_K – бар'єрні ємності емітерного та колекторного переходів транзистора) і прагне до рівня E_G^1 . При досягненні значення $U_{BE} = U_{BE_H}$ емітерний перехід відкривається, його опір різко зменшується і подальший заряд ємностей припиняється. Протягом часу підготовки до вмикання t_{II} транзистор продовжує залишатися в закритому стані. Цю затримку можна визначити, скориставшись відомим правилом, відповідно до якого три-

валість будь-якої початкової ділянки експонентного процесу (рис. 2.5) описується виразом:

$$t_K - t_i = \tau \cdot \ln \frac{U(\infty) - U(t_i)}{U(\infty) - U(t_K)}, \quad (2.7)$$

де τ – стала часу процесу;

$U(\infty) - U(t_i)$ – повний розмах експоненти;

$U(\infty) - U(t_k)$ – частина експоненти, яка залишилася після t_k .

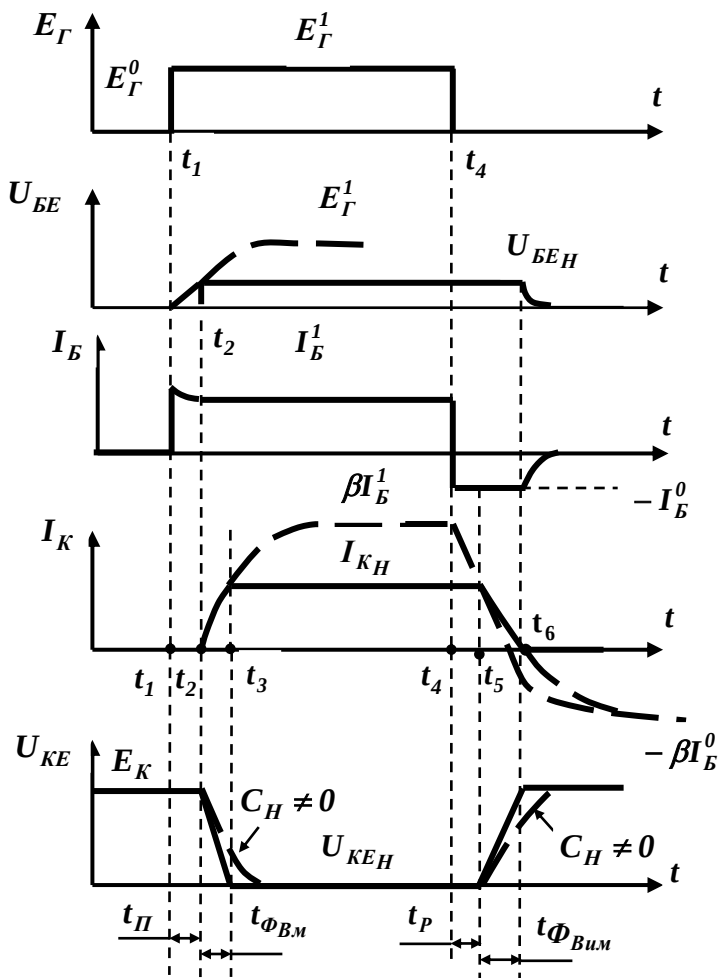


Рис. 2.4. Перехідні процеси в ключі зі спільним емітером

Таким чином,

$$t_{II} = \tau_3 \ln \frac{E_{\Gamma}^1}{E_{\Gamma}^1 - U_{BE_H}}. \quad (2.8)$$

У момент часу t_2 транзистор переходить в активний режим. Його колекторний струм наростає зі сталою часу τ_β ($C_n = 0$), прагнучи до рівня:

$$\beta I_B^1 = \beta \frac{E_I^1 - U_{BE_H}}{R_I}. \quad (2.9)$$

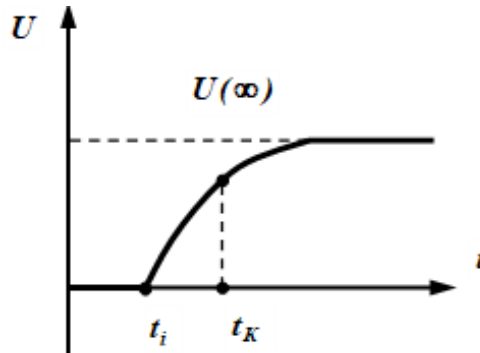


Рис. 2.5. Пояснення правила визначення початкової ділянки експоненти

При досягненні значення I_{K_H} транзистор переходить у режим насичення і зростання струму колектора припиняється. Оскільки ємність навантаження $C_H = 0$, то напруга на колекторі змінюється аналогічно струму I_K й у момент часу t_3 досягає рівня U_{KE_H} .

Відповідно до правила (вираз 2.7) тривалість фронту вмикання:

$$t_{\phi_{BM}} = t_3 - t_2 = \tau_e \ln \frac{\beta I_B^1}{\beta I_B^1 - I_{K_H}} = \tau_e \ln \frac{S_H}{S_H - 1}, \quad (2.10)$$

де $\tau_e \approx \tau_\beta + (\beta_0 + 1)C_K R_{K_H}$ (при $C_H = 0$);

β_0 – коефіцієнт передачі струму бази на низькій частоті;

$$\tau_\beta = \frac{\tau_\alpha}{1 - \alpha_0}, \text{ де } \tau_\alpha = \frac{1}{\omega_\alpha}, \text{ } \omega_\alpha - \text{верхня гранична частота, на якій модуль}$$

комплексного коефіцієнта передачі струму емітера в схемі зі спільним емітером $\alpha(\omega)$ зменшується в $\sqrt{2}$ стосовно коефіцієнта передачі струму на низькій частоті α_0 ;

R_{K_H} – опір паралельно з'єднаних опорів R_K і навантаження R_H .

Для зменшення часу затримки вмикання $t_{3m}^{1,0} = t_{II} + t_{\phi_{BM}}$ необхідно збільшувати вмикаючий струм I_B^1 і підвищувати ступінь насичення транзистора (E_I^1 збільшувати, R_I зменшувати).

Для типових транзисторів і джерел сигналу керування в ІС час вмикання ключа складає одиниці наносекунд.

Після закінчення часу $t_{3m}^{1,0}$ транзистор знаходиться у режимі насичення, його струми практично не змінюються, а заряд надлишкових носіїв у базі продовжує наростати до рівня $Q_H = \tau_n I_B^1$ зі сталою часу накопичення τ_n (зазвичай $\tau_n \approx \tau_\beta$). За час $t = (2...3)\tau_n$ процес накопичування зарядів практично завершується і транзистор переходить у стаціонарний режим насичення [2].

Вимикання ключа

У момент часу t_4 керуючий сигнал $E_\Gamma = E_\Gamma^0 = 0$. За рахунок накопичених неосновних носіїв в області бази переходи транзистора залишаються відкритими і на них підтримується напруга, близька до U_{BEH} . У базовому колі протікає зворотний струм (вимикаючий струм):

$$I_B^0 = \frac{U_{BEH}}{R_\Gamma} \quad (2.11)$$

(тут не врахований об'ємний опір тіла бази).

Поряд з рекомбінаційним процесом цей струм, протікаючи в зворотному напрямку, зменшує накопичений заряд Q_H зі сталою часу τ_n . Процес прагне до асимптотичного рівня $Q_{AC} = -I_B^0 \tau_n$. Доки заряд не досяг критичного значення $Q_{Kp} = I_{B_{Kp}} \tau_n$ (цей інтервал називають часом розсіювання) колекторний струм і напруга U_{KE} не змінюються.

Відповідно до (формули 2.7) час розсіювання:

$$t_P = \tau_n \ln \frac{U(\infty) - U(t_4)}{U(\infty) - U(t_5)} \approx \tau_\beta \ln \frac{I_B^0 + I_B^1}{I_B^0 + I_{B_{Kp}}} \approx \tau_\beta \ln \frac{\beta I_B^0 + \beta I_B^1}{\beta I_B^0 + I_{K_H}}. \quad (2.12)$$

Остання форма запису виразу (2.12) дозволяє ілюструвати процес накопичення та розсіювання заряду за допомогою часової діаграми для колекторного струму $I_K = \beta I_B$ (на рис. 2.4 цей струм показаний пунктирною лінією).

Отже, для зменшення часу розсіювання необхідно зменшувати ступінь насичення транзистора (при $S_H = 1$ час $t_P = 0$) і збільшувати вимикаючий струм I_B^0 .

У момент часу t_5 транзистор знову переходить в активний режим. Його колекторний струм зменшується зі сталою часу τ_B , прагнучи до рівня $-\beta I_B^0$. При досягненні нульового рівня транзистор закривається. Формування фронту вимикання закінчується:

$$t_{\Phi_{Вим}} = \tau_B \ln \frac{\beta I_B^0 + I_{K_H}}{\beta I_B^0}. \quad (2.13)$$

Очевидно, для зменшення $t_{\Phi_{Вим}}$ необхідно збільшувати вимикаючий струм I_B^0 .

Як і при вмиканні, наявність ємності навантаження C_H призводить до зростання $t_{\Phi_{Вим}}$. Це збільшення не перевершує величини $2,2C_H R_K$, що відповідає $t_{\Phi_{Вим}}$ при миттєвому закриванні транзистора.

Затримка вимикання ключа $t_{3m}^{0,1} = t_P + t_{\Phi_{Вим}}$ визначається, головним чином, часом t_P і може складати десятки наносекунд.

Швидкодію ключа характеризують максимальною частотою проходження вхідних сигналів

$$F = \frac{1}{t_{3m}^{1,0} + t_{3m}^{0,1}} = \frac{1}{t_{II} + t_{\Phi_{Вим}} + t_P + t_{\Phi_{Вм}}}, \quad (2.14)$$

за якої перехідні процеси встигають завершитися на всіх етапах при вмиканні та вимиканні. Часто для характеристики швидкодії використовується середній час затримки сигналу:

$$t_{3T_{сер}} = 0,5 (t_{3m}^{1,0} + t_{3m}^{0,1}). \quad (2.15)$$

При кінцевій тривалості фронту вхідного сигналу, що зазвичай і має місце, затримки вмикання та вимикання прийнято відраховувати на заданих (час-

тіше 50-процентних) рівнях вхідного й вихідного сигналів. Ці інтервали називаються відповідно часом затримки поширення сигналу при вмиканні та вимиканні.

Засоби підвищення швидкодії

Аналіз перехідних процесів у ключі показує, що затримки вмикання та вимикання залежать відповідно від вмикаючого струму бази і вимикаючого струму бази. Чим більші струми, тим крутіші фронти і менший час підготовки. Однак струм вмикання I_B^1 впливає не тільки на затримку вмикання, але й вимикання. Так, бажання зменшити $t_{3m}^{1,0}$ шляхом збільшення струму I_B^1 призводить до підвищення ступеня насичення S_H і, отже, до зростання часу розсіювання при вимиканні. Тому на практиці у відношенні до струму I_B^1 приймається компромісне рішення. Що стосується струму I_B^0 , то він впливає тільки на етапі вимикання. По цьому його прагнуть зробити максимально можливим. Розповсюдженим прийомом до ІС є створення низькоомних кіл для вмикаючого струму за допомогою діода Шотткі, який шунтує опір R_F (рис. 2.6). Низькоомне коло створюється також у схемі на рис. 2.2, де в ролі R_F виступає вихідний опір насиченого транзистора VT_F .

Найбільш тривалим етапом при перемиканнях є розсіювання. Тому швидкодія ключів підвищується більш істотно при використанні в них нелінійного зворотного зв'язку за допомогою діода Шотткі (рис. 2.7), що дозволяє практично виключити насичення і, отже, етап розсіювання.

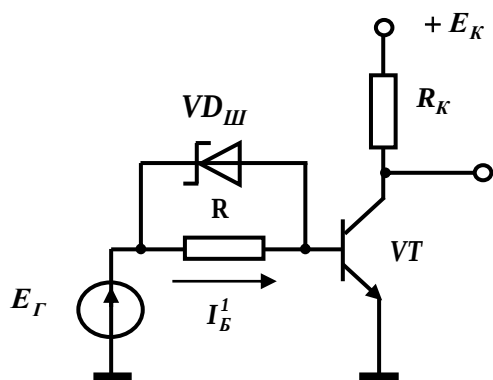


Рис. 2.6. Ключ з форсованим вимиканням за допомогою діода Шотткі

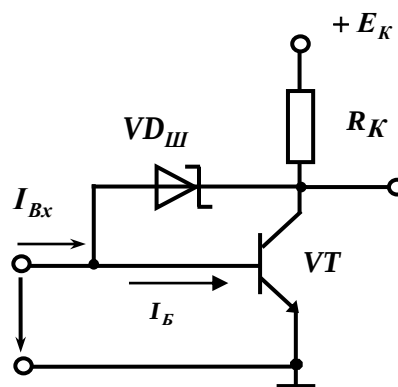


Рис. 2.7. Ключ з нелінійним зворотним зв'язком

Коли транзистор закритий або працює в активному режимі, напруга на колекторі досить висока, діод закритий і не впливає на роботу ключа. При під-

ході транзистора до режиму насичення, коли напруга між базою і колектором $U_{БК} = U_{БЕ} - U_{КЕ}$ досягає порога відмикання діода U_{VD0} , діод шунтує колекторний перехід, фіксуючи на ньому напругу на рівні, близькому до U_{VD0} . Тому що $U_{VD0} = 0,3...0,4$ В, тобто менше порога відмикання $U_{БК0}$, колекторний перехід не відкривається і транзистор у режим насичення не входить. Завдяки цьому при вимиканні $t_p = 0$. Однак платою за підвищення швидкодії тут є деяке підвищення нижнього рівня напруги на відкритому ключі.

2.2.2. Ключі на МОН-транзисторах

Створення цифрових ІС з підвищеним ступенем інтеграції обумовило особливий інтерес до базових логічних елементів з дуже малою споживаною потужністю і займаною на кристалі площею. Традиційний спосіб підвищення економічності за рахунок підвищення опору резисторів неминуче призводить до збільшення їх геометричних розмірів та паразитної ємності на підкладці. Кардинальним способом вирішення проблеми є використання в ключових схемах замість резисторів динамічного навантаження – МОН-транзисторів. Вони здатні добре працювати у режимі мікрострумів та мають малі габарити [3].

Отримали поширення дві схеми ключових елементів: на транзисторах з каналами одного типу провідності і на комплементарних транзисторах – на парах транзисторів з каналами різного типу провідності.

Ключі на МОН-транзисторах одного типу провідності.

Широке застосування знаходять *n*-канальні транзистори, оскільки вони забезпечують більш високу швидкодію, ніж *p*-канальні, а логічні елементи на їхній основі легко узгоджуються з логічними елементами на біполярних транзисторах. Принципова схема ключа на *n*-канальних транзисторах і часові діаграми, які пояснюють його роботу, наведені на рис. 2.8. Роль динамічного навантаження виконує транзистор *T2*, у якого затвор з'єднаний зі стоком, утворюючи двополюсник.

У закритому стані *VT1*, коли на його затвор подана напруга $U_{Вх}$, яка не перевищує порога відмикання $U_{ЗВ_{пор}}$ (рис. 2.9а)), струм через *VT2* практично не протікає, тому спад напруги між стоком та витоком *VT2* практично дорівнює нулю $U_{СВ2} \approx 0$ і напруга на виході ключа приблизно дорівнює напрузі живлення $U_{Вих} \approx E_{Ж}$.

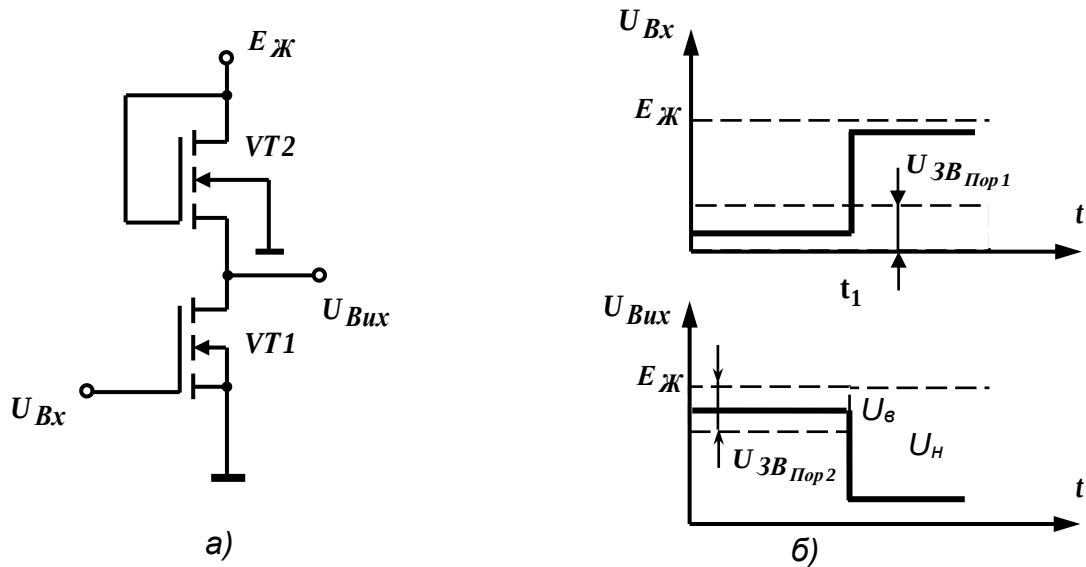


Рис. 2.8. Схема (а) і часові діаграми роботи ключа (б) на n -канальних МОН-транзисторах

Звідси випливає, що $U_{CB2} \approx 0$ і $VT2$ теж закритий. Точне значення U_{Bux}^1 визначається точкою перетину вихідної характеристики транзистора $VT1$ при напрузі між затвором та витоком $U_{3B1} \approx 0$ і лінії навантаження, яка становить динамічну характеристику $I_{C2} = f(U_{3B2})$ транзистора $VT2$ (точка A на рис. 2.9б)). Ця точка знаходиться в інтервалі $E_{Ж} \dots (E_{Ж} - U_{3B_{Пор2}})$, зміщуючись до однієї або іншої границі інтервалу залежно від співвідношення струмів витоку транзисторів. Зважаючи на це, з рис. 2.9б) видно, що наступний ключ буде надійно відкритий, якщо мінімально можлива вихідна напруга даного ключа $E_{Ж} - U_{3B_{Пор2}} > U_{3B_{Пор1}}$. Звідси випливає вимога до напруги джерела живлення: $E_{Ж} > U_{3B_{Пор1}} + U_{3B_{Пор2}}$.

У відкритому стані ключа, коли на затвор транзистора $VT1$ подана напруга $U_{Bx} \approx E_{Ж}$, його канал має низький опір, і напруга $U_{Bx} = U_{Ocm}$ (точка B на рис. 2.9б)). Якщо при цьому $U_{3B2} = E_{Ж} - U_{Ocm} > U_{3B_{Пор2}}$, то відкритий також транзистор $VT1$.

Остаточна напруга U_{Ocm} для аналогічного наступного ключа є відмикаючою напругою, отже, повинна задовольняти нерівності: $U_{Ocm} < U_{3B_{Пор1}}$, тобто бути близькою до нуля. Це можливо, коли опір каналу відкритого транзисто-

ра **VT1** значно (до двох порядків) менший опору каналу відкритого **VT2**. Іншими словами, питома крутість транзистора **VT1** повинна бути істотно вище питомої крутості **VT2**. На практиці це досягається в основному використанням транзисторів різної геометрії: **VT1** має короткий і широкий канал, а **VT2** – вузький і довгий.

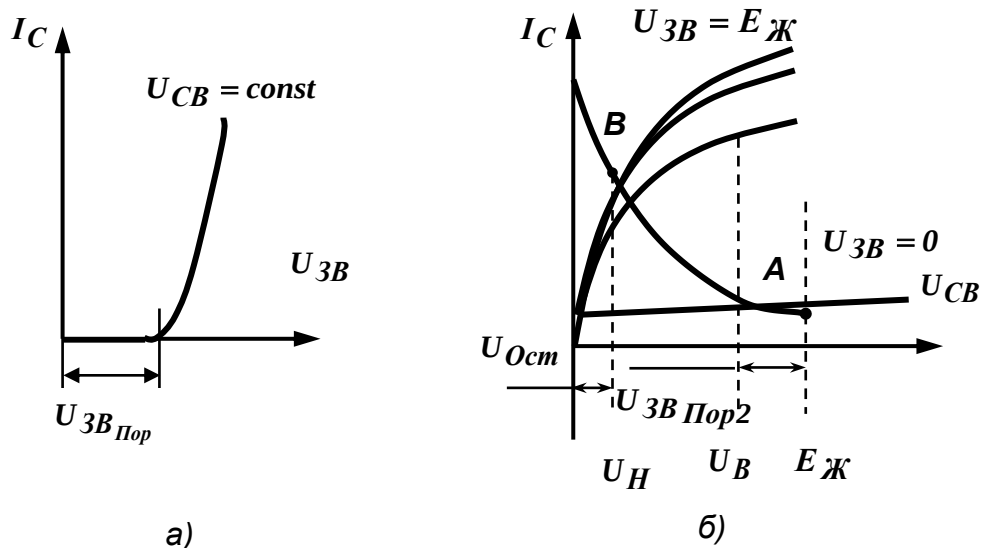


Рис. 2.9. Стокозатворна (а) і вихідні статичні характеристики (б) МОН-транзистора

Оскільки мінімальний опір каналу відкритого транзистора **VT1** зазвичай складає сотні й більш Ом, опір каналу відкритого транзистора **VT1** повинен бути десятки кілоом. Остання обставина істотно погіршує швидкодію ключа. Дійсно, на етапі вимикання, коли транзистор **VT1** швидко закривається, заряджання паразитної ємності навантаження відбувається через високоомне коло – транзистор **VT1**. Навіть при незначній ємності (одиниці пікофарад) постійна часу кола заряджання складає десятки наносекунд. Тому ключові елементи на одноступінних **МОН**-транзисторах застосовується, головним чином, у великих інтегральних схемах (**ВІС**), де паразитні ємності незначні і, крім того, реалізуються такі переваги, як невелика площа, простота і низька вартість виготовлення.

Ключ на комплементарних МОН-транзисторах

Схема та часові діаграми роботи ключа на комплементарних **МОН**-транзисторах наведені на рис. 6.10. У ньому затвори транзисторів об'єднані, джерело **VT1** підключене до спільної шини, а джерело **VT2** – до шини джерела живлення $E_{Ж}$.

Нехай на інтервалі $0...t_1$ на вході $U_{Bx} < U_{3B_{пор1}}$. Транзистор **VT1** закритий. Якщо при цьому $|U_{3B2}| = |U_{Bx} - E_{Ж}| > |U_{3B_{пор2}}|$, то транзистор **VT2** відкритий. Струм у спільному колі визначається струмом витоку транзистора **VT1**, тобто мізерно малий. Тому напруга на низькоомному каналі **VT2** теж дуже мала, і $U_{Bux} \approx E_{Ж}$.

При $U_{Bx} > U_{3B_{пор1}}$ і $|U_{3B2}| < |U_{3B_{пор2}}|$ (рис. 2.10 б); $t > t_1$) відповідно відкритий **VT1** і закритий **VT2**. Струм у загальному колі, як і раніше, дорівнює струму витоку закритого транзистора (тепер уже **VT2**). Тому $U_{Bux} \approx 0$.

Отже, через дуже малі залишкові напруги на відкритих транзисторах перепад вихідних рівнів ключа наближається до напруги джерела живлення $E_{Ж}$.

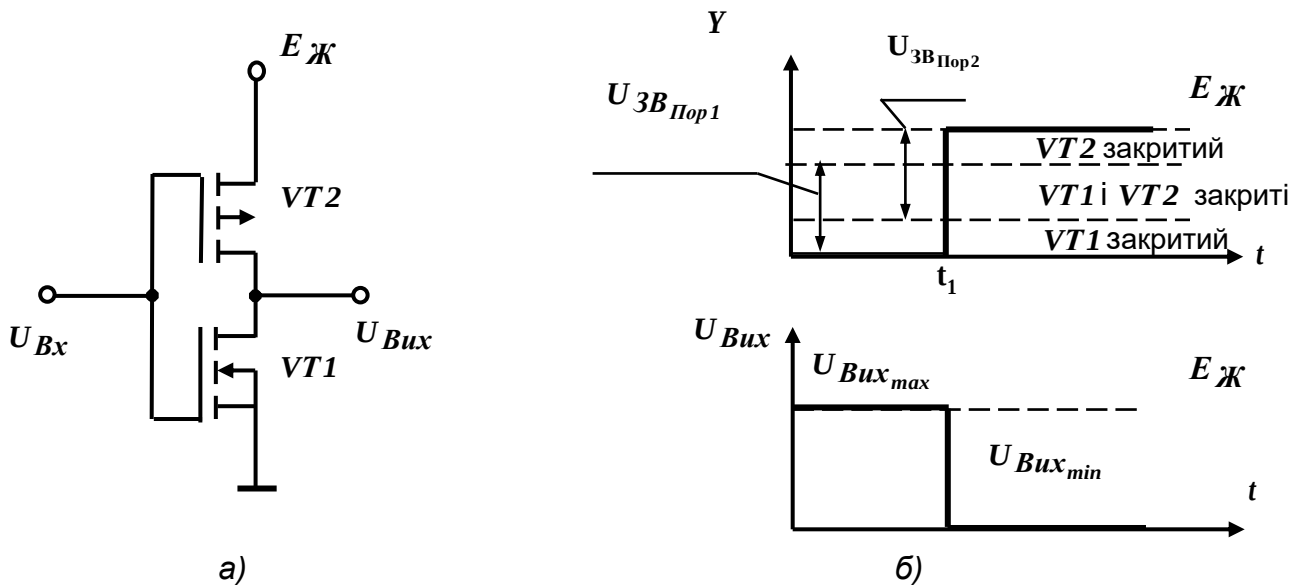


Рис. 2.10. Схема (а) та часові діаграми роботи ключа (б) на комплементарних МОН-транзисторах

Вихідний опір ключа визначається опором відкритого транзистора.

В інтересах підвищення швидкодії його виготовляють по можливості малим. Завдяки цьому швидкодія ключів на комплементарних **МОН**-транзисторах не уступає швидкодії ключів на біполярних транзисторах.

В обох статичних станах потужності від джерела живлення ключ майже не споживає, тому що один із транзисторів закритий. Для запобігання одночасного відмикання обох транзисторів при перемиканнях, коли U_{Bx} змінюється в межах $0...E_{Ж}$, необхідно забезпечити умову $U_{3B_{пор1}} + |U_{3B_{пор2}}| \geq E_{Ж}$.

Зі збільшенням частоти перемикач зростає середній струм перезаряду ємності навантаження, обумовлюючи зростання динамічної споживаної потужності. Вона може стати обмежуючим фактором на допустиму частоту перемикачів і ємність навантаження.

2.2.3. Перемикач струму

Перемикач струму виконується зазвичай на біполярних транзисторах і становить диференційний підсилювальний каскад (рис. 2.11а)), який працює в режимі великого сигналу, але без насичення транзисторів. У ньому, залежно від вхідної напруги, струм емітерного кола I_0 , який створюється джерелом $E_{\text{Ж}}$, протікає або через $VT1$, або через $VT2$, тобто перемикається з першого плеча схеми в друге. На базу транзистора $VT1$ подається вхідна керуюча напруга, а на базу транзистора $VT2$ – постійна опорна напруга $E_{On} \approx 0,5 (U_{Bx_{\text{max}}} + U_{Bx_{\text{min}}})$. Колекторні кола транзисторів підключені до спільної шини ($E_K = 0$).

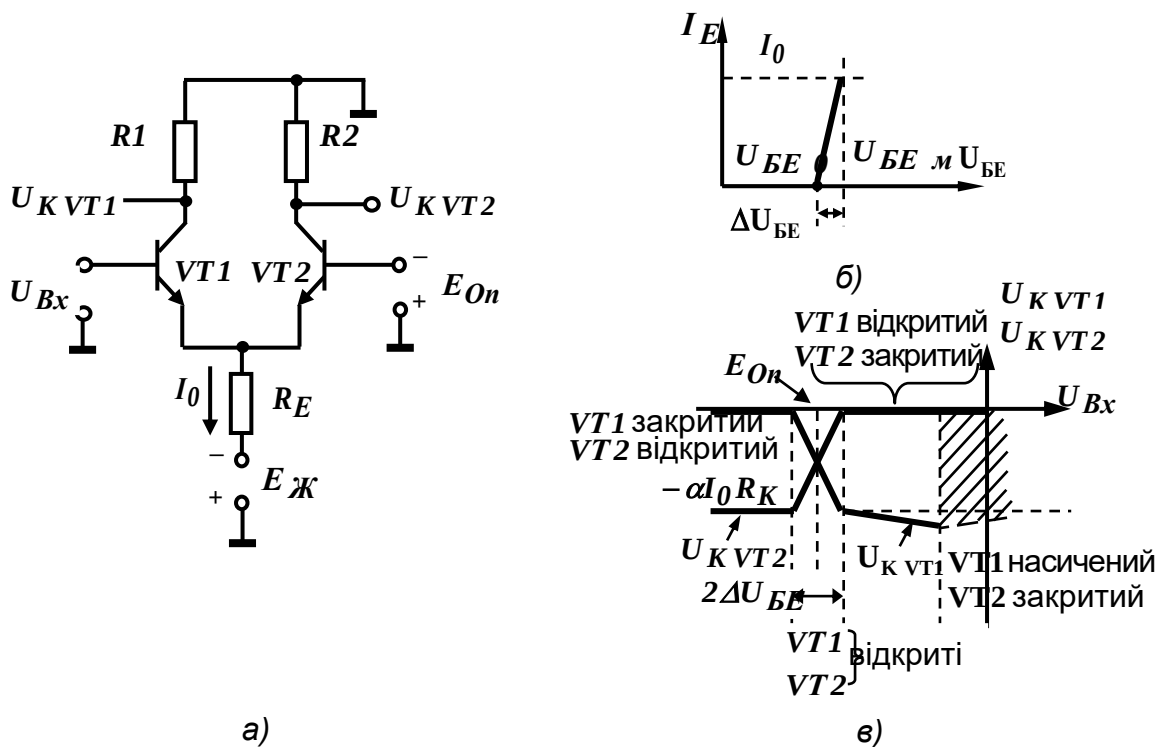


Рис. 2.11. Перемикач струму (а), апроксимовані залежності струмів емітерів (б) і напруг на колекторах (в) транзисторів перемикача

Якщо прийняти апроксимацію вхідної характеристики транзисторів, як показано на рис. 2.11б), то для повного перемикачання струму I_0 з одного плеча в друге достатньо зменшити напругу U_{BE} у відкритому транзисторі на величи-

ну ΔU_{BE} , а у закритому – збільшити на ΔU_{BE} . Отже, ширина активної ділянки передаточних характеристик (області перемикавання) диференційного каскаду (рис. 2.11 в)) складає $2\Delta U_{BE}$ (порядку $0,2...0,3В$). При $U_{Bx} \leq E_{On} - \Delta U_{BE}$ транзистор $VT1$ закритий, а $VT2$ відкритий, напруга $U_{K VT1} = 0$, $U_{K VT2} = -\alpha I_0 R2$.

При $U_{Bx} \geq E_{On} + \Delta U_{BE}$ $VT1$ відкритий, а $VT2$ закритий, відповідно $U_{K VT1} \approx -\alpha I_0 R1$, $U_{K VT2} = 0$. Зауважимо, що в останньому випадку при збільшенні U_{Bx} напруга $U_{K VT1}$ продовжує потроху спадати й після перемикавання транзисторів. Однак спад невеликий, тому коефіцієнт передачі каскаду зі спільним емітером на транзисторі $VT1$ з емітерним навантаженням R_E дорівнює приблизно $\frac{R1}{R_E}$, значно менше одиниці. Подальше збільшення U_{Bx} може привести до насичення транзистора $VT1$. Якщо прийняти умову насичення $U_{BK} \geq 0$, то на високий рівень вхідної напруги необхідно накласти обмеження:

$$U_{Bx_{max}} \leq -\alpha I_0 R1.$$

Таким чином, необхідні статичні стани перемикача струму забезпечуються при:

$$\left. \begin{array}{l} U_{Bx_{min}} \leq E_{On} - \Delta U_{BE} \\ E_{On} + \Delta U_{BE} \leq U_{Bx_{max}} \leq -\alpha I_0 R1 \end{array} \right\}. \quad (2.16)$$

Часові діаграми роботи перемикача представлені на рис.2.12.

Швидкодія перемикача струму може бути високою. Це обумовлено такими чинниками:

по-перше, відкриті транзистори не переходять в насичення, внаслідок чого в перехідних процесах відсутній етап розсіювання надлишкових зарядів;

по-друге, транзистор $VT2$ працює у режимі схеми зі спільною базою, при якій досягається найбільш висока швидкість його перемикавання;

по-третє, для перемикавання струму I_0 потрібні великі перепади вхідного сигналу. Вважаючи, що вони створюються аналогічними перемикачами струму, є можливість використати резистори в колі колектора R_K з малим опором (сот-

ні Ом), що значно зменшує вплив ємності колекторного переходу C_K транзисторів на час перемикавання [4].

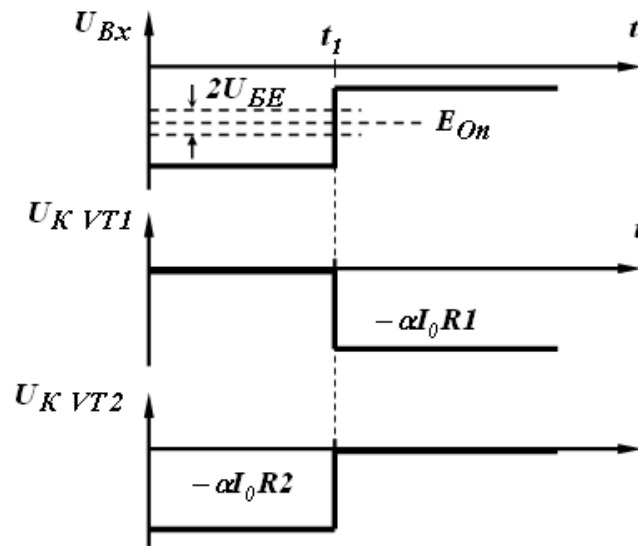


Рис. 2.12. Часові діаграми роботи перемикача струму

Загальна тривалість перехідного процесу при перемиканнях може складати десятки долі наносекунди. Однак, враховуючи, що найкращі частотні властивості транзисторів проявляються при значних колекторних струмах, споживана від джерела живлення потужність велика (десятки мВт).

2.3. Класифікація й основні параметри базових логічних елементів

У процесі розвитку мікросхемотехніки виникли кілька типів базових логічних елементів, які мають досить високі значення параметрів та зручні для інтегрального виконання. Класифікація їх наведена на рис. 2.13. Активними приладами є біполярні транзистори (БТ), МОН-транзистори (МОНТ) і транзистори з інжекційним живленням (ІТ).

Залежно від типу активних приладів, за допомогою яких виконується вхідна (І, ЧИ) і вихідна (І) операції, базові логічні елементи та цифрові ІС на їхній основі прийнято поділяти на такі типи:

- транзисторної логіки (ТЛ);
- діодно-транзисторної логіки (ДТЛ);
- транзисторно-транзисторної логіки (ТТЛ);
- інтегральної інжекційної логіки (І²Л).

Дана символіка пояснюється так: ТЛ – вхідна і вихідна операції виконуються на одних і тих же транзисторах; ТТЛ – вхідна операція виконується на

одних транзисторах, вихідна на інших; ДТЛ – вхідна операція виконується на діодах, вихідна на транзисторах.

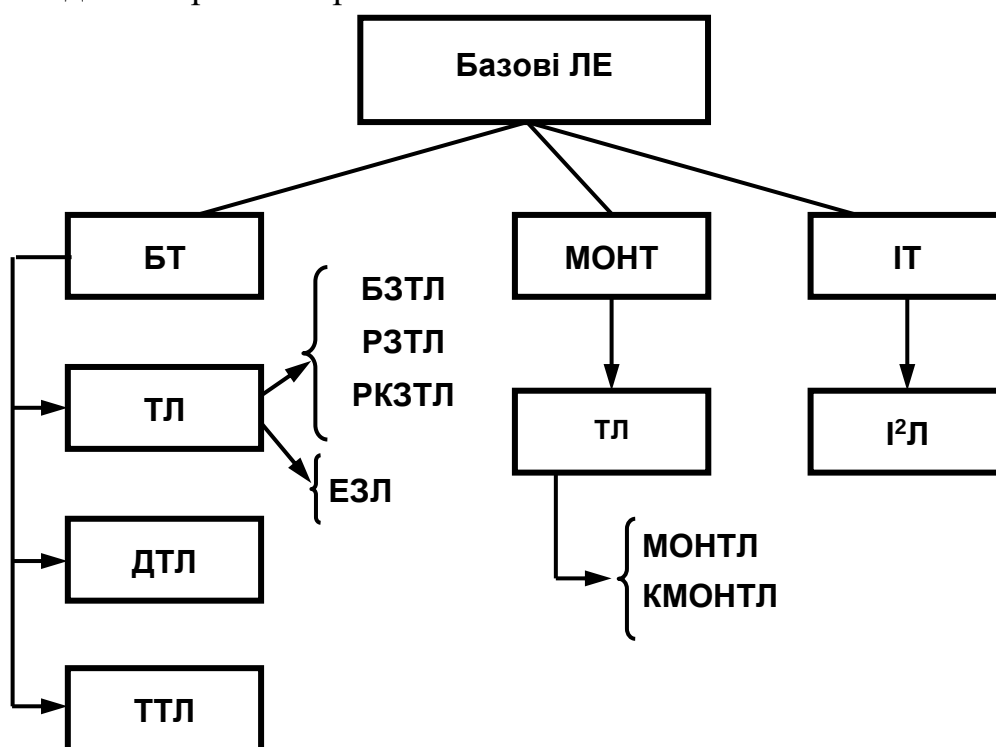


Рис. 2.13. Класифікація базових логічних елементів

Елементи транзисторної логіки поділяються ще за способом об'єднання електродів транзисторів на елементи з об'єднаними колекторами й об'єднаними емітерами (емітерно-зв'язана логіка – **ЕЗТЛ** або просто **ЕЗЛ**). Перші становлять набір ключів зі спільним емітером і зі спільним колекторним навантаженням. Якщо в колах баз транзисторів включені резистори або резистори із шунтуючими їх конденсаторами, то такі **БЛЕ** називають відповідно з резистивними зв'язками (**РЗТЛ** або **РТЛ**) і з резистивно-конденсаторними зв'язками (**РКЗТЛ** або **РКТЛ**), якщо без резисторів – зв'язок безпосередній (**БЗТЛ**).

Схеми на **МОН**-транзисторах реалізують транзисторну логіку і позначаються **n-МОНТЛ** або **p-МОНТЛ**, якщо транзистори однотипні, і **КМОНТЛ**, якщо комплементарні.

Варто мати на увазі, що інтегральні схеми **БЗТЛ**, **РЗТЛ**, **РКЗТЛ** не перспективні й у нових розробках комп'ютерної схемотехніки не використовуються. Схеми **ДТЛ** теж втратили практичну цінність для масового застосування, але вони бувають іноді корисні, наприклад, для захисту входів цифрових пристроїв від інтенсивних перешкод, при використанні довгих неузгоджених захищених сигнальних ліній та ін. Найбільш широке поширення у даний час одержали серії цифрових ІС **ТТЛ** (у тому числі з діодами **Шотткі** – **ТТЛШ**), **ЕЗЛ**,

I²Л, n-МОНТЛ і КМОНТЛ. Тому нижче розглядаються **БЛЕ** тільки таких типів.

Властивості **БЛЕ** і створювані на їх основі різні цифрові **ІС** оцінюються низкою електричних параметрів. Основними параметрами **ЛЕ** є такі.

Час затримки поширення сигналу в ЛЕ – характеризує швидкодію елемента. Розрізняють час затримки при вмиканні $t_{3m}^{1,0}$ і вимиканні $t_{3m}^{0,1}$. На практиці частіше використовують середню затримку, обумовлену як:

$$t_{3m_{сер}} = 0,5 (t_{3m}^{0,1} + t_{3m}^{1,0}) . \quad (2.17)$$

Часові інтервали $t_{3m}^{0,1}$ і $t_{3m}^{1,0}$ відраховуються звичайно на рівні 0,5 перепадів напруги на вході і виході **ЛЕ** (рис. 2.14).

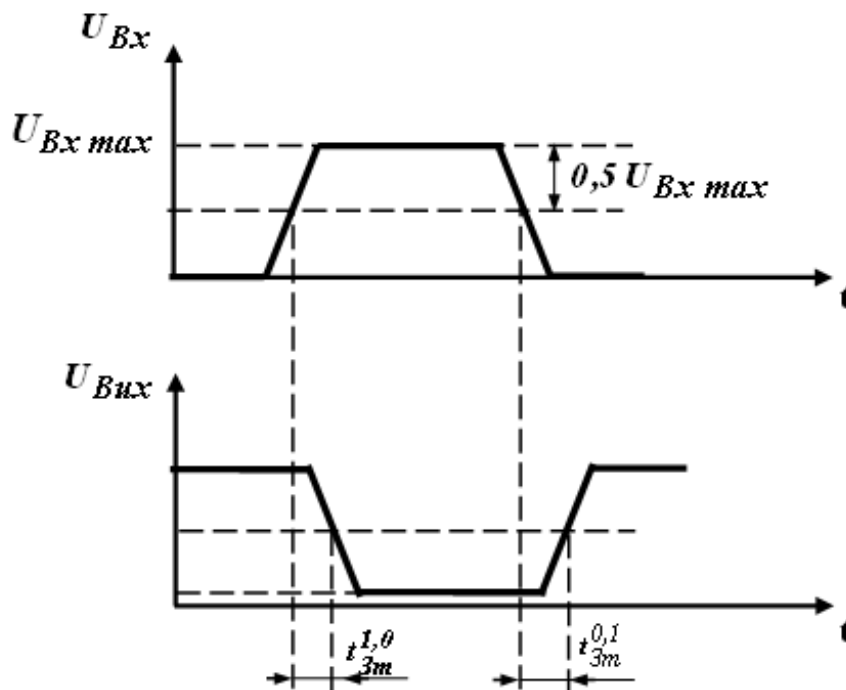


Рис. 2.14. Пояснення визначення часу затримки вмикання та вимикання

Споживаною від джерела живлення потужністю характеризується економічність **ЛЕ**. Розрізняють споживану потужність у стані логічної одиниці P_{Cn}^1 , логічного нуля P_{Cn}^0 і середню:

$$P_{Cn_{сер}} = 0,5 (P_{Cn}^1 + P_{Cn}^0) . \quad (2.18)$$

Для однотипних **БЛЕ** добуток

$$A_{Сер} = P_{Cn_{Сер}} t_{3m_{Сер}} \quad (2.19)$$

названий середньою роботою перемикавання, майже постійний у широкому діапазоні значень $P_{Cn_{Сер}}$ та $t_{3m_{Сер}}$. Отже, елемент із підвищеною роботою перемикавання $P_{Cn_{Сер}}$ має меншу затримку перемикавання $t_{3m_{Сер}}$ і навпаки. Для різнотипних **БЛЕ** робота перемикавання може істотно відрізнятися. Таким чином, $A_{Сер}$ є своєрідним показником якості типу **БЛЕ**.

Напруга логічного нуля U^0 і логічної одиниці U^1 – значення високого і низького рівнів напруги на виході **БЛЕ** в двох логічних станах. Різницю цих рівнів називають логічним перепадом U_m .

Вхідний струм логічного нуля $I_{Вх}^0$ і логічної одиниці $I_{Вх}^1$ – максимальне значення струмів у вхідних колах **ЛЕ** при подачі на них відповідно рівнів U^0 і U^1 .

Коефіцієнт розгалуження по виходу $K_{Роз}$ – вказує на максимальну кількість входів **ІС** даної серії, які одночасно можуть підключатися до виходу **ЛЕ** без порушення його нормального функціонування.

Коефіцієнт об'єднання по входу $K_{Об}$ – визначає кількість входів, по яких реалізується логічна функція.

Для сучасних **БЛЕ** типові значення параметрів складають:

$P_{Cn_{Сер}}$ – десятки частки мікроват ... десятки міліват;

$t_{3m_{Сер}}$ – одиниці ... сотні наносекунд;

U_m – десятки частки ... одиниці вольт;

$K_{Роз}$ – 4...30;

$K_{Об}$ – 2...5.

Поряд зі статичними параметрами практичний інтерес мають також статичні характеристики: передаточна $U_{Вих} = f(U_{Вх})$, вхідна $I_{Вх} = f(U_{Вх})$ і вихідна $U_{Вих} = f(I_{Вих})$. Характеристики є більш інформативними для аналізу та проектування цифрових **ІС** і пристроїв на їх основі. Вони дозволяють не тільки визначати більшість статичних параметрів, але й виявляти ряд особливих властивостей. Так, наприклад, за допомогою передаточної характеристики, типовий вигляд якої показаний на рис. 2.15, можна отримати значення U^0 і U^1 , а також

область значень вхідної напруги, у межах якої відбувається перемикання БЛЕ, тобто порогову область ($U_{Пор}^0 \dots U_{Пор}^1$). Це, у свою чергу, дозволяє оцінити статичну завадостійкість БЛЕ, тобто здатність протистояти впливу сигналу, що заважає, тривалість якого значно перевершує час затримки $t_{3m_{сер}}$. Допустима вмикаюча завада визначається як $U_{П}^0 = U_{Пор}^0 - U^0$, а допустима вимикаюча завада $-U_{П}^1 = U^1 - U_{Пор}^1$.

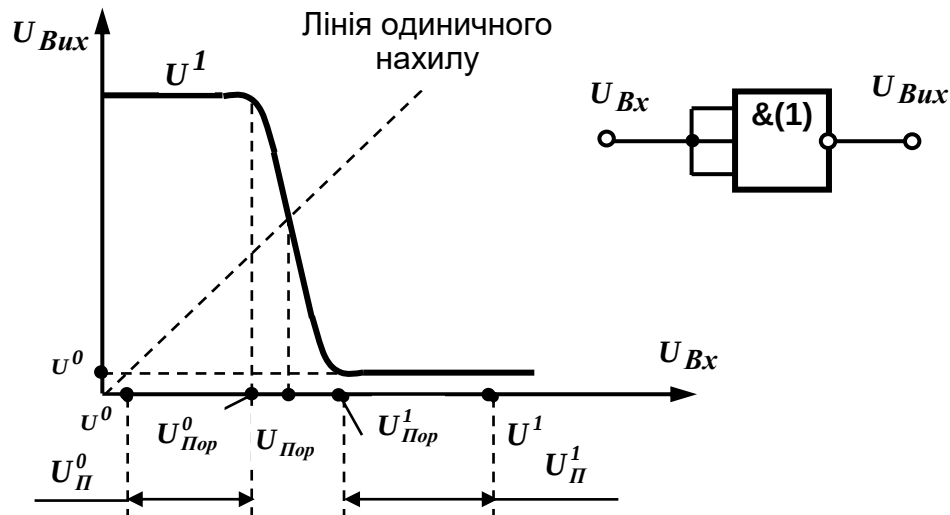


Рис. 2.15. Передаточна характеристика БЛЕ

Знайшло поширення також поняття порогової напруги $U_{Пор}$. Її значення відповідає точці перетину лінії одиничного нахилу з передаточною характеристикою. Для більшості БЛЕ область перемикання вузька ($0,1 \dots 0,2 В$). Тому іноді вважають $U_{Пор} \approx U_{Пор}^0 \approx U_{Пор}^1$.

2.4. Базовий логічний елемент транзисторно-транзисторної логіки

Базовий елемент ТТЛ (рис. 2.16) містить багатоемітерний транзистор $VT1$, який разом з резистором $R1$ виконує операцію I , та складний інвертор, що становить послідовно включені ключ із спільним емітером (простий інвертор) на $VT2$ і $R2$ та емітерний повторювач на $VT4$ з динамічним навантаженням VD і $VT3$. Резистор $R3$ забезпечує коло для струму розсіювання надлишкового заряду з бази $VT3$ при його запиранні. Резистор $R4$ обмежує струм транзистора $T4$ при перевантаженні та при вимиканні логічного елемента, коли $VT4$ уже відкрився, а $VT3$ ще не закритий. Діод VD , як буде показано нижче, забезпечує закритий стан транзистора $VT4$ при відкритому $VT3$.

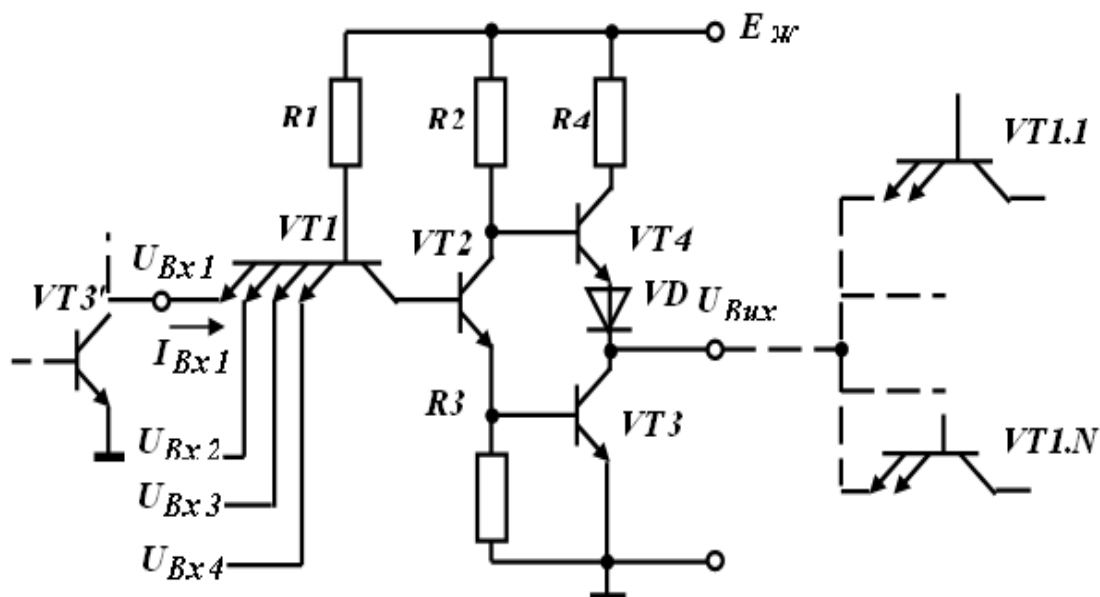


Рис. 2.16. Базовий елемент ТТЛ

Багатоемітєрний транзистор (БЕТ) відрізняється від звичайних біполярних транзисторів наявністю декількох (як правило чотирьох) емітерів (рис. 2.17), об'єднаних спільною базою. Емітери розташовані так, що безпосередня взаємодія між ними через область бази практично відсутня. Тому *багатоемітєрний транзистор* можна розглядати як сукупність декількох незалежних транзисторів з об'єднаними колекторами та базами. Але на відміну від такого складання він займає меншу площу на кристалі і має малі паразитні ємності. Крім того, в інверсному режимі коефіцієнт передачі струму β_i дуже малий ($0,05...0,005$) [3].

Розглянемо роботу логічного елемента при керуванні вхідними сигналами з виходів аналогічних ЛЕ. Для спрощення аналізу й оцінок основних параметрів приймемо апроксимацію вхідних характеристик транзисторів, як показано на рис. 6.3а), та будемо вважати падіння напруги на насиченому транзисторі (U_{KE_H}) і його переходах (U_{BE_H} , U_{BK_H}) незалежними від струмів і рівними відповідно $0,1$ В та $0,7$ В.

Якщо на всі входи ЛЕ подана напруга логічного нуля $U^0 = U_{KE_H}$ (транзистори $T3'$ ЛЕ, які задають вхідний сигнал, насичені), то струм бази I_{B1}^0 транзистора $VT1$ протікає через відкриті емітерні переходи. При цьому на його базі присутня напруга $U_{B1}^0 = U^0 + U_{BE_H}$.

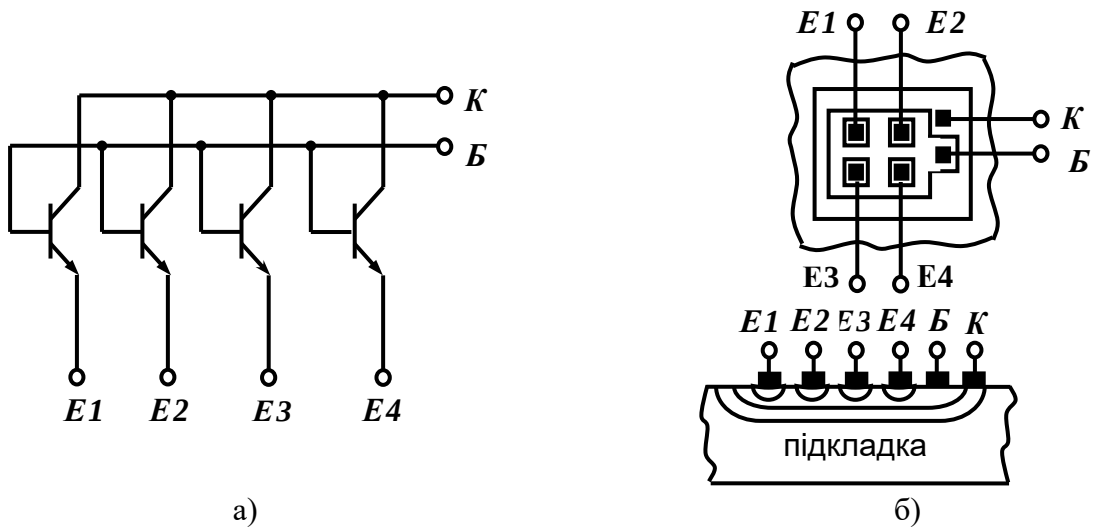


Рис.2.17. Електрична модель (а) і структура (б) багатоемітерного транзистора

Цієї напруги недостатньо для відмикання двох послідовно з'єднаних переходів: колекторного **VT1** і емітерного **VT2**. Тому транзистори **VT2** і **VT3** закриті. Напруга на колекторі другого транзистора

$$U_{KVT2} = E_{Ж} - I_{BVT4}R2 \approx E_{Ж}. \quad (2.20)$$

Транзистор **VT4** і діод **VD** відкриті. На виході елемента

$$U_{Вих} = U^1 = U_{KVT2} - U_{BEVT4} - U_{VD} \approx E_{Ж} - 2U_{BEH}. \quad (2.21)$$

При $E_{Ж} = 5B$ величина $U^1 \approx 3,6B$.

Помітимо, що транзистор **VT4** працює в активному режимі як емітерний повторювач. Активний режим забезпечується тим, що в штатних сталих умовах напруга на колекторному переході

$$U_{BKVT4} = I_{KVT4}R4 - I_{BVT4}R2 \quad (2.22)$$

з огляду на малість струму I_{BVT4} й опору **R4** близька до нуля, перехід закритий.

Транзистор **VT1** насичений, тому що

$$I_{BVT1}^0 = \frac{E_{Ж} - U_{BEHVT1} - U^0}{R1} \gg I_{BKPVT1} = \frac{I_{KHVT1}}{\beta_1} \approx 0. \quad (2.23)$$

Вимкнений стан логічного елемента зберігається, якщо хоча б на одному вході діє сигнал U^0 . При цьому струм бази I_{BVT1}^0 лише перерозподіляється по

вхідних колах. При подачі рівня U^0 тільки на один вхід у його колі тече максимальний струм $I_{Bx}^0 \approx -I_{BVT1}^0$, який отримав назву вхідного струму логічного нуля.

Якщо на всі входи подана висока напруга (U^1), емітерні переходи $VT1$ закриті й струм бази I_{B1}^1 тече через колекторний перехід у базу транзистора $VT2$. Багатоемітерний транзистор знаходиться в активному інверсному режимі. В емітерних колах протікають вхідні струми логічної одиниці $I_{Bx}^1 = \beta_i I_{BVT1}^1$. Ці струми підсумовуються зі струмом I_{BVT1}^1 і утворюють вмикаючий струм бази транзистора $VT2$:

$$I_{BVT2} = I_{BVT1}^1 + m\beta_i I_{BVT1}^1 \cong I_{BVT1}^1, \quad (2.24)$$

де m – кількість задіяних входів ЛЕ. Невелике значення інверсного коефіцієнта передачі струму β_i дозволяє виключити вплив кількості входів ЛЕ на струм I_{BVT2} , отже, на ступінь насичення $VT2$, а також забезпечує розв'язку між джерелами вхідних сигналів. Емітерний струм транзистора $VT2$ тече в основному через базу транзистора $VT3$ і насичує його. Напруга на його колекторі $U_{Bux} = U_{KE_H} = U^0 = 0,1B$.

При цьому транзистор $VT3$ закритий, тому що різниці

$$U_{KVT2} - U_{Bux} = U_{BE_H VT3} + U_{KE_H VT2} - U_{KE_H VT3} \approx U_{BE_H} \quad (2.25)$$

недостатньо для відмикання двох послідовно з'єднаних переходів: емітерного $VT4$ і діода VD . Отже, призначення діода – забезпечити закритий стан $VT4$ при відкритому $VT3$.

Таким чином, при позитивному кодуванні ЛЕ виконує операцію **НІ-І** (вмикається тільки при подачі на усі входи високого рівня напруги). Він має невеликий вихідний опір як у вимкненому, так і в увімкненому станах, обумовлений відповідно вихідним опором емітерного повторювача на $VT4$ і опором насиченого транзистора $VT3$. Це сприяє зменшенню часу перезаряджання ємності навантаження. Підвищенню швидкодії допомагають також сприятливі умови роботи транзисторів. Так, при перемиканні транзистора $VT1$ його базовий струм і заряд у базі практично не змінюються, що виключає інерційні процеси накопичування та розсіювання надлишкового заряду. Транзистори ж $VT2$ і

VT3 відкриваються і насичуються великими вмикаючими струмами ($I_B \gg I_{B_{кр}}$), а на стадії закривання для вимикаючих струмів створюються досить низькоомні базові кола: для **VT2** це насичений транзистор **VT1**, для **VT3** – резистор **R3**.

Більш детальну інформацію, необхідну для проектування різних пристроїв, дають статичні характеристики ЛЕ. Найбільше поширення одержали передаточна і вхідна характеристики. Для розглянутого ЛЕ вони мають вигляд, показаний на рис. 2.18.

На характеристиках можна виділити три ділянки. На ділянці **AB** елемент вимкнений, **BD** – працює в підсилювальному режимі і **DF** – увімкнений. На ділянці **AB** ($U_{Bx} = 0 \dots U_{BEH}$) напруга на базі транзистора **VT2** менше напруги, яка відмикає. Тому він закритий, і напруга $U_{Bux} = U^1 = \text{const}$. Вхідний струм зі зростанням U_{Bx} зменшується:

$$I_{Bx} = \frac{E_{Ж} - U_{BEH \text{ VT1}} - U_{Bx}}{R1}. \quad (2.26)$$

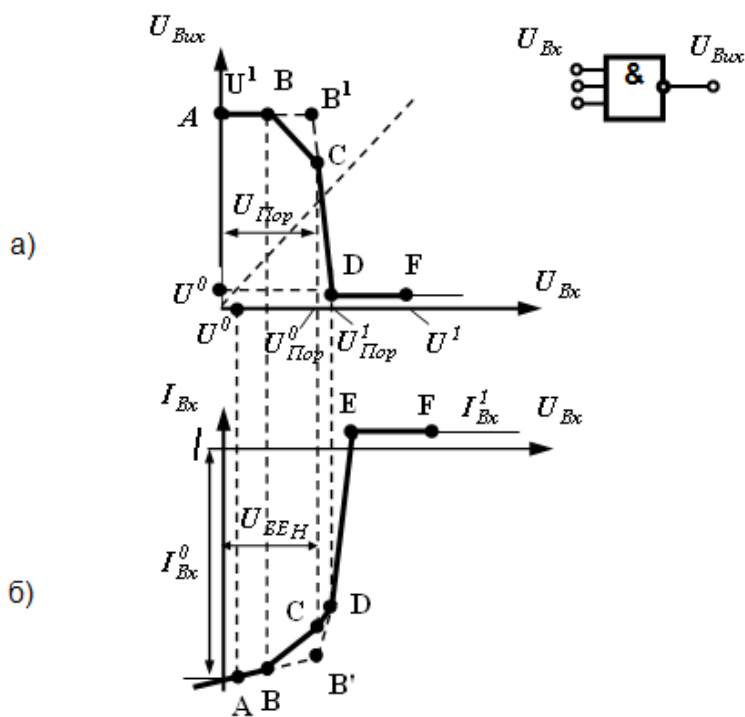


Рис. 2.18. Апроксимовані передаточна (а) та вхідна (б) характеристики базового ТТЛ елемента

На ділянці **BC** ($U_{Bx} \approx U_{BE_H} \dots 2U_{BE_H}$) транзистор **VT2** працює в активному режимі з коефіцієнтом передачі напруги $K_2 \approx \frac{R_2}{R_1} \approx 1$ (транзистор **VT2** закритий). Зі зростанням U_{Bx} напруга U_{K2} , а отже, і U_{Bux} зменшуються. Частина струму бази **VT1** відгалужується в базу **VT2**, тому спад I_{Bx} крутіше. На ділянці **CD** транзистор **VT3** відкритий і працює в активному режимі. Крутість характеристики значно більше, а сама ділянка **CD** вузька (не більш **0,1B**), тому що коефіцієнт передачі K_2 великий. Останнє пояснюється тим, що в колі емітера **VT2** опір **R3** шунтується невеликим вхідним опором відкритого транзистора **VT3**. У точці **VD** транзистори **VT2** і **VT3** насичені, і подальше підвищення U_{Bx} призводить лише до повного перемикання струму бази **VT1** у коло бази **VT2** (ділянка **DE**), збільшуючи ступінь насичення останнього.

При зміні напруги джерела живлення E_J і температури навколишнього середовища характеристики дрейфують, утворюючи область невизначеності. У зв'язку з цим необхідно відзначити, що електричні параметри **ЛЕ**, які наводяться в довідковій літературі, відповідають границям цієї області, тобто гіршим умовам застосування. На практиці, якщо умови застосування задані, доцільно оцінювати та використовувати реальні значення параметрів. Наприклад, рівень U^1 вважається як $E_J - 2U_{<BE_H}$, а порогів як $U_{Пор} \cong 2U_{BE_H}$. Причому, якщо при кімнатній температурі $U_{BE} \approx 0,8 \text{ В}$, то при $T = 100^\circ \text{C}$ за рахунок температурного дрейфу $U_{BE_H} = 0,55 \dots 0,6 \text{ В}$.

У даний час на основі **ТТЛ** елементів використовується велика кількість серій **ІС**. Принципові схеми базових **ЛЕ** в них практично однакові і відрізняються від наведеної на рис. 2.16, головним чином, номіналами опорів резисторів і деякими схемними особливостями, які ілюструються на рис. 2.19 і 2.20. Так, паралельно входам підключені діоди **D1**, **D2**, які отримали назву антидзвонівих. Вони обмежують завадові викиди негативної полярності, які утворюються у результаті взаємних наведень у сигнальних колах. Обмеження необхідне для захисту від пробую емітерних переходів **VT1**, а також для підвищення завадостійкості цифрових пристроїв шляхом "гасіння" коливальних процесів у електричне зв'язаних сигнальних колах.

Для **ЛЕ** середньої і підвищеної споживаної потужності, а також елементів **ТТЛШ** типовим характерним схемотехнічним рішенням є підключення в коло

емітера **VT2** динамічного навантаження, яке складається з **R3**, **R5** і **VT5**. Воно дозволяє отримати форму передаточної характеристики **ЛЕ** близьку до прямокутної (рис. 2.18, пунктирна лінія) за рахунок звуження області підсилювального режиму (**B'C**), тому що транзистор **VT2** може відкриватися лише при $U_{Bx} \approx 2U_{BEH}$. Тим самим підвищується завадостійкість до відмикаючої завади.

Динамічне навантаження призводить до погіршення умов розсмоктовування надлишкових зарядів у базі **VT3** при вимиканні. Тому паралельно емітерному переходу **VT2** підключають діод **VD3** (рис. 2.19), який забезпечує низькоомне коло для базового вимикаючого струму транзистора **VT3**: база **VT3** – діод **VD3** – насичений транзистор **VT1** – джерело сигналу.

І нарешті, з метою підвищення навантажувальної здатності **ЛЕ** замість діода зсуву **VD4** застосовують транзистор **VT6** (рис. 2.20), який разом з **VT4** утворює складений транзистор.

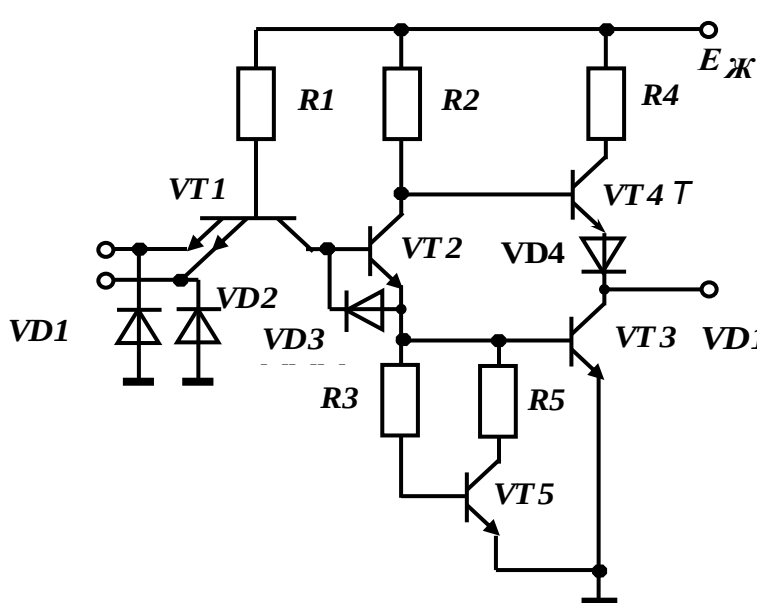


Рис.2.19. Модифікований базовий елемент ТТЛ

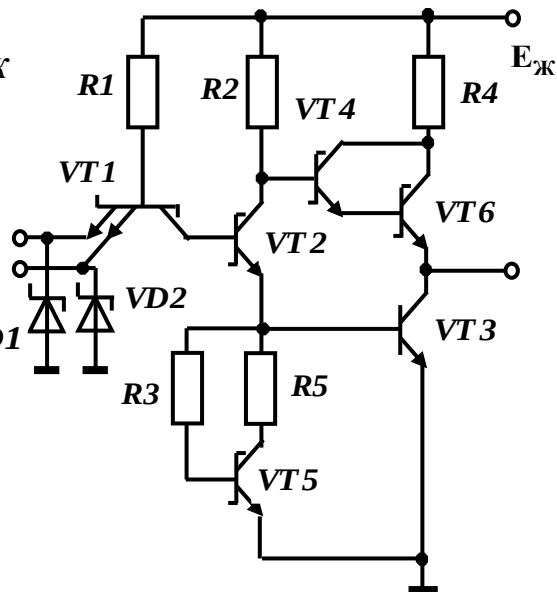


Рис.2.20. Базовий елемент ТТЛШ

Реалізація цифрових **ІС** і пристроїв тільки на основі базових **ЛЕ** звичайно не є більш раціональною – не забезпечує простоту та найкращі електричні параметри. Більш прийнятні результати одержуються при використанні поряд з базовими **ЛЕ** допоміжних елементів, які виконують функції **НІ-І**; **НІ-ЧИ**; **І**; **ЧИ**; **НІ**; **ВИКЛЮЧАЛЬНЕ ЧИ**; з відкритим колекторним виходом; із трьома статичними станами та ін. Такі елементи створюються найчастіше шляхом зміни топології з'єднань на кристалі базового **ЛЕ**.

Розглянемо приклади допоміжних ЛЕ, які зазвичай є в складі серій цифрових ІС. На рис. 2.21 наведена схема ЛЕ НІ-І з відкритим колектором. Для нормальної роботи вихідного транзистора $VT3$ його колектор підключають до джерела живлення через зовнішнє навантаження: резистор, елемент індикації, реле, ЛЕ тощо. Напруга джерела може відрізнятися від штатної. Таким чином, елементи з відкритим колектором забезпечують роботу з нетиповим навантаженням.

З іншого боку, елементи з відкритим колектором допускають паралельне підключення декількох виходів до спільного навантаження (рис. 2.22).

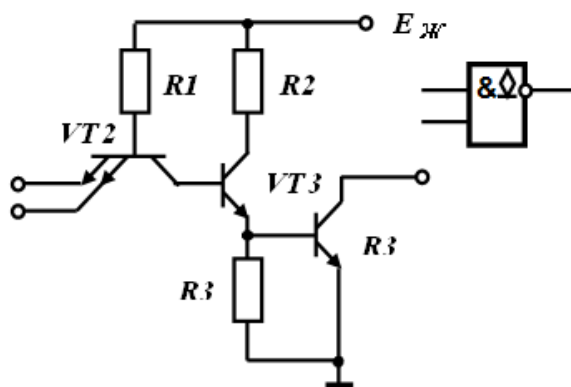


Рис. 2.21. Елемент ТТЛ НІ-І з відкритим колекторним виходом

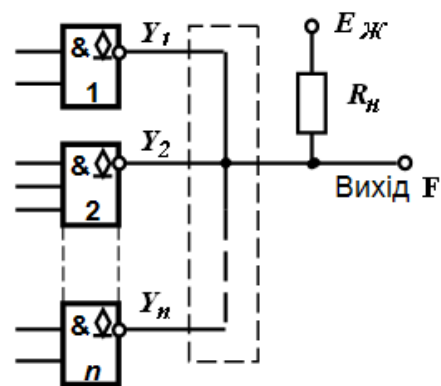


Рис. 2.22. Монтажна логіка І на елементах з відкритим колекторним виходом

Об'єднання виходів називають монтажною логікою. При такому з'єднанні, якщо на виході одного з декількох елементів буде низька напруга (транзистор $VT3$ насичений), то вона ж виявиться на спільному виході. Щоб забезпечити логічну одиницю на спільному виході, необхідно мати логічні одиниці на усіх виходах (закритий стан транзисторів $VT3$ всіх елементів).

Це еквівалентно операції І над вихідними сигналами. У даному прикладі монтажне І визначається за формулою $F = Y_1 Y_2 \dots Y_n$.

При створенні монтажної логіки варто враховувати, що кожна змінна Y втрачає самостійність і використовуватися окремо вже не може.

Якщо паралельно транзисторові $VT2$ у базовому ЛЕ підключити ще транзистор $VT2'$, керований окремим багатоемітерним транзистором $VT1'$ (рис. 2.23), то вийде елемент НІ-І-ЧИ, який реалізує функцію $Y = \overline{x_4 x_3 \vee x_2 x_1}$.

Дійсно, щоб на виході був логічний нуль, необхідно відкрити $VT2$ або $VT2'$, для чого потрібно подати логічні одиниці на всі емітери хоча б одного з транзисторів $VT1$ або $VT1'$.

Помітимо, що якщо транзистори $VT1$ і $VT1'$ мають по одному емітеру, елемент реалізує операцію **НІ-ЧИ**.

Елементи, що випускаються окремо в складі серії, які містять багатоемітерний транзистор і транзистор інвертора (рис. 2.24), отримали назву **РОЗШИРНИКИ ПО ЧИ**.

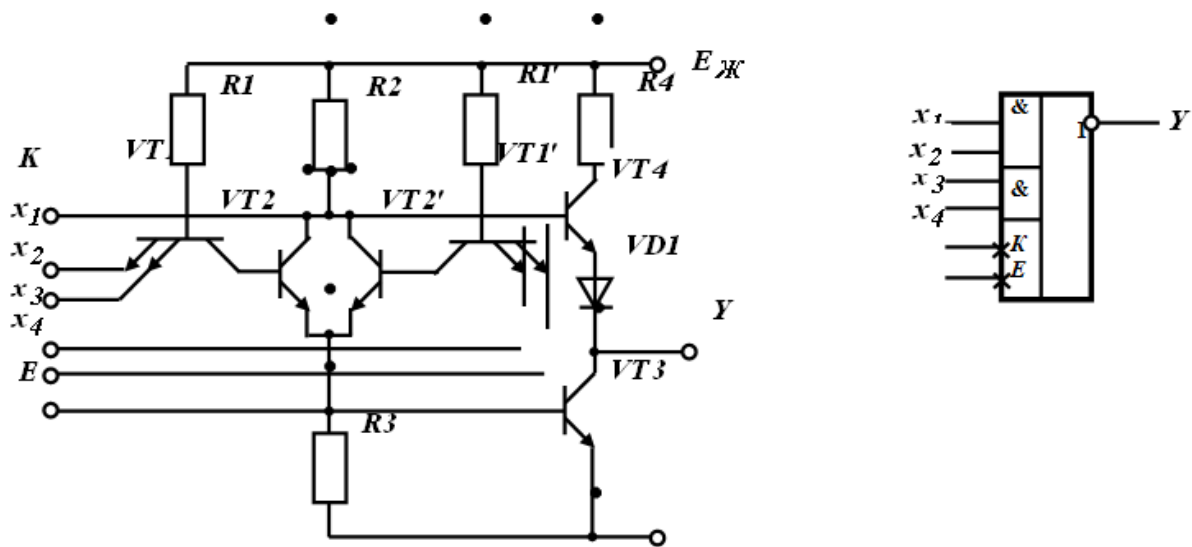


Рис. 2.23. Елемент НІ-І-ЧИ ТТЛ

Вони своїми виводами K та E підключаються до однойменних виводів (якщо такі передбачені) логічних елементів **НІ-І**, **НІ-І-ЧИ** (наприклад, рис. 2.23) і тим самим розширюють функціональні можливості останніх по **ЧИ**.

На рис. 2.25 показаний приклад схеми елемента, який виконує операцію **І**. У схемотехнічному плані він становить кон'юнктор на **БЕТ** і два послідовно з'єднаних інвертори.

На рис. 2.26 наведена схема елемента, здатного приймати три статичних стани. Два з них – звичайні (увімкнений, вимкнений), а третій – високоімпедансний, за якого вихідні транзистори $VT3$ і $VT4$ закриті, що рівнозначно відключенню виходу від навантаження. Переведення у високоімпедансний стан здійснює додатковий транзистор $VT5$ при подачі на керуючий вхід E напруги логічної одиниці незалежно від значення вхідного інформаційного сигналу.

Застосування трьох стабільних елементів дозволяє по черзі підключати джерела сигналів до спільного навантаження або до інформаційної шини й у та-

кий спосіб ущільнювати канали передачі даних (рис. 2.27 а)), а також створювати магістралі з двонаправленим потоком інформації (рис. 2.27 б)).

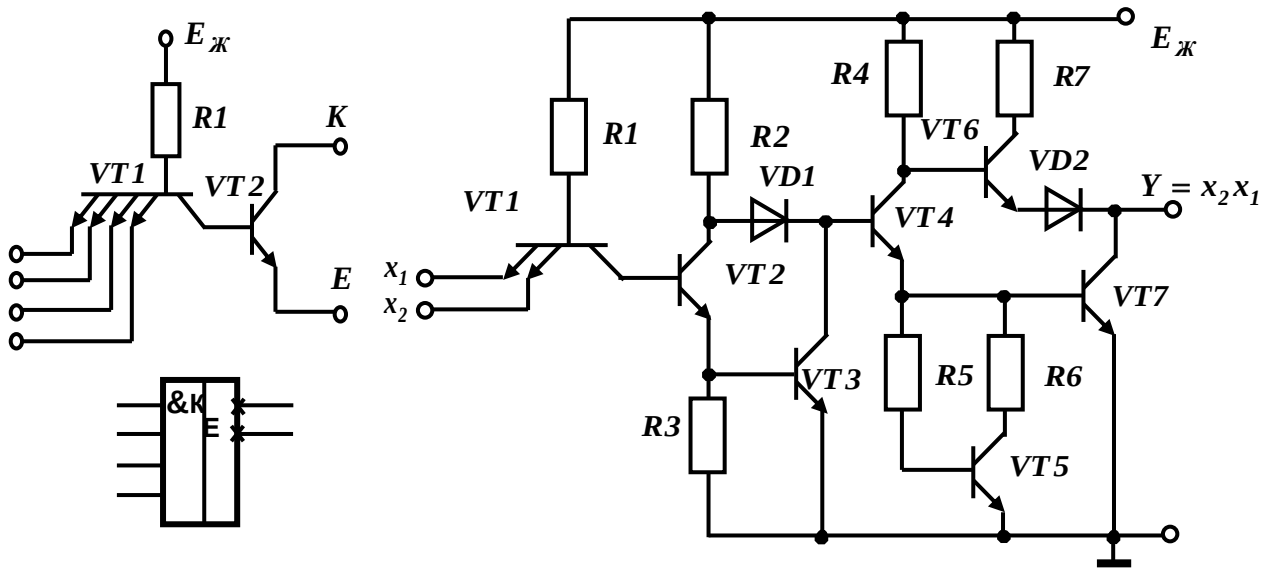


Рис. 2.24. Розширник по ЧИ

Рис. 2.25. Елемент ІТТЛ

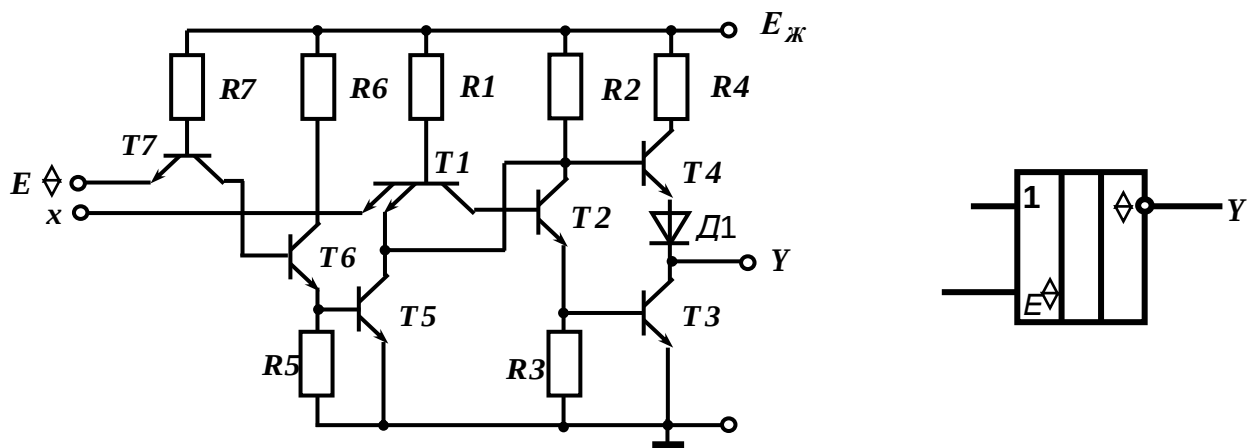


Рис. 2.26. Елемент з трьома станами

Помітимо, що високоімпедансний стан сам по собі не є логічним, але керуючий сигнал, який створює його, може визначати логічний рівень на виході пристрою (ІС), до складу якого входить даний елемент. Наприклад, схема, зображена на рис. 2.27 а), реалізує функцію $Y = \bar{x}_1 E \diamond \vee \bar{x}_2 \overline{E \diamond}$.

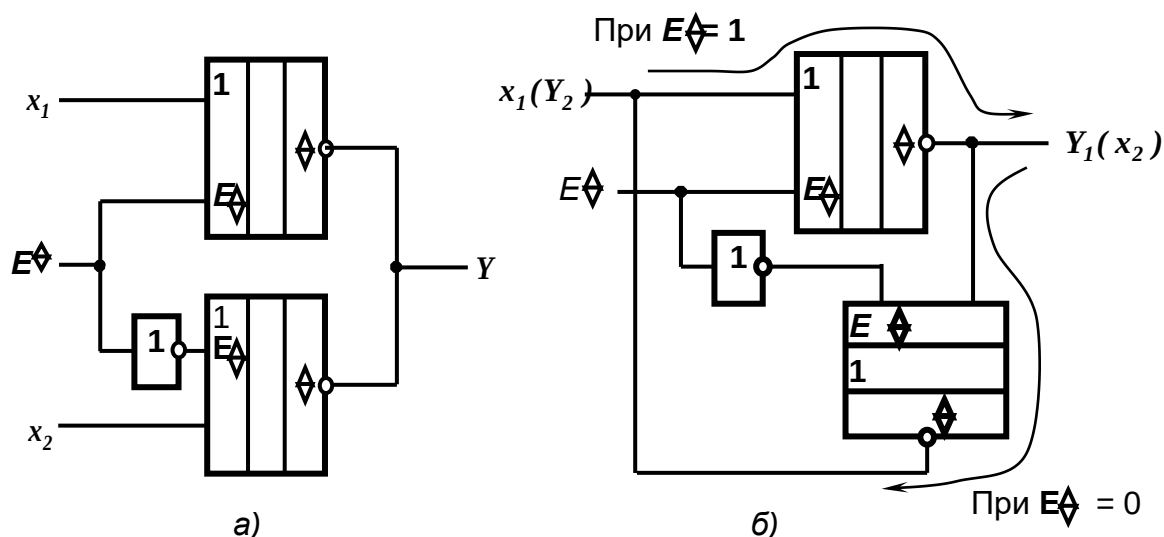


Рис. 2.27. Однонаправлений на два входи (а) та двонаправлений на один вхід (б) комутатори

2.5. Базовий логічний елемент емітернозв'язаної логіки

Елементи **ЕЗЛ** будують на основі перемикача струму (рис. 2.11 а)).

Для керування за декількома входами в одне плече перемикача паралельно вмикають декілька транзисторів (на рис. 2.28 це $VT1$ і $VT3$).

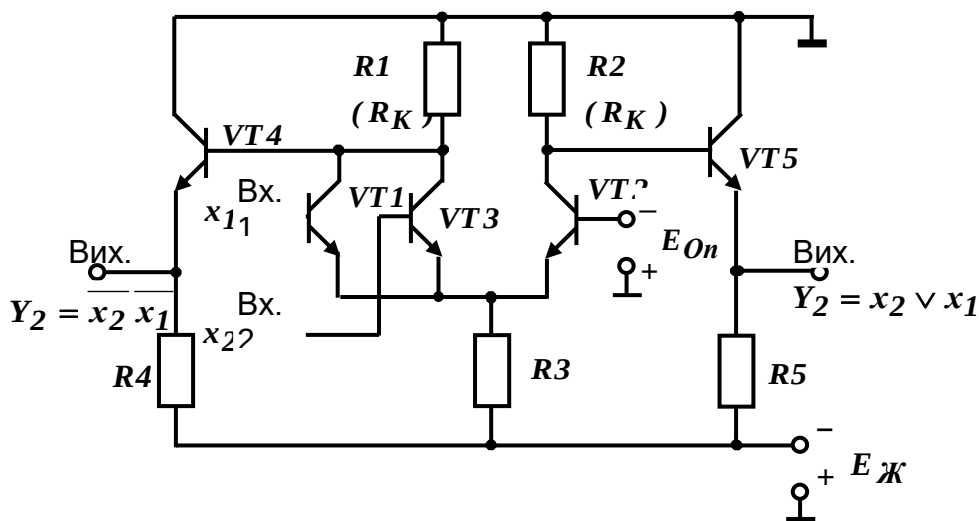


Рис. 2.28. Спрощена схема елемента ЕЗЛ

До складу логічного елемента входять також два емітерні повторювачі на транзисторах $VT4$ і $VT5$. Вони забезпечують низький вихідний опір і узгодження вихідних логічних рівнів із входніми рівнями, необхідними для роботи перемикача без режиму насичення транзисторів.

Розглянемо роботу **ЕЗЛ** елемента. Якщо на обох входах діє низький рівень напруги $U_{Bx_{min}}$, що відповідно до (2.16) забезпечує закритий стан транзи-

сторів $VT1$ та $VT3$ і відкритий стан $VT1$, то потенціал колекторів $VT1$ та $VT3$ дорівнює нулю і на першому виході ЛЕ буде високий рівень напруги:

$$U_{Vux_{max}} = -U_{BE_M}, \quad (2.27)$$

який визначається падінням напруги на емітерному переході транзистора $VT4$. На колекторі транзистора $VT2$ напруга $U_{K_{VT2}} = -\alpha I_0 R_K$ (рис. 2.11 в)) передається на другий вихід ЛЕ:

$$U_{Vux_{min}} = -\alpha I_0 R_K - U_{BE_M}. \quad (2.28)$$

При подачі хоча б на один вхід високої напруги $U_{Vx_{max}}$ (2.16) загальний струм I_0 перемикається в ліве плече, і на виходах рівні напруги змінюються на протилежні: на першому виході – високий рівень, на другому – низький.

Зі сказаного вище випливає, що ЛЕ по першому виходу виконує операцію **НІ-І**, по другому – **ЧИ**.

Для оцінки логічних рівнів U^0 і U^1 варто виходити з того, що вони повинні бути однаковими на входах і виходах елемента. Відповідно до виразів (2.11) і (2.27) рівність

$$U^1 = U_{Vx_{max}} = U_{Vux_{max}} = -U_{BE_M} \quad (2.29)$$

забезпечується при $\alpha I_0 R_K \leq U_{BE_M}$. Причому найбільший логічний перепад $U_{Vux_{max}} - U_{Vux_{min}}$ (найбільша завадостійкість) виходить при $\alpha I_0 R_K = U_{BE_M}$. Враховуючи це, з (2.28) випливає, що

$$U^0 = U_{Vx_{min}} = U_{Vux_{min}} = -2U_{BE_M}. \quad (2.30)$$

Опорна напруга $E_{On} = -1,5U_{BE_M}$.

Логічний перепад у ЕЗЛ елемента $U_M = U^1 - U^0 = U_{BE_M}$ малий, складає близько **0,8 В**. Це обумовлює невисоку завадостійкість, але в той же час сприяє підвищенню швидкодії (швидше відбувається перезарядження паразитних ємностей). З огляду на те, що цільовим призначенням ЕЗЛ елемента є забезпечення максимальної швидкодії, властиві йому низька завадостійкість, велика споживана потужність і несумісність за логічними рівнями з іншими базовими ЛЕ у ряді випадків стають не визначальними для застосування недоліками.

На рис. 2.29 наведена схема базового ЛЕ серії **100 (K500)**. Усі входи ЛЕ через резистори витоку $R4, R5$ підключені до джерела живлення.

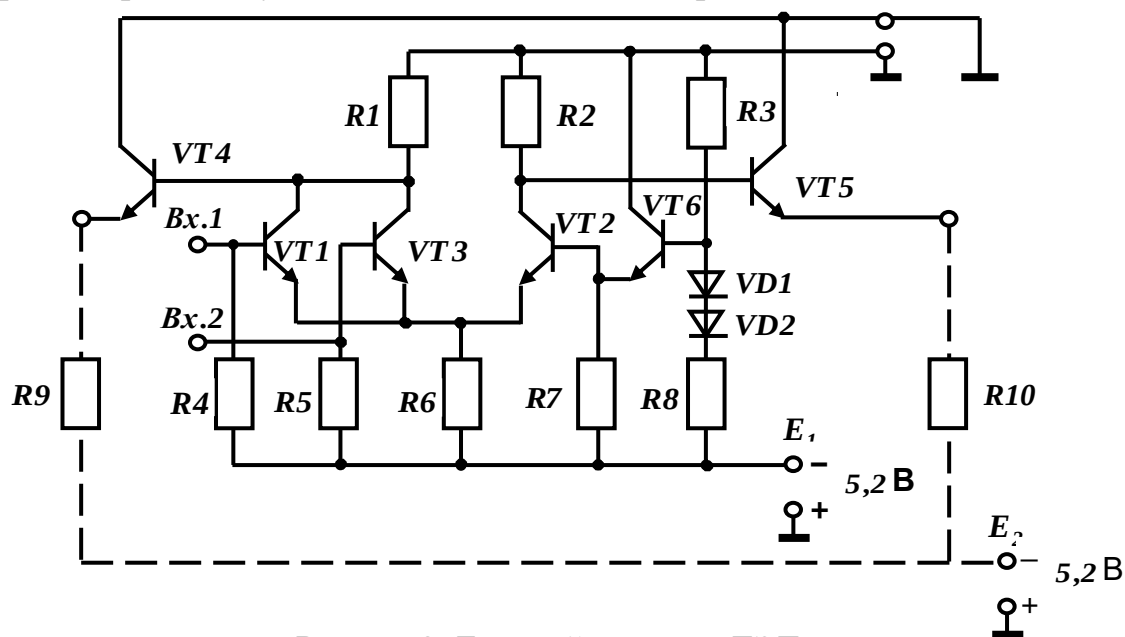


Рис. 2.29. Базовий елемент ЕЗЛ

Таке включення дозволяє залишати невикористовуванні входи вільними. Для ослаблення впливу імпульсних завад, які виникають у колекторних колах емітерних повторювачів при роботі на низькоомне навантаження, використовуються дві шини “корпус”: одна для емітерних повторювачів, інша для перемикача струму. Причому шини виготовляють з великим перерізом. Емітерні повторювачі можуть навантажуватися на зовнішні (у складі окремої ІС) низькоомні резистори $R9, R10$ і живитись від окремого джерела E_2 . З метою зменшення споживаної потужності до не використовуваних виходів зовнішні резистори не підключаються. Опорна напруга створюється термокомпенсованою схемою ($VT6, VD1, VD2, R3, R7, R8$) [5].

Як і в серіях **ТТЛ**, поряд з базовими ЛЕ до складу серій **ЕЗЛ** включаються допоміжні ЛЕ.

2.6. Елементи на МОН-транзисторах

2.6.1. Елементи на однотипних МОН-транзисторах

Серіям цифрових ІС на однотипних **МОН**-транзисторах властива наявність двох базових елементів, хоча основа в них спільна – ключ (рис. 2.8 а)). Подвійність обумовлена бажанням розширити вихідні функціональні можливості елементів у зв’язку з простотою й однорідністю компонентів, які складаються з набору керованих транзисторів і спільного навантаження (некерованого транзистора). Якщо керовані транзистори з’єднати паралельно, то ЛЕ викону-

ють операцію **НІ-ЧИ**, якщо послідовно – **НІ-І**. Дійсно, у першому випадку (рис. 2.30 *а*)), щоб $U_{Вих} = U^0$, необхідно подати логічну одиницю хоча б на один вхід, у другому (рис. 2.30 *б*)) – на всі входи.

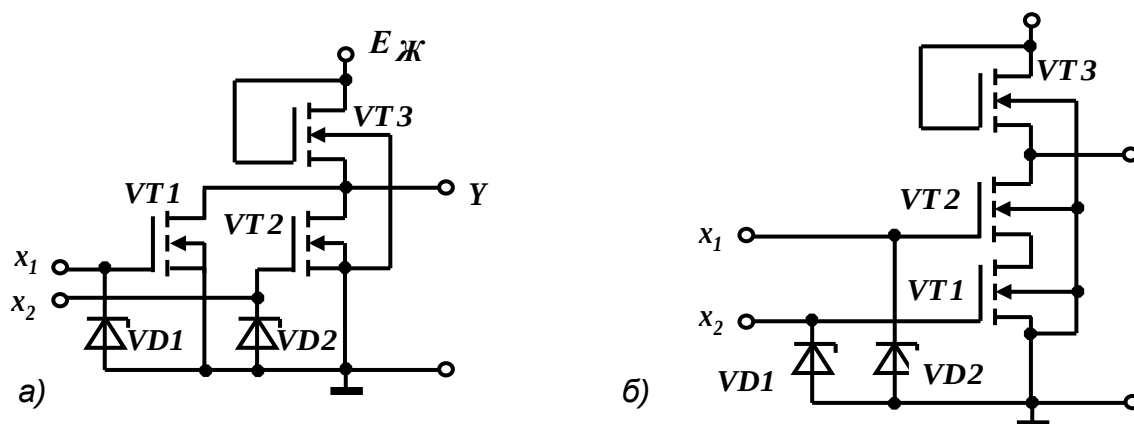


Рис. 2.30. Логічні елементи НІ-ЧИ (*а*) та НІ-І (*б*) на *n*-МОН-транзисторах

Кількість керованих транзисторів обмежена: при паралельному з'єднанні через зростання струму витоку закритих транзисторів падає рівень U^1 , при послідовному з'єднанні у відкритому стані зростає рівень U^0 .

У зв'язку з тим, що затвори транзисторів і підкладка розділені тонким шаром діелектрика, який має величезний опір витоку (приблизно 10^{12} Ом) і не дуже високу електричну міцність (150 ... 200 В), виникає небезпека необоротного пробоя цього шару статичним зарядом, який може бути переданий на затвор при поводженні з ІС. З метою захисту транзисторів від пробоя кожен вхід ЛЕ обладнують охоронним колом. На рис. 2.30 такі кола утворюють стабілітрони **VD1**, **VD2**, які обмежують вхідні напруги на безпечних рівнях.

Як впливає з п. 2.2.2.1, ключі на однотипних **МОН**-транзисторах мають великий вихідний опір. Тому при введенні охоронних кіл зростає паразитна ємність для елементів, які навантажуються, і різко падає швидкодія цифрового пристрою. Ця обставина багато в чому визначила застосування **МОНТЛ** елементів, головним чином, у великих ІС (**ВІС**), де, будучи в “глибинці”, вони не мають потреби в охоронних колах і ємності навантаження для них невеликі.

2.6.2. Елементи на комплементарних **МОН**-транзисторах

Базовими елементами серій ІС **КМОНТЛ** є інвертор і двонаправлений ключ. Реальна схема інвертора відрізняється від схеми ключа на комплементарних

МОП-транзисторах (рис. 2.10 а)), розглянутої в п. 2.2.2.2, лише наявністю захисту входів, які є зовнішніми виводами ІС (рис. 2.31).

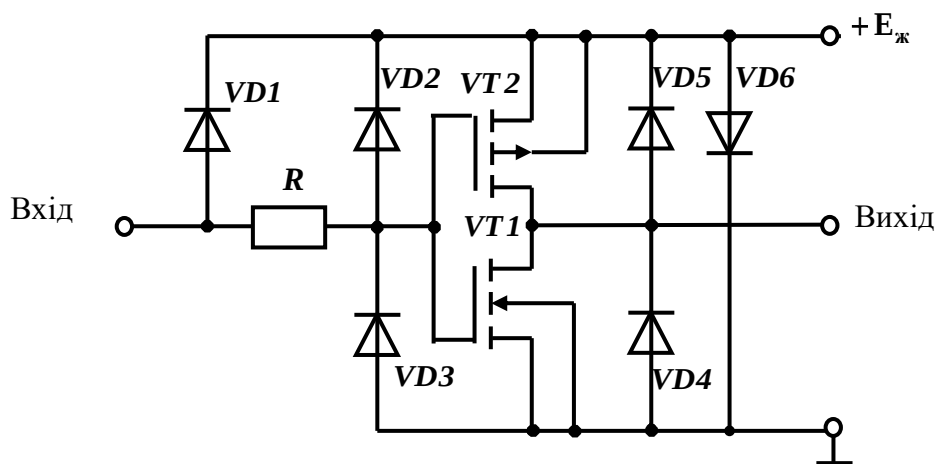


Рис. 2.31. Принципова схема інвертора на комплементарних МОП- транзисторах

Охоронна схема містить резистор R опором $0,5...1,5\text{ кОм}$ і діоди $VD1...VD3$, які шунтують вхідну напругу або на шину джерела живлення $+E_{\text{ж}}$, або на спільну шину. В нормальних умовах, коли вхідна напруга не виходить за межі $0,7...(E_{\text{ж}} + 0,7)\text{ В}$, діоди закриті. Якщо напруга, яка подається на вхід, виходить за вказані межі й внутрішній опір джерела сигналу малий, слід рахуватись з небезпечністю ушкодження охоронного кола надмірним вхідним струмом. Обмеження струму забезпечує резистор R . Верхній край його опору визначається допустимим погіршенням швидкодії ключа через зростання часу перезарядки вхідної ємності транзисторів і діодів $VD2, VD3$.

На відміну від діодів, які спеціально вводяться в схему для захисту від пробою, існують діоди $VD4...VD6$, які є. Супутні діоди $VD4, VD5$ підключені до виходу інвертора. Вони залишаються закритими, якщо напруга на виході, так як і на вході, знаходиться в межах напруги живлення. Діод $VD6$ відкривається при переполюсовці джерела живлення. Охоронні кола і супутні діоди на принципових схемах звичайно не зображають.

Для запобігання небажаних явищ напругу живлення варто завжди подавати раніше будь-яких вхідних сигналів, особливо якщо їхні джерела мають невеликий вихідний опір. Вимикання апаратури варто виконувати у зворотному порядку. Логічні елементи інших типів будуються на основі базових інверторів шляхом зміни топології з'єднань їх транзисторів на кристалі ІС. Так, для одержання елемента **НІ-ЧИ** n -канальні транзистори з'єднуються паралельно, а p -

каналні – послідовно (рис. 2.32 а)), для елемента **НІ-І** навпаки – *n*-каналні послідовно, а *p*-каналні паралельно (рис. 2.32 б)).

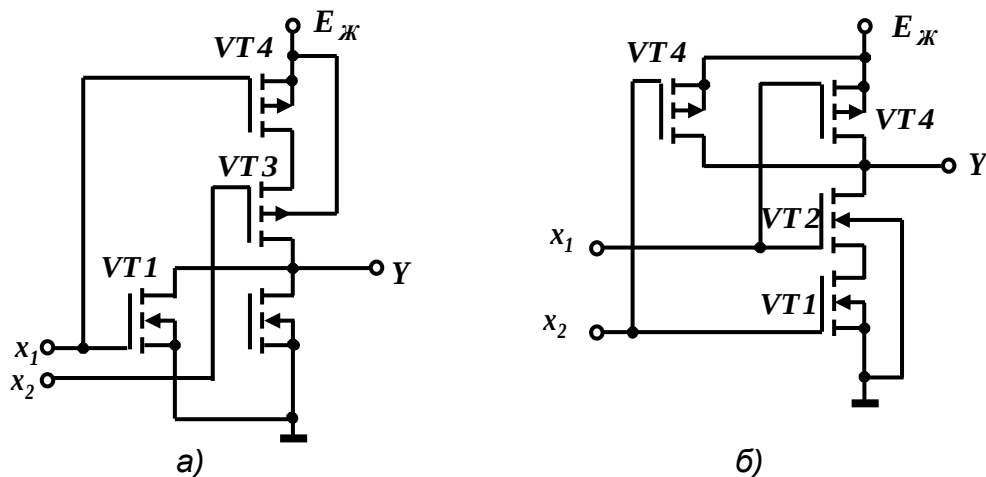


Рис.2.32.Логічні елементи НІ-ЧИ (а) й НІ-І (б) на комплементарних МОН-транзисторах

Утворюється два яруси транзисторів щодо вихідного виводу. Як і в окремому інверторі, тут при будь-якій комбінації входних сигналів у сталому режимі струм від джерела живлення практично відсутній, тому що один з ярусів завжди закритий. Наприклад, для **ЛЕ НІ-ЧИ** нижній ярус закритий за наявності на всіх входах сигналів логічного нуля, а при інших комбінаціях закритий верхній ярус. У логічному елементі **НІ-І** стани транзисторів протилежні.

Через малість залишкової напруги на відкритих транзисторах кількість входів **КМОНТЛ** елементів на рівні U^0 і U^1 майже не впливає. Однак елементи з кількістю входів більш чотирьох зазвичай не створюють. За необхідності розширити кількість логічних входів використовують монтажну логіку. На рис. 2.33, а) показаний приклад отримання багатовхідного елемента **НІ-І** шляхом підключення до виходів трьохвхідних **ЛЕ НІ-І** діодної схеми **ЧИ**. Схема реалізує функцію

$$Y = \overline{x_3 x_2 x_1} \vee \overline{x_6 x_5 x_4} = \overline{x_6 x_5 \dots x_1}.$$

При використанні елементів **НІ-ЧИ** та діодної схеми **І** (рис. 2.33 б)) розширюється кількість змінних для операції **НІ-ЧИ**:

$$Y = \overline{x_3 \vee x_2 \vee x_1} \cdot \overline{x_6 \vee x_5 \vee x_4} = \overline{x_6 \vee x_5 \vee \dots \vee x_1}.$$

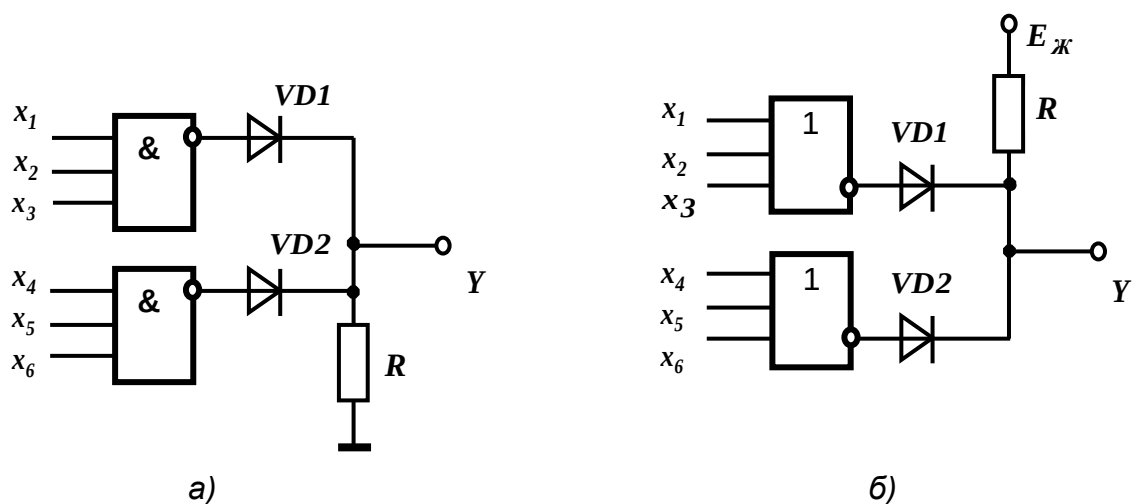


Рис. 2.33. Схеми для збільшення кількості змінних, над якими виконуються операції І-НІ (а) й ЧИ-НІ (б)

Двонаправлений ключ (ДК) у серіях **ІС КМОНТЛ** є другим базовим елементом. Він становить специфічний вузол, який не має функціональних аналогів серед мікросхем інших видів логіки. Призначення двонаправленого ключа можна уподібнити призначенню реле. Виготовляються **ДК** як у вигляді самостійних **ІС**, так і у складі **ІС**, які мають інші функціональні призначення.

Двонаправлений ключ без пристрою керування (рис. 2.34) складається з пари комплементарних **МОН**-транзисторів. Витоки і стоки в них з'єднані перехресно та виведені назовні. Два взаємно інверсні сигнали E та $\bar{E}$, які подаються на затвори транзисторів, керують ключем.

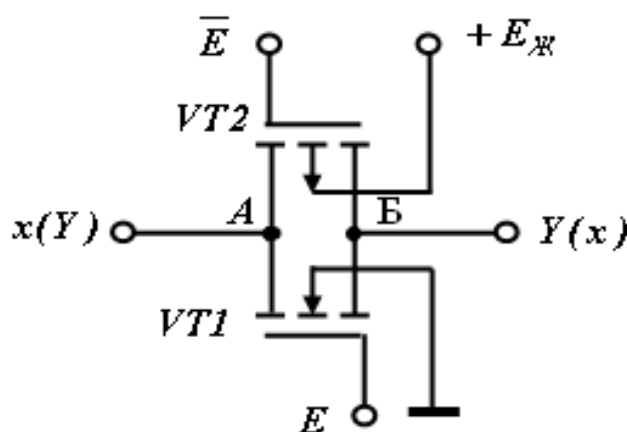


Рис. 2.34. Двонаправлений ключовий елемент

Обидва транзистори відкриті, коли на затворі n -канального транзистора $VT1$ напруга $U^1 \approx E_{Ж}$, а на затворі p -канального транзистора $VT2$ напруга

$U^0 \approx 0$. Провідні канали обох транзисторів мають невеликий опір (сотні Ом) і мають двосторонню провідність між точками А і Б.

Паралельне включення транзисторів зменшує спільний опір і робить його майже незалежним від напруги, яка комутується. Це дозволяє використовувати ДК і для комутації аналогових сигналів.

Коли керуючі сигнали на затворах змінюють своє значення, транзистори запираються й опір між точками А і Б зростає (до 10^9 Ом).

Керуючий сигнал E зазвичай формується за допомогою інвертора. На рис. 2.35 а) це транзистори $VT5$ і $VT4$. Умовне графічне позначення ДК приймемо таким, як показано на рис. 2.35 б).

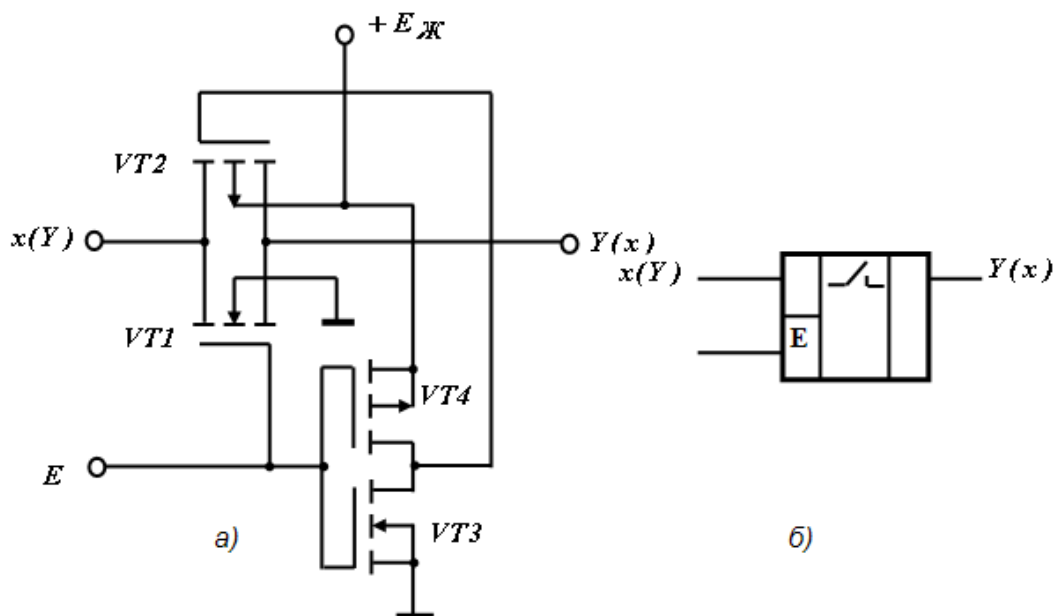


Рис. 2.35. Схема (а) й умовне графічне позначення (б) двонаправленого ключа

У силу своєї специфічності двонаправлені ключі обумовили і звичайні схемотехнічні рішення цифрових пристроїв. Наприклад, якщо у вихідне коло джерела сигналу включити ДК, вийде елемент із трьома статичними станами.

При закритих транзисторах ДК елемент знаходиться в стані високого імпедансу. У ряді випадків застосування ДК дозволяє істотно спростити цифрові пристрої, зокрема тригери, лічильники, регістри [1].

На закінчення необхідно відзначити, що мікросхеми КМОНТЛ, як ніякі інші, мають статичні параметри, близькі до ідеальних:

майже не споживають потужності;

мають дуже великий вхідний і малий вихідний опір, отже, високу навантажувальну здатність;

відрізняються високою завадостійкістю, обумовленою тим, що логічний перепад близький до $E_{ж}$, а пороговий рівень близький до $0,5 E_{ж}$.

Пороговий і логічний рівні практично не залежать від температури.

До позитивних якостей необхідно віднести і здатність нормально працювати в широкому діапазоні напруг живлення (3 ... 15 В), що дозволяє спростити сполучення з цифровими ІС інших типів (наприклад ТТЛ) і з багатьма аналоговими пристроями.

Помітимо, що зі збільшенням напруги живлення швидкодія теж збільшується. Обумовлено це пропорційним збільшенням рівня U^1 , при якому транзистори відкриваються більше й опір їх провідних каналів зменшується. Основний недолік мікросхем КМОНТЛ – порівняно невисока швидкодія ($t_{3m_{сеп}}$ складає десятки наносекунд). У багатьох випадках, коли вона не відіграє вирішальної ролі, цим мікросхемам варто віддавати перевагу. Порівняно невисока швидкодія не є принциповою властивістю КМОН-структур. Це часове явище, що переборюється у міру удосконалювання технології виготовлення.

2.7. Базовий логічний елемент інжекційної логіки

Як розвиток елементів транзисторної логіки з безпосередніми зв'язками в останні роки з'явилися елементи інтегральної інжекційної логіки (І²Л). Зовні їх схемна специфіка полягає у тому, що виключені навантажувальні резистори, а транзистори мають так зване інжекційне живлення, тобто електрична енергія, необхідна для роботи, вводиться шляхом інжекції неосновних носіїв струму за допомогою спеціального інжектора.

Структура транзистора з інжекційним живленням показана на рис. 2.36. На відміну від звичайного транзистора тут є додатковий **p₁-n₁**-перехід (інжектор) і його електрод (**I**). Крім того, області **n₁** та **n₂** виконують зворотні функції: **n₁** – емітерну, **n₂** – колекторну. Таку структуру можна зобразити у вигляді двох транзисторів: **p₁-n₁-p₂** і **n₁-p₂-n₂** (рис. 2.37).

Ключовий елемент **Кл**, з'єднаний паралельно входу *n-p-n*-транзистора, служить для керування. Його роль зазвичай виконує аналогічний транзистор іншого елемента.

Якщо в коло інжектора включене джерело $E_{Ж}$, то в ньому тече струм

$$I_i = \frac{E_{Ж} - U'_{EB}}{R}.$$

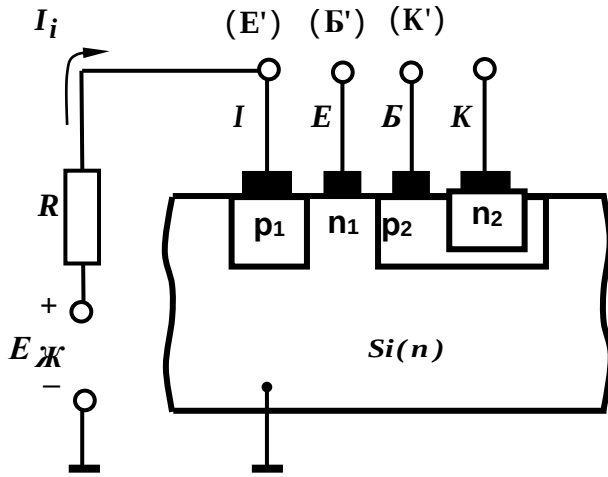


Рис.2.36. Структура транзистора з інжекційним живленням

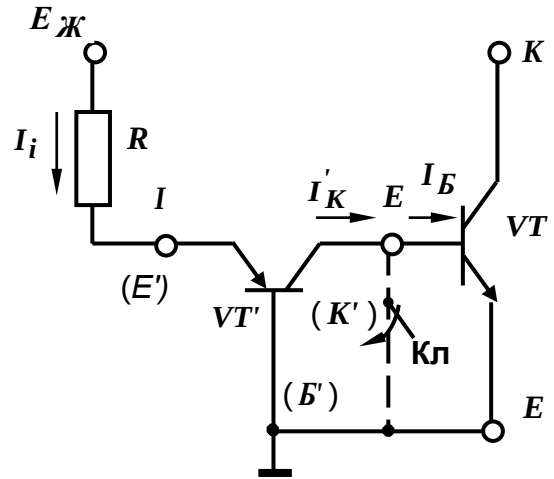


Рис.2.37. Еквівалентна електрична модель транзистора з інжекційним живленням

При $E_{Ж} \gg U'_{EB}$ струм $I_i = \frac{E_{Ж}}{R} = \text{const}$. У колекторному колі транзистора VT' протікає струм $I'_K = \alpha' I_i = \text{const}$. Це еквівалентно наявності в колі бази транзистора генератора струму $I_{\Gamma} = I'_K$, що дає підставу спростити схему і привести до вигляду, зображеному на рис. 2.38.

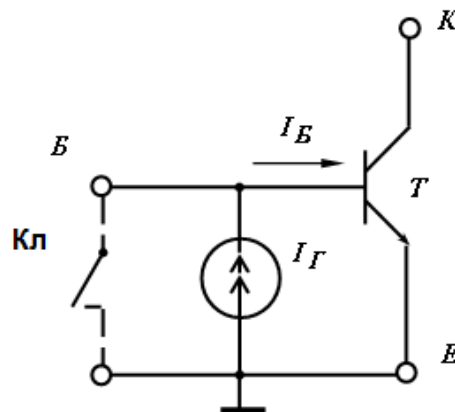


Рис. 2.38. Спрощена електрична модель транзистора з інжекційним живленням

Режим роботи транзистора T залежить від стану ключа $КЛ$. Якщо ключ замкнути, то струм I_{Γ} протікає через нього, напруга $U_{BE} \approx 0$, струм бази

$I_B = 0$, транзистор закритий. Якщо ключ розімкнати, то $I_B = I_G$, транзистор відкритий і, при типовому навантаженні насичений. Покажемо це, зобразивши кілька розглянутих структур у вигляді послідовного кола (рис. 6.39). Нехай зовнішнє коло бази транзистора $VT1$ розірване. Тоді $I_{B1} = I_{G1}$, і транзистор відкритий. Для його насичення повинна виконуватися умова: $I_B \geq I_{B_{кр}}$, тобто $I_{G1} \geq \frac{I_{G2}}{\beta_1}$, де β_1 – статичний коефіцієнт передачі струму $VT1$, а I_{G2} – струм генератора в колі бази транзистора $VT2$. Оскільки $I_{G2} \approx I_{G1}$, умова насичення виконується при $\beta_1 \geq 1$. Забезпечити таке значення коефіцієнта β не складно.

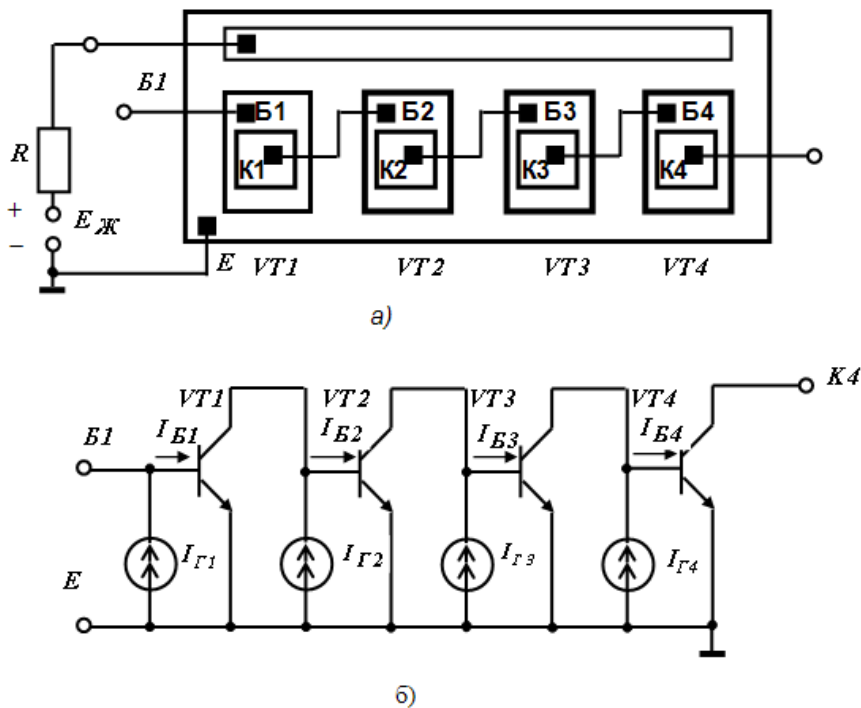


Рис. 2.39. Планарна структура (а) та електрична схема (б) послідовно з'єднаних інверторів I^2L

При насиченому транзисторі $VT1$ транзистор $VT2$, який навантажується на нього, закритий, тому що струм I_{G2} тече через $VT1$ та $I_{B2} = 0$. Стан наступних транзисторів, таким чином, чергується: $VT3$ відкритий, $VT4$ закритий і т.д. Розглянуте коло становить послідовне з'єднання інверторів. При цьому найбільша напруга U_{BE_H} , яка відповідає рівню логічної одиниці, встановлюється на входах відкритих транзисторів $VT1$ і $VT3$, а мінімальна U_{KE_H} ,

яка відповідає рівню логічного нуля U^0 , – на входах закритих транзисторів $VT2$ і $VT4$.

Базовим елементом I^2L можна вважати сам інвертор у трохи зміненому вигляді – у нього транзистор VT багатоколекторний, що дозволяє отримувати інші логічні елементи (**ЧИ**, **I**, **НІ-І** тощо) монтажним способом.

На рис. 2.40 зображена схема **ЛЕ** на транзисторах, з трьома колекторами. Тут інжектором є багатоколекторний транзистор VT' . За допомогою монтажно́ї логіки організована операція **НІ-ЧИ** по двох виходах (Y_4 і Y_5) та інверсія входних сигналів по трьох виходах (Y_1 , Y_2 , Y_3). Доведемо це.

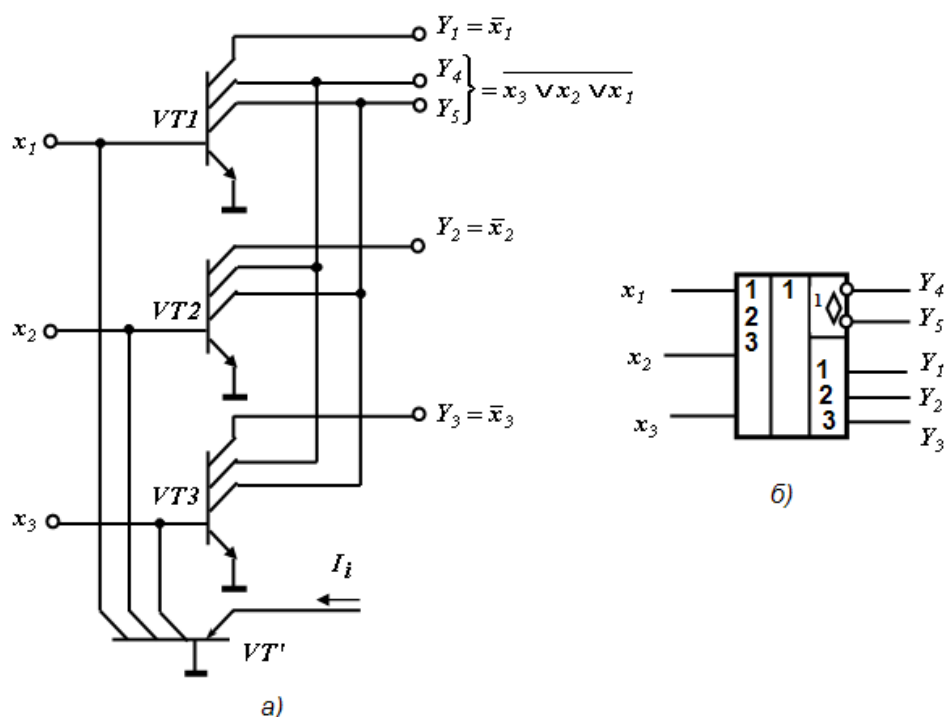


Рис. 2.40. Схема (а) та умовне графічне позначення (б) багатофункціонального логічного елемента I^2L

Якщо джерела сигналів x_i мають низький вихідний опір, то струми I_{Fi} , які задаються транзистором VT' , відводяться від баз $VT1...VT3$ у входні кола, і ці транзистори закриті. Напруга на входах $U^0 \approx 0$. Напруга на всіх виходах **ЛЕ** дорівнює напрузі U_{BE_n} навантажувальних транзисторів (на схемі навантаження не показано). Якщо хоча б одне з джерел сигналу x_i переходить у високоомний стан ($x_i = 1$), то струм I_F відповідного транзистора перемикається в базу, і він насичується. При цьому на його інверсному виході та на виходах Y_4 і Y_5 встановлюється рівень логічного нуля.

Зазначимо, що ЛЕ має два однакових, але електрично-незалежних один від одного виходів: Y_4 і Y_5 . Такий схемотехнічний прийом притаманний монтажній логіці. Він дозволяє розгалужувати вихідний сигнал за навантаженнями.

На рис. 2.41 показано одержання операції МОНТАЖНЕ І шляхом простого з'єднання колекторів транзисторів з інжекційним живленням у спільній точці. Вивід від цієї точки є виходом монтажного ЛЕ. Дійсно, якщо в колекторних (вхідних) колах струм не тече, то на всіх входах x_i та на виході за рахунок струму навантаження I_F встановлюється висока напруга U_{BEH} . Якщо хоча б одне колекторне коло стає низькоомним, на виході напруга виходить близькою до нуля.

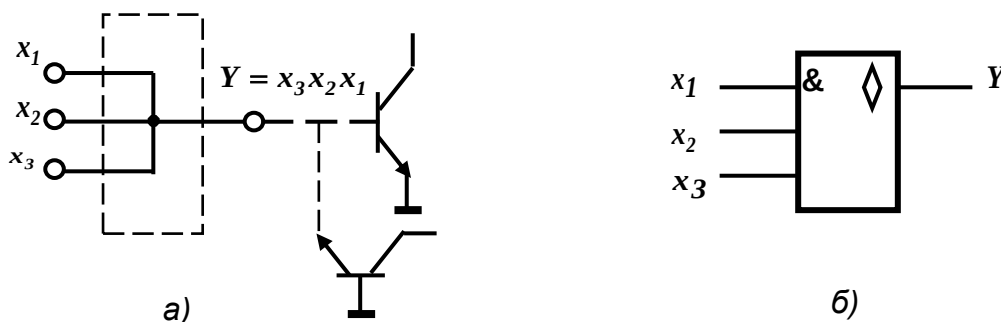


Рис. 2.41. Схема виконання операції МОНТАЖНЕ І на елементах I^2L (а) та її умовне графічне позначення (б)

Елементи I^2L характеризуються високою економічністю та швидкодією. При струмі інжектора **10...100 мкА**, що відповідає споживаній потужності до ста мікроват, час $t_{3m_{сеп}}$ може складати одиниці наносекунд. Висока швидкодія пояснюється невеликим (**0,5...0,7 В**) перепадом логічних рівнів і малою паразитною ємністю структури. Застосовуючи діоди Шотткі, можна знизити $t_{3m_{сеп}}$ до **0,1 нс**.

Реалізувати переваги I^2L в простих ІС неможливо через низьку завадостійкість. Однак структури I^2L дуже перспективні для ВІС та надвеликих інтегральних мікросхем (НВІС). Завадостійкість у них підвищується за рахунок використання в якості периферійних каскадів, які мають зв'язок із зовнішніми виводами, елементів з великим перепадом логічних рівнів.

2.8. Порівняльна оцінка базових логічних елементів

Важливим етапом проектування цифрової апаратури є вибір серій ІС, які найбільш прийнятно задовольняють вимогам за швидкодією, енергоспоживан-

ням, завадостійкістю, навантажувальною здатністю, стійкістю до спеціальних впливів, надійністю тощо. Враховується і функціональний склад серій.

Оскільки майже всі перелічені показники будь-якої ІС у складі конкретної серії визначаються параметрами базового ЛЕ, порівняльну оцінку серій цифрових ІС зазвичай зводять до порівняльної оцінки базових ЛЕ.

У табл. 2.1 наведені порівняльні дані ІС за деякими конструктивно-технологічними та електричними параметрами ЛЕ, які використовуються в ІС високого ступеня інтеграції як базові.

Таблиця 2.1

Типові параметри базових ЛЕ

Параметр	n-МОН	КМОН	ТТЛ, ТТЛШ	ЕЗЛ	І ² Л
Площа, $\times 10^{-3}$, мм ²	4 – 7	6 – 19	12 – 38	12 – 31	2 – 4
Кількість етапів дифузії і легування	3	4	4	4 – 5	3 – 4
Кількість етапів маскування	6	7	7	8 – 9	5 – 7
Середня затримка поширення t_{3mCep} , нс	40 – 100	15 – 50	3 – 10	0,5 – 2	> 3
Статична споживана потужність P_{CnCep} , мВт	0,2 – 0,5	< 0,001	1 – 3	5 – 15	< 0,2
Напруга живлення $E_{ж}$, В	3 – 27	3 – 27	5	– 5	1 – 2
Напруга логічного нуля U^0 , В	1 – 2	1 – 2	0,4	– 1,5	0,05
Напруга логічної одиниці U^1 , В	3 – 15	3 – 15	2,4 – 4,5	– 0,95	0,6 – 0,8

Зіставлення табличних даних за сукупністю параметрів виявляє очевидні переваги інжекційної логіки. Елементи І²Л у багато разів менші елементів інших структур, досить технологічні й економічні, мають найнижчу роботу перемикавання. Це обумовлює перспективність їхнього застосування в ВІС і НВІС.

З огляду на те, що одним з основних обмежуючих на ступінь інтеграції факторів виступає розсіювання споживаної потужності, дуже перспективні для НВІС також елементи КМОНТЛ. Структури на однотипних МОН-

транзисторах відрізняються високою щільністю упакування. Тому їм віддається перевага в напівпровідникових динамічних запам'ятовуючих пристроях великої ємності.

Для побудови швидкодіючих **ВІС** використовуються елементи **ЕЗЛ**. Однак, велика споживана потужність і низька щільність упакування не дозволяють досягти високого ступеня інтеграції. Це ж, хоча й меншою мірою, стосується використання елементів **ТТЛ**. У цьому зв'язку **ВІС** на біполярних транзисторах часто виготовляють секційними (на 2, 4 розряди), що дозволяє з'єднувати їх паралельно й тим самим нарощувати структуру цифрового пристрою до необхідної розрядності.

У споживчому плані про базовий **ЛЕ** і відповідні **ІС** будь-якої складності можна відзначити таке.

Елементи **ЕЗЛ** найбільш швидкодіючі, здатні перемикатися з частотою 100 МГц і більше. Однак вони вимагають спеціального захисту від завад по сигнальних колах і шинах живлення, застосування узгоджених ліній зв'язку, організацію ефективного тепловідводу від **ІС**, друкованих плат, блоків тощо. Рекомендуються для використання тоді, коли **ІС** інших типів логіки не забезпечують необхідної швидкодії [6].

Елементи **ТТЛ** і **ТТЛШ** відрізняються від усіх інших тим, що перекривають широкий діапазон значень $t_{3m_{Cep}}$ і $P_{Cn_{Cep}}$, тобто складають ту “золоту середину”, яка найчастіше потрібна на практиці. Вони мають прийнятну завадостійкість і навантажувальну здатність. На їхній основі створено більше десяти функціональних повних взаємоузгоджених серій, які утворюють своєрідний параметричний ряд за $t_{3m_{Cep}}$ і $P_{Cn_{Cep}}$. Мікросхеми **ТТЛ** і **ТТЛШ** одержали найбільше поширення в цифрових пристроях широкого застосування.

З усіх **ЛЕ** на однотипних **МОН**-транзисторах перевага віддається **n**-канальним. Вони мають добрі показники за всіма параметрами крім затримки перемикання. Істотно низька швидкодія, особливо при наявності ємності навантаження, зробила недоцільним створення на їхній основі **ІС** малого і середнього ступеня інтеграції.

Подібне пояснення монопольного використання в **ІС** тільки високого ступеня інтеграції має також інжекційна логіка – елементи **І²Л** відрізняються вкрай низькою завадостійкістю.

Елементи **КМОНТЛ** на декілька порядків більш економічні від всіх інших логік, мають високу завадостійкість і прогресуючу швидкодію, здатні пра-

цювати в широкому діапазоні напруг живлення. Вони вважаються одними з найперспективніших для створення ІС різного ступеня інтеграції.

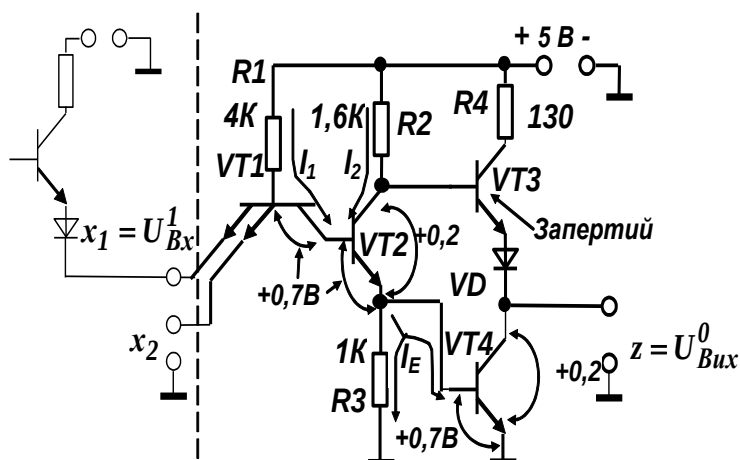
Таким чином, у даний час для ІС малої і середньої інтеграції перспективні базові **ЛЕ ТТЛ, ТТЛШ, ЕЗЛ і КМОНТЛ**, для **ВІС і НВІС – І²Л, КМОНТЛ**, а, у ряді випадків, **n-МОНТЛ**. У швидкодіючих ІС підвищеної інтеграції використовуються модифіковані елементи **ЕЗЛ і ТТЛШ**.

Успіхи в розвитку технології, освоєнні нових напівпровідникових та інших матеріалів, безумовно, будуть сприяти поліпшенню якісних показників ІС. Можливі також перерозподіли ролей “лідерів” між базовими **ЛЕ**, але в цілому схемотехнічні переваги та недоліки їх визначились.

КОНТРОЛЬНІ ПИТАННЯ

1. Дати характеристику базового логічного елемента.
2. Пояснити принцип функціонування транзисторного ключа із спільним емітером. Які режими роботи транзистора використовуються при цьому?
3. Які фізичні процеси впливають на швидкодію транзисторного ключа?
4. Пояснити принцип побудови та функціонування транзисторного ключа на уніполярному (**МОН**) транзисторі.
5. Навести класифікацію базових логічних елементів.
6. Які основні параметри характеризують властивості базових логічних елементів?
7. Пояснити призначення кожного елемента базового логічного елемента транзисторно-транзисторної логіки.
8. Пояснити в чому полягає необхідність використання елементів **ТТЛ** з відкритим колектором? Як працюють такі елементи?
9. Пояснити принцип роботи елементів **ТТЛ** з трьома стабільними станами.
10. Яким чином зміниться (або не зміниться) робота базового елемента **ТТЛ** (рис.2.16), якщо діод **VD** підключити не до емітера, а до бази транзистора **VT4**?
11. У чому полягає суть фізичних процесів, що відбуваються під час функціонування базового логічного елемента емітернозв'язаної логіки?
12. Надати порівняльну характеристику базовим логічним елементам комп'ютерної схемотехніки.

13. Визначити потужність, що споживається для наведеної схеми базового елемента ТТЛ за умови впливу на його вхід сигналу логічної одиниці $x_1 = U_{Bx}^1$.

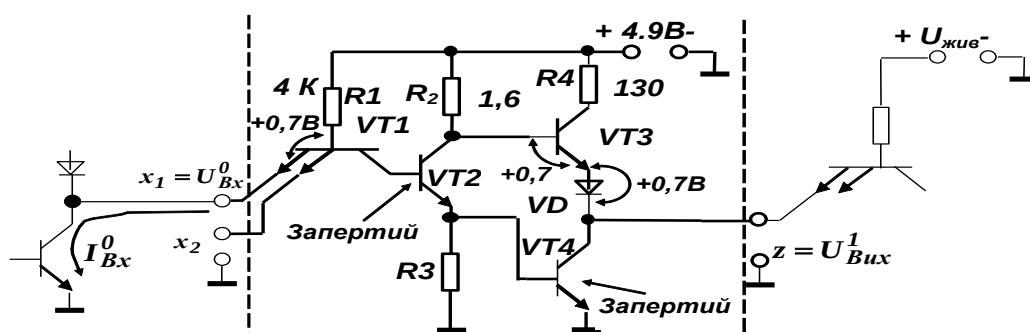


14. Пояснити роботу схеми інвертора на комплементарних **МОН**-транзисторах.

15. Навести схему двонаправленого ключа на комплементарних **МОН**-транзисторах. Дати характеристику режимам роботи ключа.

16. Пояснити принцип роботи базового логічного елемента інжекційної логіки.

17. Для наведеної схеми базового ТТЛ елемента визначити величину вхідного струму I_{Bx}^0 та вихідної напруги за умови впливу на його вхід сигналу логічного нуля $x_1 = U_{Bx}^0 = 0,2 \text{ B}$.



3. ЛОГІЧНІ ОСНОВИ ПОБУДОВИ ЕЛЕМЕНТІВ

3.1. Логічні основи побудови елементів

Цифровий автомат (ЦА) – це математична модель реального (технічного) пристрою, який служить для перетворення цифрової інформації.

У загальному випадку ЦА має n входів $x_1, x_2, \dots, x_n$ та m виходів $z_1, z_2, \dots, z_m$, кожен з яких може приймати деяке кінцеве число дискретних значень (рис. 3.1.)

Входи $x_1, x_2, \dots, x_n$ утворюють множину входів ЦА $X = \{x_1, x_2, \dots, x_n\}$. Виходи $z_1, z_2, \dots, z_m$ утворюють множину виходів ЦА $Z = \{z_1, z_2, \dots, z_m\}$.

Залежність між вихідними та вхідними сигналами в цифровому автоматі реалізується за допомогою з'єднання елементів, які прийнято називати логічними.

Цифровий автомат без пам'яті (комбінаційний автомат) – це такий ав-

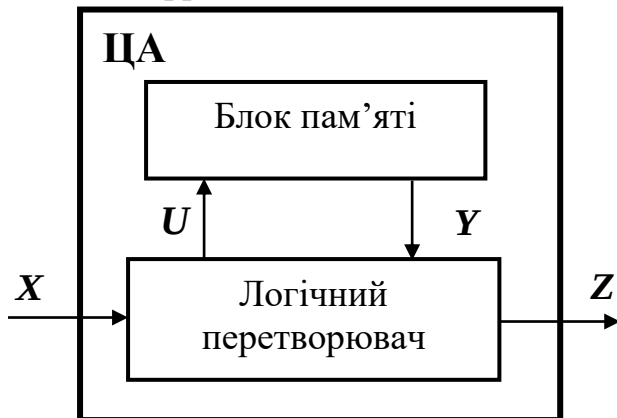


Рис.3.1. Структура цифрового автомата

томат, у якого кожному вхідному набору вхідних сигналів відповідає визначений вихідний набір незалежно від порядку надходження вхідних наборів.

В ЦА можуть бути елементи, які здатні міняти свій стан під впливом вхідних сигналів і зберігати цей стан до моменту надходження наступних сигналів. Такі елементи прийнято називати елементами пам'яті.

Цифровий автомат з пам'яттю

– такий автомат, вихідний набір якого залежить не тільки від вхідного набору, що діє в даний момент, але і від усіх попередніх вхідних наборів, а також від порядку їх надходження.

У загальному випадку блок пам'яті може мати K елементів пам'яті $y_1, y_2, \dots, y_k$, кожен з яких може приймати деяку кінцеву кількість станів.

Елементи $y_1, y_2, \dots, y_k$ утворюють множину елементів пам'яті автомата $Y = \{y_1, y_2, \dots, y_k\}$.

Логічний перетворювач автомата крім формування вихідних сигналів забезпечує управління блоком пам'яті цифрового автомата. Для управління пам'яттю може бути l виходів управління пам'яттю $u_1, u_2, \dots, u_l$, які утворюють множину сигналів управління блоком пам'яті цифрового автомату $U = \{u_1, u_2, \dots, u_l\}$.

Якщо кожному вхідному сигналу автомата присвоїти конкретне значення на множині $\{0,1\}$, то отримана сукупність значень вхідних сигналів буде становити так званий вхідний набір цифрового автомата. В загальному випадку вхідний набір автомата запишеться наступним виразом:

$$w = \tilde{x}_n \dots \tilde{x}_2 \tilde{x}_1, \text{ де } \tilde{x}_i = \begin{cases} 0, & (x_i = 0) \\ 1, & (x_i = 1) \end{cases}.$$

Для однозначного задавання набору домовилися змінну $\tilde{x}_i$, яка має одиничне значення, записувати як x , а якщо ця змінна приймає нульове значення, то $\bar{x}$.

Наприклад, при $n = 4$ вхідний набір під номером 13 буде відповідати двійковому представленню цього числа $W_{13} = \overset{1}{x_4} \overset{1}{x_3} \overset{0}{x_2} \overset{1}{x_1}$.

Сукупність наборів, на яких визначена ЛФ, утворюють область завдання ЛФ. Кількість вхідних наборів складає величину $N_{Bx} = 2^n$.

Логічними змінними (ЛЗ) називаються знаки (символи), які приймають різні значення із відповідної області. Якщо область значень (ЛЗ) складають два значення, то такі (ЛЗ) називають двійковими або булевими. Логічною функцією називають функцію, що залежить від логічних змінних та приймає значення з тієї ж множини, що і змінні, від яких вона залежить.

Набори змінних, на яких логічна функція приймає значення 1, будемо називати *одиничними наборами*, а набори, на яких функція приймає значення 0, будемо називати *нульовими наборами*.

На кожному наборі логічних змінних логічна функція може приймати два значення, тому співвідношення, що пов'язує кількість логічних функцій залежно від кількості змінних виглядає наступним чином: $N_{Bx} = 2^{2^n}$.

Будь-яку логічну функцію, що залежить від n змінних ($n > 2$) можна виразити через функції, що залежать від однієї або двох змінних. Тому логічні функції, що залежать від нуля, однієї або двох змінних, займають особливе місце в теорії логічних функцій. Ці функції називаються елементарними логічними функціями.

Розглянемо ці функції.

При $n = 0$ існують дві логічні функції $f_0 \equiv 0$ та $f_1 \equiv 1$. Функція $f_0 \equiv 0$ називається константою нуля, а функція $f_1 \equiv 1$ називається константою одиниці.

При $n = 1$ існують чотири логічні функції (табл.3.1)

Таблиця 3.1

Логічні функції однієї логічної змінної

f_i	x		Аналітичний запис функції	Назва функції
	0	1		
f_0	0	0	$f_0(x) \equiv 0$	константа 0
f_1	1	0	$f_1(x) = x$	повтор "x"
f_2	0	1	$f_2(x) = \bar{x}$	інверсія " $\bar{x}$ " (НІ)
f_3	1	1	$f_3(x) \equiv 1$	константа 1

При $n = 2$ існує $2^{2^2} = 16$ логічних функцій (табл. 3.2).

Таблиця 3.2

Набір логічних функцій двох змінних

Аргументи					Функція	Найменування функції
x_1 x_2	0 0	0 1	1 0	1 1		
	0	0	0	0	$Y_1 = 0$	Константа 0
	0	0	0	1	$Y_2 = x_2 x_1$	Кон'юнкція; операція І
	0	0	1	0	$Y_3 = \bar{x}_2 x_1$	Заборона по x_2
	0	0	1	1	$Y_4 = x_1$	Тотожність x_1
	0	1	0	0	$Y_5 = x_2 \bar{x}_1$	Заборона по x_1
	0	1	0	1	$Y_6 = x_2$	Тотожність x_2
	0	1	1	0	$Y_7 = x_2 x_1 \vee \bar{x}_2 \bar{x}_1 =$	виключальне ЧІ
	0	1	1	1	$x_2 \oplus x_1$	"сума за модулем 2"
	1	0	0	0	$Y_8 = \bar{x}_2 \vee \bar{x}_1$	Диз'юнкція; операція ЧІ
	1	0	0	1	$Y_9 = x_2 \vee x_1 = x_2 \downarrow x_1$	Операція НІ – ЧІ (стрілка Пірса)
	1	0	1	0	$Y_{10} = x_2 x_1 \vee \bar{x}_2 \bar{x}_1 = x_2 \sim x_1$	РІВНОЗНАЧНІСТЬ (еквівалентність)
	1	0	1	1	$Y_{11} = \bar{x}_2$	Інверсія x_2 ; заперечення x_2
	1	1	0	0	$Y_{12} = \bar{x}_2 \vee x_1 = x_2 \rightarrow x_1$	Імплікація від x_2 до x_1
	1	1	0	1	$Y_{13} = \bar{x}_1$	Інверсія x_1 ; заперечення x_1
	1	1	1	0	$Y_{14} = x_1 \vee x_2 = x_1 \rightarrow x_2$	Імплікація від x_1 до x_2
	1	1	1	1	$Y_{15} = x_1 x_2 = x_1 / x_2$	Операція НІ-І (штрих Шеффера)
	1	1	1	1	$Y_{16} = 1$	Константа 1

Значення розглянутих функцій полягає в тому, що з них може бути побудована довільна логічна функція, що залежить більш ніж, від 2-х змінних.

Існує поняття функціонально повної системи логічних функцій (елементів). До такої системи відноситься такий набір функцій (елементів), використовуючи який можна реалізувати будь-яку (яку завгодно складну) логічну функцію.

3.2. Логічні елементи. Взаємна відповідність булевих функцій та логічних елементів

На практиці логічні операції реалізують за допомогою елементів, які носять назву логічних елементів. На вході логічного елемента подають логічні змінні $X_n, X_{n-1}, \dots, X_1$, кожна з яких має область визначення $\{0,1\}$, а на виході його формується результат логічної операції $f(X_n, X_{n-1}, \dots, X_1)$, який також має область визначення $\{0,1\}$.

Логічні елементи розрізняються між собою характером виконуваної функції, кількістю входів/виходів та інших ознак.

Входи та виходи логічних елементів залежно від сигналу, за яким сприймається (генерується) визначене значення двійкової змінної, поділяються на прямі й інверсні.

На прямому вході (виході) логічного елемента двійкова змінна має значення логічної 1, коли сигнал на цьому вході (виході) має значення, прийняте за 1. На інверсному вході (виході) логічного елемента двійкова змінна має значення 1, коли сигнал на цьому вході (виході) відповідає прийнятому за 0.

Входи, рівноцінні в логічному відношенні, допускають об'єднання; при цьому вони діють як один вхід. На функціональних і принципових схемах логічні елементи зображують у вигляді прямокутника (рис. 3.2), у верхній внутрішній частині якого вказують символ функції, наприклад "&" для І та "1" для ЧІ.

На схемі входи показують з лівої сторони прямокутника, виходи – з правої. Допускається інша орієнтація прямокутника, при якій входи розташовують зверху, а виходи – знизу. Інверсні входи і виходи виділяють невеликим кружком біля виводу.

Виводи живлення зазвичай не показують.

Виводи, які не несуть логічної інформації, позначають зіркою.

У сучасних цифрових пристроях логічні стани подаються двома рівнями напруги (потенціалів): **L** (англ. **low** – низький) і **H** (англ. **high** – високий). Ці два значення називають логічними рівнями. Існують два роди логічних угод

(потенційного кодування) залежно від того, яким рівнем напруги кодується логічна 1 та, відповідно, логічний 0. В угоді позитивної логіки високому рівню напруги ставлять у відповідність логічну 1, а низькому – логічний 0. У випадку негативної логіки – навпаки.

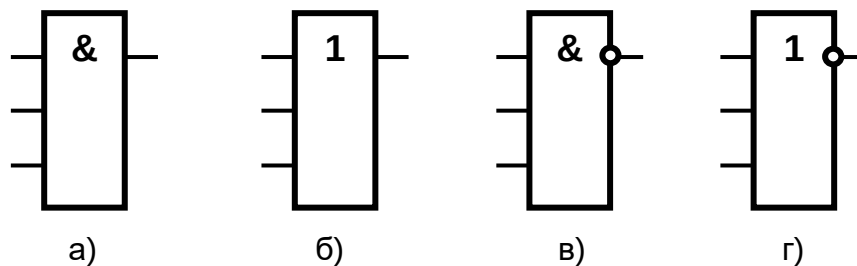


Рис. 3.2. Логічні елементи: а) І; б) АБО; в) НЕ-І; г) НЕ АБО

Елемент, який виконує логічні функції, можна оцінювати з позицій як позитивної, так і негативної логіки. Його функціональна роль в обох випадках буде різною. У табл. 3.3, як приклад, наведена таблиця істинності, виражена в рівнях сигналів, для якогось двовхідного елемента.

Таблиця 3.3
Рівні сигналів

x_2	x_1	$Y(x_2, x_1)$
L	L	L
L	H	L
H	L	L
H	H	H

Таблиця 3.4
Операція І

x_2	x_1	$Y(x_2, x_1)$
0	0	0
0	1	0
1	0	0
1	1	1

Таблиця 3.5
Операція АБО

x_2	x_1	$Y(x_2, x_1)$
1	1	1
1	0	1
0	1	1
0	0	0

За правилами позитивної логіки ($H = 1$) елемент виконує операцію І (табл. 3.4), а в негативній логіці ($H = 0$) він функціонує як елемент АБО (табл. 3.5). Властивість подвійності логічного елемента можна показати також, використовуючи закон де Моргана. Так, якщо $Y(x_2, x_1) = x_2 x_1$, то $\overline{Y(x_2, x_1)} = \overline{x_2 x_1} = \overline{x_2} \vee \overline{x_1}$. Зважаючи на сказане вище, елементи, які виконують логічні операції, допускається зображувати на схемах у двох логічно еквівалентних формах (рис.3.3) [1].

Якщо є зображення логічного елемента, його еквівалентну форму можна отримати, проробивши таке:

в основному полі зображення елемента символ операції “&” замінити на символ “1” або навпаки;

усі прямі входи (виходи) замінити інверсними, а інверсні – прямими.

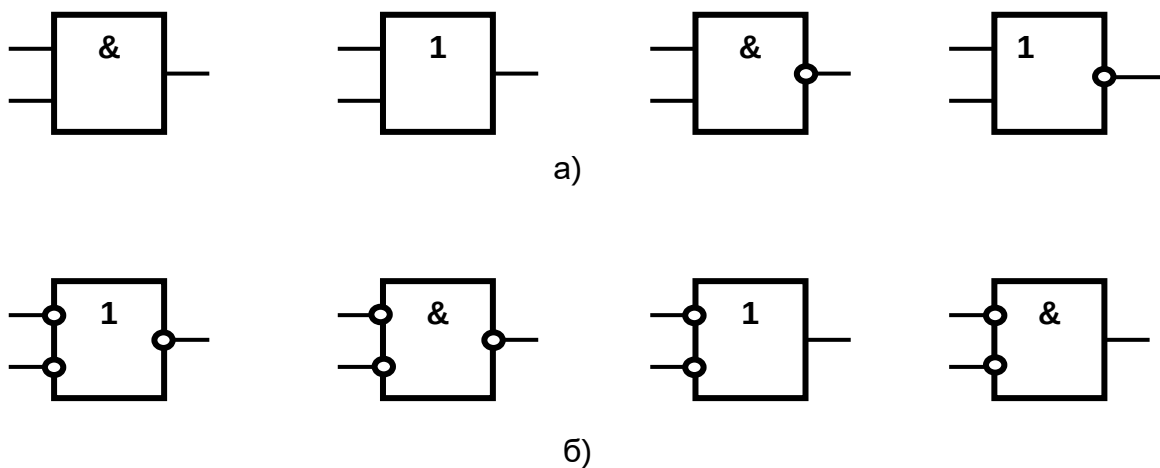


Рис. 3.3. Еквівалентні форми зображення логічних елементів:
a – в позитивній логіці; *б* – в негативній логіці

Угода позитивної логіки має переважне застосування. Тому, якщо не робиться спеціальне застереження щодо виду логіки, прийнято вважати її позитивною. В той же час, з погляду наочності і зручності читання схем для тих випадків, коли перемикаючим сигналом служить логічний 0, зручніше зображувати елементи в негативній логіці.

3.3. Основні аксіоми і закони булевої алгебри

Математичний апарат, котрий описує дії дискретних і цифрових пристроїв, базується на алгебрі логіки або булевій алгебрі, названій на ім'я її основоположника *Д. Буля*. Основні аксіоми та закони цієї алгебри представлені в табл. 3.6.

Булева алгебра оперує двійковими змінними, які умовно позначаються як 1 та 0.

Якщо кожному набору логічних змінних поставлено у відповідність значення логічної функції, то така функція носить назву повністю визначеної логічної функції, в іншому випадку – не повністю визначеної логічної функції. Набори, на яких логічна функція не задана, називаються невизначеними або умовними наборами. Таким чином, при табличному представленні логічних функцій її одиничні набори позначаються як “1”, нульові набори – як “0”, а невизначені – як “~”.

В основі булевої алгебри лежить поняття *перемикальної* або *логічної функції* вигляду $Y = f(x_n, \dots, x_2, x_1)$ щодо аргументів $x_n, \dots, x_2, x_1$, які, як і її аргументи, можуть приймати тільки два значення – 0 або 1.

Основні аксіоми і закони булевої алгебри

Аксіоми (тотожності)	$\bar{1} = 0; \quad \bar{0} = 1$
	$1 \vee x = 1; \quad 0 \cdot x = 0$
	$0 \vee x = x; \quad 1 \cdot x = x$
	$x \vee x = x; \quad x \cdot x = x$
	$x \vee \bar{x} = 1; \quad x \cdot \bar{x} = 0$
	$\overline{\overline{x}} = x$
Переставний закон (закон комутативності)	$x_2 \cdot x_1 = x_1 \cdot x_2;$ $x_2 \vee x_1 = x_1 \vee x_2$
Сполучний закон (закон асоціативності)	$x_3(x_2x_1) = (x_3x_2)x_1 = x_3x_2x_1;$ $x_3 \vee (x_2 \vee x_1) = (x_3 \vee x_2) \vee x_1 =$ $x_3 \vee x_2 \vee x_1$
Розподільний закон (закон дистрибутивності)	$x_3(x_2 \vee x_1) = x_3 \cdot x_2 \vee x_3x_1;$ $x_3 \vee (x_2x_1) = (x_3 \vee x_2)(x_3 \vee x_1)$
Закон інверсії (закон де Моргана)	$\overline{x_2 \cdot x_1} = \bar{x}_2 \vee \bar{x}_1;$ $\overline{x_2 \vee x_1} = \bar{x}_2 \cdot \bar{x}_1$
Закон поглинання (абсорбція)	$x_2 \vee x_2 \cdot x_1 = x_2;$ $x_2(x_2 \vee x_1) = x_2$
Закон склеювання	$x_2x_1 \vee \bar{x}_2 \cdot x_1 = x_1;$ $(x_2 \vee x_1)(\bar{x}_2 \vee x_1) = x_1$

Усі можливі логічні функції n змінних можна утворити за допомогою трьох основних операцій:

- 1) логічне заперечення (інверсія, операція NI , позначається штрихом над змінною);
- 2) логічне множення (кон'юнкція, операція I , позначається як “ $\wedge$ ” або “ $\cdot$ ”);
- 3) логічне додавання (диз'юнкція, операція CI , позначається як “ $\vee$ ” або “ $+$ ”).

Операція логічне заперечення над однією змінною характеризується такими властивостями: функція $Y = 1$ при аргументі $x = 0$ та $Y = 0$, якщо $x = 1$.

Позначається логічне заперечення рискою над змінною: $Y = \bar{x}$ (ігрек дорівнює не ікс). Відповідно $\bar{\bar{Y}} = x$.

Операція логічного множення для двох змінних характеризується так: $0 \cdot 0 = 0$; $0 \cdot 1 = 0$; $1 \cdot 0 = 0$; $1 \cdot 1 = 1$, тобто функція $Y = 1$, якщо всі аргументи дорівнюють 1.

Операцію логічного додавання позначають $Y = x_1 \vee x_2$. Вона означає, що: $0 \vee 0 = 0$; $0 \vee 1 = 1$; $1 \vee 0 = 1$; $1 \vee 1 = 1$, тобто $Y = 1$, якщо хоча б один аргумент дорівнює 1.

Кон'юнкція та диз'юнкція можуть виконуватися над будь-якою кількістю змінних.

У булевій алгебрі основними аналітичними формами представлення логічних функцій є диз'юнктивна та кон'юнктивна форма.

Диз'юнктивною нормальною формою (ДНФ) логічної функції є диз'юнкція (логічне додавання) будь-якої кінцевої множини попарно різних елементарних кон'юнкцій (логічних множень), наприклад, $Y(x_3, x_2, x_1) = \bar{x}_3 x_2 x_1 \vee x_3 \bar{x}_2 \vee \bar{x}_1$.

Кон'юнктивною нормальною формою (КНФ) логічної функції є кон'юнкція будь-якої кінцевої множини попарно різних елементарних диз'юнкцій, наприклад:

$$F(x_3, x_2, x_1) = (x_3 \vee x_2 \vee \bar{x}_1)(\bar{x}_2 \vee x_1) \bar{x}_3.$$

Логічна функція може бути задана словесно, алгебраїчним виразом і таблицею істинності. Наприклад, функцію $Y_{10} = f(x_2, x_1)$, задану у вигляді словесного опису: $Y_{10} = 1$, коли значення змінних $x_2 = x_1$, та $Y_{10} = 0$, коли $x_2 \neq x_1$, можна подати у вигляді таблиці істинності (табл. 2.7) або в алгебраїчній формі $Y_{10} = x_2 x_1 \vee \bar{x}_2 \bar{x}_1$. Для операцій кон'юнкції і диз'юнкції таблична форма завдання має вигляд як у табл. 3.8.

Таке подання функції називається її досконалою диз'юнктивною нормальною формою (ДДНФ).

Таблиця істинності містить усі $N = 2^n$ можливі набори значень змінних і значення функції, котрі відповідають кожному з наборів.

Щоб здійснити перехід від табличного подання до алгебраїчного, кожному числовому набору змінних ставиться у відповідність мінтерм m_i . Мінтерм (конституента одиниці) – це кон'юнкція всіх змінних, причому ті змінні, які в

наборі мають значення 1, подаються в прямому вигляді, а ті, які мають значення 0 – в інверсному.

Таблиця 3.7
Таблиця істинності
функції Y_{10}

x_2	x_1	$Y_{10} = x_2 \sim x_1$
0	0	1
0	1	0
1	0	0
1	1	1

Таблиця 3.8
Таблиця істинності функцій Y_2
та Y_8

x_2	x_1	$Y_2 = x_2 \cdot x_1$	$Y_8 = x_2 \vee x_1$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Тоді шуканий алгебраїчний вираз подається у вигляді такої логічної суми:

$$Y = \sum_{i=1}^N f_i m_i, \quad (3.1)$$

де f_i, m_i – значення функції (0 або 1) і мінтерм, які відповідають i -му числовому набору змінних; N – кількість різних наборів змінних.

Наприклад, потрібно отримати алгебраїчний вираз в ДДНФ функції Y_{10} , заданої табл. 3.7. Для цього необхідно скласти мінтерми (табл. 3.9) і скористатися виразом (3.1).

Таблиця 3.9
Мінтерми, макстерми і значення функції Y_{10}

x_2	x_1	Мінтерми	Макстерми	Значення функції Y_{10}
0	0	$m_1 = \overline{x_2} \cdot \overline{x_1}$	$M_1 = x_1 \vee x_2$	$f_1(x_2, x_1) = 1$
0	1	$m_2 = \overline{x_2} \cdot x_1$	$M_2 = x_2 \vee \overline{x_1}$	$f_2(x_2, x_1) = 0$
1	0	$m_3 = x_2 \cdot \overline{x_1}$	$M_3 = \overline{x_2} \vee x_1$	$f_3(x_2, x_1) = 0$
1	1	$m_4 = x_2 \cdot x_1$	$M_4 = \overline{x_2} \vee \overline{x_1}$	$f_4(x_2, x_1) = 1$

Отримуємо вираз

$$Y_{10} = f_1(x_2, x_1) \cdot m_1 \vee f_2(x_2, x_1) \cdot m_2 \vee f_3(x_2, x_1) \cdot m_3 \vee f_4(x_2, x_1) \cdot m_4 = 1 \cdot \overline{x_2} \cdot \overline{x_1} \vee 0 \cdot \overline{x_2} \cdot x_1 \vee 0 \cdot x_2 \cdot \overline{x_1} \vee 1 \cdot x_2 \cdot x_1 = \overline{x_2} \cdot \overline{x_1} \vee x_2 \cdot x_1.$$

Інша алгебраїчна форма подання функції виходить при використанні *макстермів*.

Макстермом (конституентою нуля) називається диз'юнкція всіх змінних, причому ті змінні, які в наборі мають значення 0, подаються в прямому вигляді, а ті, які мають значення 1 – в інверсному. Алгебраїчний вираз функції виходить у вигляді логічного множення:

$$Y = \prod_{i=1}^N (f_i \vee M_i), \quad (3.2)$$

де f_i, m_i – значення функції і макстерм, які відповідають i -му числовому набору змінних; N – кількість різних наборів змінних.

Таке подання функції називається її *досконалою кон'юнктивною нормальною формою (ДКНФ)*.

Вирази функції в ДДНФ і ДКНФ звичайно розрізняються за виглядом, але вони еквівалентні один одному. Наприклад, $Y_{10} \text{ (ДКНФ)} = \overline{x_2} \overline{x_1} + x_2 x_1$ та $Y_{10} \text{ (ДКНФ)} = (x_2 \vee \overline{x_1})(\overline{x_2} \vee x_1)$. Крім того, вирази в ДДНФ і ДКНФ часто не є найбільш простими. Використовуючи тотожності та закони булевої алгебри, можна в багатьох випадках отримати більш прості форми подання функції, які називаються *мінімізованими*.

При відносно невеликій кількості змінних ($n \leq 6$) дуже зручним і наочним є графічне подання функцій у вигляді так званих карт мінтермів, найбільш розповсюдженою формою яких вважаються карти Карно.

Зворотний перехід від алгебраїчного до табличного подання функцій виконується шляхом послідовної підстановки в даний алгебраїчний вираз всіх N можливих числових наборів змінних, визначення відповідних значень $Y = f_i(x_n, x_{n-1}, \dots, x_1)$ для кожного i -го набору та заповнення таблиці істинності.

Як відзначалося вище, для перетворення алгебраїчних виразів використовують тотожності (аксіоми) і закони булевої алгебри. Основні з них наведені в табл. 3.6.

У цій таблиці алгебраїчні вирази тотожностей і законів задані парами на підставі принципу дуальності (подвійності), відповідно до якого операції

кон'юнкції та диз'юнкції допускають взаємну заміну, якщо одночасно поміняти логічну 1 на 0, 0 на 1, знак “ $\vee$ ” на “ $\times$ ”, а “ $\times$ ” на “ $\vee$ ”.

Застосування аксіом і законів булевої алгебри покажемо на прикладі перетворення функції Y_{10} з ДКНФ у ДДНФ:

$$Y_{10} = (x_2 \vee \bar{x}_1)(\bar{x}_2 \vee x_1) = x_2\bar{x}_2 \vee x_2x_1 \vee \bar{x}_1\bar{x}_2 \vee \bar{x}_1x_1 = x_2x_1 \vee \bar{x}_1\bar{x}_2.$$

Для цього можна використовувати розподільний закон, а потім тотожність (табл.3.6).

3.4. Мінімізація логічних функцій за допомогою карт Карно

Задача мінімізації логічної функції зводиться до знаходження такої її форми, яка вміщує найменшу кількість елементарних логічних функцій, – диз'юнкцій, кон'юнкцій та інверсій.

Існує кілька способів мінімізації булевих функцій: за допомогою законів і аксіом булевої алгебри або графічні методи. Мінімізація за допомогою законів і аксіом булевої алгебри дозволяє виражати функції через різні базиси та їх комбінації, наприклад,

$$Y = (x_3x_2 \vee \overline{x_3 \vee x_2})x_1 = x_3x_1 = \overline{\overline{x_3} \vee \overline{x_1}}.$$

Серед графічних методів мінімізації логічних функцій можна виділити насамперед метод Квайна – Мак – Класки, метод Блека – Порецького, метод Петрика та метод мінімізації за допомогою карт Карно або діаграм Вейча. Розглянемо більш докладніше метод карт Карно, як найбільш розповсюджений та зручний метод, що дозволяє швидко вирішувати задачі мінімізації булевих функцій для невеликої кількості аргументів [7].

Метод Карно заснований на законі склеювання. Склеюються набори, які відрізняються один від одного значенням одного розряду. Такі набори називаються сусідніми. Карно закодував чарунки своєї карти так, що в сусідніх чарунках виявилися сусідні набори, тобто набори, які склеюються.

Карта Карно – це графічний засіб для спрощення логічних рівнянь або перетворення таблиць істинності у відповідну логічну схему простим і упоряд-

кованим шляхом. Карта Карно використовується для розв'язання задач з будь-якою кількістю вхідних змінних, однак нижче робота з картами Карно обмежена чотирма змінними, тому що задачі з великою кількістю змінних занадто складні і найкраще розв'язуються за допомогою комп'ютерних програм.

Як і таблиця істинності, карта Карно є засобом, що показує співвідношення між значенням логічних змінних та відповідного їм значення логічної функції. На рис.3.4 зображені нумерація чарунок карт Карно для трьох та чотирьох змінних, а також логічні функції, що задані таблицями істинності та картами Карно.

Ці приклади ілюструють такі важливі принципи побудови карт Карно:

1. Таблиця істинності дає значення вихідного сигналу $Y(x_n, x_{n-1}, \dots, x_1)$ для будь-якої комбінації вхідних змінних $x_n, x_{n-1}, \dots, x_1$. Карта Карно дає ту ж інформацію, але в трохи іншому вигляді. Кожен набір вхідних змінних таблиці істинності відповідає чарунці карти Карно. Наприклад, на рис. 3.4а) умова $x_3 = 0, x_2 = 0, x_1 = 0$ з таблиці істинності відповідає чарунці $\overline{x_3} \overline{x_2} \overline{x_1}$ на карті Карно. У таблиці істинності цьому випадку відповідає $Y(x_3, x_2, x_1) = 1$, отже, у карті Карно в чарунку $\overline{x_3} \overline{x_2} \overline{x_1}$ поміщується 1. Точно так, умові $x_3 = 0, x_2 = 0, x_1 = 1$ у таблиці істинності відповідає чарунка $\overline{x_3} \overline{x_2} x_1$ карти Карно. Оскільки у цьому випадку $Y(x_3, x_2, x_1) = 0$, то в чарунку $\overline{x_3} \overline{x_2} x_1$ ставиться 0. Всі інші чарунки заповнюються за аналогічними правилами. Точно така ж послідовність використовується і для карт із чотирма змінними, як це показано на рис. 3.4.

2. Квадрати карт Карно позначаються таким чином, щоб горизонтально прилягаючі чарунки відрізнялися тільки однією змінною. Наприклад, ліва верхня чарунка карти для чотирьох змінних відповідає значенню $\overline{x_4} \overline{x_3} \overline{x_2} \overline{x_1}$, а чарунка, що прилягає до неї праворуч, – $\overline{x_4} \overline{x_3} \overline{x_2} x_1$ (відрізняється тільки змінною x_1). Точно так, вертикально прилягаючі одна до одної чарунки відрізняються тільки однією змінною. Наприклад, ліва верхня чарунка відповідає наборові змінних $\overline{x_4} \overline{x_3} \overline{x_2} \overline{x_1}$, а чарунка прямо під нею – $\overline{x_4} \overline{x_3} x_2 \overline{x_1}$ (відрізняється тільки змінною x_2). Варто зауважити, що кожна чарунка у верхньому рядку вважається прилеглою до відповідної чарунки у нижньому рядку. Наприклад, чарунка $\overline{x_4} \overline{x_3} \overline{x_2} \overline{x_1}$ у верхньому рядку прилягає

до чарунки $\overline{X_4} X_3 X_2 X_1$ з нижнього рядка, тому що вони теж відрізняються тільки однією змінною X_4 . Можна вважати, що верхня частина карти як би загорнена і торкається її основи. Чарунки лівого стовпця прилягають до відповідних чарунок правого стовпця.

x_3	x_2	x_1	$Y(x_3, x_2, x_1)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Таблиця істинності для логічної функції 3-х змінних

x_3	$x_2 x_1$			
	00	01	11	10
0	1	0	1	0
1	0	1	1	0

Карта Карно для логічної функції 3-х змінних

а) Таблиця істинності та карта Карно для логічної функції 3-х змінних

x_4	x_3	x_2	x_1	$Y(x_3, x_2, x_1)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Таблиця істинності для логічної функції 4-х змінних

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Нумерація карти Карно для логічної функції 4-х змінних

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	1	1
10	0	0	0	1

Карта Карно для логічної функції 4-х змінних

б) Таблиця істинності та карта Карно для логічної функції 4-х змінних

Рис. 3.4. Карти Карно та таблиці істинності для логічних функцій трьох (а) та чотирьох (б) змінних

Щоб вертикально і горизонтально прилягаючі чарунки відрізнялися тільки однією змінною, позначення від верхньої частини до нижньої необхідно

подати у певному порядку, як це показано на рис. 2.4: $\overline{X_4} \overline{X_3}$, $\overline{X_4} X_3$, $X_4 \overline{X_3}$, $X_4 X_3$. Подібний же порядок зберігається і при позначенні чарунок з ліва направо: $\overline{X_2} \overline{X_1}$, $\overline{X_2} X_1$, $X_2 \overline{X_1}$, $X_2 X_1$.

Коли карта Карно заповнена нулями й одиницями, вираз вихідного сигналу Y у диз'юнктивній формі може бути отриманий логічним додаванням разом чарунок, які містять 1.

У карті для трьох змінних на рис. 2.4 б) одиниці містять чарунки $\overline{X_3} \overline{X_2} \overline{X_1}$, $\overline{X_3} X_2 \overline{X_1}$, $X_3 \overline{X_2} \overline{X_1}$ і $X_3 X_2 \overline{X_1}$, тому

$$Y(X_3, X_2, X_1) = \overline{X_3} \overline{X_2} \overline{X_1} + \overline{X_3} X_2 \overline{X_1} + X_3 \overline{X_2} \overline{X_1} + X_3 X_2 \overline{X_1}.$$

Групування

Вираз, отриманий для вихідного сигналу $Y(X_3, X_2, X_1)$, можна спростити правильним поєднанням у картах Карно чарунок, які містять одиниці. Процес такого підбору називається *групуванням*.

Групування чарунок попарно. На рис. 3.5 а) наведені карти Карно для двох логічних функцій трьох змінних. Ці карти містять чарунки з одиничними наборами логічних функцій, які вертикально та горизонтально прилягають одна до одної.

Так, вертикально суміжні чарунки мають позначення $\overline{X_3} X_2 \overline{X_1}$ та $X_3 X_2 \overline{X_1}$. Варто уваги, що в цих двох членах тільки перемінна X_3 змінює свій стан із прямого на інверсний, а X_2 та $\overline{X_1}$ залишаються незмінними. Ці два члени можуть бути згруповані (сполучені) таким чином, що в результаті змінну X_3 можна скоротити, оскільки вона з'являється як у нормальному стані, так і в інверсному. Це легко довести:

$$Y(X_3, X_2, X_1) = \overline{X_3} X_2 \overline{X_1} + X_3 X_2 \overline{X_1} = X_2 \overline{X_1} (X_3 + \overline{X_3}) = X_2 \overline{X_1} (1) = X_2 \overline{X_1}.$$

Такий же принцип справедливий для будь-якої пари вертикально або горизонтально прилягаючих одна до однієї чарунок з одиницями. На рис. 3.5 б) наведений приклад двох горизонтально прилягаючих одиниць. Якщо їх об'єднати, то можна буде скоротити змінну X_2 , тому що вона з'являється в позначенні чарунок і у прямій, і в інверсній формах, що дає результат

$$Y(X_3, X_2, X_1) = \overline{X_3} X_1.$$

Інший приклад показаний на рис. 3.5 в). У картах Карно, горизонтально сусідніми можуть бути чарунки, які не мають безпосередньої границі. Але, якщо уявити вертикальний циліндр, зроблений із карти Карно шляхом суміщення правої та лівої границі карти, то ці чарунки набувають загальну для них межу. Отже, дві одиниці на цій карті можна об'єднати в групу, що дасть результат:

$$Y(x_3, x_2, x_1) = \overline{x_3} \overline{x_2} \overline{x_1} + \overline{x_3} x_2 \overline{x_1} = \overline{x_3} \overline{x_1}.$$

На рис. 3.5 г) карта Карно містить дві пари чарунок з одиницями, які також можна об'єднати, хоча вони знаходяться у верхньому та нижньому рядку, оскільки верхній та нижній рядок карти Карно також вважаються сусідніми (треба уявити горизонтальний циліндр). При об'єднанні цих комірок скорочується змінна x_4 і ми отримуємо кінцевий результат:

$$Y(x_4, x_3, x_2, x_1) = \overline{x_4} \overline{x_3} \overline{x_2} x_1 + x_4 \overline{x_3} \overline{x_2} x_1 = \overline{x_3} \overline{x_2} x_1.$$

$Y(x_3, x_2, x_1) = \overline{x_2} x_1$

x_3	$x_2 x_1$			
	00	01	11	10
0	0	0	0	1
1	0	0	0	1

(а)

$Y(x_3, x_2, x_1) = \overline{x_3} x_1$

x_3	$x_2 x_1$			
	00	01	11	10
0	0	1	1	0
1	0	0	0	0

(б)

$Y(x_3, x_2, x_1) = \overline{x_3} x_1$

x_3	$x_2 x_1$			
	00	01	11	10
0	1	0	0	1
1	0	0	0	0

(в)

$Y(x_4, x_3, x_2, x_1) = \overline{x_3} \overline{x_2} x_1$

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	0	1	0	0
01	0	0	0	0
11	0	0	0	0
10	0	1	0	0

(г)

Рис. 3.5. Приклади об'єднання двох сусідніх чарунок, які містять одиниці:

- а – пара вертикально прилягаючих одиниць;
- б – пара горизонтально прилягаючих одиниць;
- в, г – дві прилягаючі одиниці у верхньому та нижньому рядках

Таким чином, об'єднання в карті Карно пари прилягаючих чарунок, які містять одиниці, приводить до скорочення тієї змінної, яка змінює свою форму в цих чарунках і логічна функція представляється у *диз'юнктивній нормальній формі* (ДНФ).

Аналогічні об'єднання комірок можуть виконуватися з комірками, що зберігають нульове значення логічної функції з тією тільки різницею, що алгебраїчний вираз для логічної функції буде представлений у *кон'юнктивній нормальній формі* (КНФ).

Групування чарунок по чотири (квартети). Карта Карно може містити групу з чотирьох чарунок з одиницями, які прилягають одна до одної. Такі групи називають квартетами. На рис. 3.6 наведені кілька прикладів таких груп. На рис. 3.6 а) чотири одиниці прилягають одна до одної по горизонталі, а на рис. 3.6 б) – по вертикалі [7].

Чарунки з одиничними наборами логічної функції на рис. 3.6 в) об'єднуються тому, що верхній і нижній рядки, а також лівий і правий стовпець карти Карно є *сусідніми*.

Чотири сусідніх чарунки в середині карти Карно можна об'єднувати контуром так, як, наприклад, показано на рис. 3.6 г). Крім того, кутові чарунки також є сусідніми і порядок їх об'єднання зображений також на рис. 3.6 г).

Коли група з чотирьох чарунок об'єднана, алгебраїчний вираз логічної функції буде містити тільки ті змінні, які не змінюють своє значення у всіх чарунках обраної групи. Наприклад, на рис. 3.6 а) чотири чарунки, що містять одиниці, відповідають наборам змінних $\overline{x_3} \overline{x_2} \overline{x_1}$, $\overline{x_3} \overline{x_2} x_1$, $\overline{x_3} x_2 \overline{x_1}$, $\overline{x_3} x_2 x_1$. Вивчення цих членів показує, що лише змінна x_3 не змінює свою форму (x_2 та x_1 з'являються як у простій, так і в інверсній формах). Таким чином, кінцевий вираз для логічної функції $Y(x_3, x_2, x_1)$ буде виглядати наступним чином: $Y(x_3, x_2, x_1) = \overline{x_3}$. Це легко довести:

$$\begin{aligned} Y(x_3, x_2, x_1) &= \overline{x_3} \overline{x_2} \overline{x_1} \vee \overline{x_3} \overline{x_2} x_1 \vee \overline{x_3} x_2 \overline{x_1} \vee \overline{x_3} x_2 x_1 = \\ &= \overline{x_3} \overline{x_2} (\overline{x_1} + x_1) \vee \overline{x_3} x_2 (\overline{x_1} \vee x_1) = \\ &= \overline{x_3} \overline{x_2} \vee \overline{x_3} x_2 = \overline{x_3} (\overline{x_2} \vee x_2) = \overline{x_3}. \end{aligned}$$

В іншому прикладі (рис. 3.6 з) чотири кутових чарунки відповідають наборам логічних змінних:

$$\overline{x_4} \overline{x_3} \overline{x_2} \overline{x_1}, \quad \overline{x_4} \overline{x_3} x_2 \overline{x_1}, \quad \overline{x_4} x_3 \overline{x_2} \overline{x_1}, \quad \overline{x_4} x_3 x_2 \overline{x_1}.$$

$$Y(x_3, x_2, x_1) = \overline{x_3}$$

x_3	$x_2 x_1$			
	00	01	11	10
0	1	1	1	1
1	0	0	0	0

(a)

$$Y(x_4, x_3, x_2, x_1) = x_2 x_1$$

$x_4 x_3$		$x_2 x_1$			
		00	01	11	10
00	0	0	0	1	0
01	0	0	0	1	0
11	0	0	0	1	0
10	0	0	0	1	0

(б)

$$Y(x_3, x_2, x_1) = \overline{x_3} x_1$$

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

$$Y(x_4, x_3, x_2, x_1) = \overline{x_3} x_1$$

(в)

$$Y(x_4, x_3, x_2, x_1) = x_3 x_1$$

$x_4 x_3$		$x_2 x_1$			
		00	01	11	10
00	1	0	0	1	0
01	0	1	1	0	0
11	0	1	1	0	0
10	1	0	0	1	0

$$Y(x_4, x_3, x_2, x_1) = \overline{x_3} \overline{x_1}$$

(г)

Рис. 3.6. Типові варіанти об'єднання по чотири сусідніх чарунки, з одними наборами логічної функції:

- а – горизонтальне об'єднання сусідніх чарунок;
- б – вертикальне об'єднання сусідніх чарунок;
- в, г – об'єднання зовнішніх сусідніх чарунок

Аналіз цих членів виразу для вихідного сигналу показує, що тільки перемінні $\overline{x_3}$ і $\overline{x_1}$ залишаються незмінними, тому вже спрощений вираз для $Y(x_4, x_3, x_2, x_1)$ буде мати вигляд:

$$Y(x_4, x_3, x_2, x_1) = \overline{x_3} \overline{x_1},$$

що можна довести так само, як у попередньому прикладі. В інших випадках, наведених на рис. 3.6, вирази, зазначені біля карт Карно, теж відповідають спрощеним виразам для логічних функцій.

Підвівши підсумки, можна сказати, що групування чотирьох прилягаючих чарунок, дозволяє скоротити ті перемінні, які змінюють своє значення в чарунках, що об'єднані контуром.

Групування чарунок по вісім (октети)

Група з восьми одиниць, які прилягають одна до одної, іноді називається ще октетом. Кілька типових прикладів октетів наведені на рис. 3.7.

Коли октет групується на карті з чотирма змінними, скорочуються відразу три з чотирьох змінних, тому що тільки одна залишається незмінною у всіх восьми чарунках.

Наприклад, аналіз восьми згрупованих чарунок на рис. 3.7а) показує, що тільки змінна x_2 залишається в одній і тій же формі незалежно від чарунки. Інші ж змінні зустрічаються то в прямій, то в інверсній формах, тобто для цієї карти кінцевий вираз буде $Y = \overline{x_2}$. Можна перевірити результат і в інших прикладах на рис. 3.7.

Отже, групування октету (восьми прилягаючих чарунок, які містять одиниці) дозволяють скоротити три змінні, котрі змінюють свої форми від чарунки до чарунки.

Як було показано вище, групування пар, квартетів і октетів у картах Карно можуть бути використані для спрощення аналітичного виразу логічної функції. Сформулюємо правило об'єднання груп чарунок будь-якого розміру:

якщо логічна змінна в отриманій групі зустрічається як у прямій, так і в інверсній формах, її можна скоротити (не використовувати);

змінні, які не змінюють свою форму від чарунки до чарунки цієї групи, повинні залишитися й у кінцевому аналітичному виразі логічної функції.

Таким чином, чим більша група об'єднаних чарунок, тим більше змінних можна буде скоротити, тобто група з двох чарунок дозволяє скоротити одну змінну, група з чотирьох – дві, а група з восьми – три змінні. Цей принцип використовується для одержання за допомогою карт Карно спрощеного виразу для логічної функції, одиничні або нульові набори якої, можна об'єднати у відповідні групи.

Для мінімізації логічних функцій на карті Карно проводять максимально правильні контури. *Максимально правильний контур* – це контур, який не становить частину другого контуру і охоплює $1, 2, 4, \dots, 2^n$ чарунок, які відповідають одиничним (нульовим), або одиничним (нульовим) та невизначеним набо-

рам логічної функції, тобто контур може охоплювати не тільки одиничні або нульові набори, але й одиничні з невизначеними, або нульові з невизначеними.

$$Y(x_4, x_3, x_2, x_1) = \overline{x_2}$$

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	1	1	0	0
10	1	1	0	0

(а)

$$Y(x_4, x_3, x_2, x_1) = x_3$$

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

(б)

$$Y(x_3, x_2, x_1) = \overline{x_1}$$

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

(в)

$$Y(x_4, x_3, x_2, x_1) = \overline{x_3}$$

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

(г)

Рис. 3.7. Приклади об'єднання сусідніх чарунок, які містять одиниці, по вісім (октетами):

а,б – одиниці всередині карти;

в,г – одиниці по краях карти

Розглянемо кілька прикладів процедури спрощення (мінімізації) шляхом об'єднання чарунок з одиничними значеннями логічних функцій. Нижче описується алгоритм дій, якими варто керуватися при спрощенні (мінімізації) булевих виразів.

Етап 1. Будується карта Карно і розташовуються одиниці та нулі в ті чарунки, що відповідають одиничним та нульовим значенням логічної функції. Усі інші чарунки відповідають невизначеним наборам логічної функції.

Етап 2. Аналізується карта Карно, знаходяться області прилягаючих (сусідніх) комірок і виділяються ті комірки, для яких відсутні прилягаючі (сусідні) комірки. Вони називаються ізольованими комірками.

Етап 3. Знаходяться комірки, які прилягають тільки до ще однієї комірки. Вони групуються в будь-які пари прилягаючих (сусідніх) комірок.

Етап 4. Грукуються всі октети, навіть якщо вони будуть містити комірки з одиничними (нульовими або невизначеними) наборами логічної функції, які вже раніше були згруповані.

Етап 5. Грукуються всі квартети, які містять одну або кілька комірок, які ще не були згруповані. При цьому треба переконатись, що використовується мінімальна кількість груп.

Етап 6. Грукуються будь-які пари, які необхідні, щоб включити ті комірки, що ще не були згруповані. Треба переконатись, що використовується мінімальна кількість груп.

Етап 7. Підсумовуються всі члени, які описані кожною групою.

Ці етапи повинні виконуватися суворо у тій послідовності, у якій вони тут приведені. Використовуються вони при розв'язуванні наступних прикладів при обробці карт Карно. У кожному випадку отриманий логічний вираз буде мати мінімальну диз'юнктивну форму.

На рис. 3.8 показана карта Карно для логічної функції чотирьох змінних. Припустимо, що карта була отримана з таблиці істинності (*етан 1*). Для зручності складання груп всі чарунки пронумеровані.

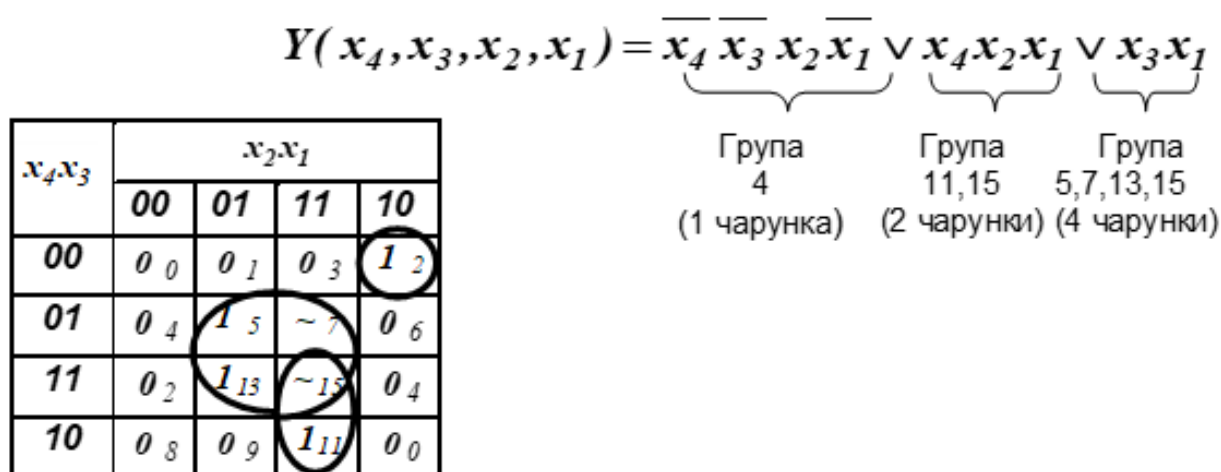


Рис. 3.8. Мінімізація логічної функції 4-х змінних за допомогою карти Карно

Розв'язання:

Етап 2. Чарунка 4 – єдина чарунка, яка містить одиницю, що не прилягає до жодної іншої одиниці. Вона виділяється у власну групу – групу 4.

Етап 3. Чарунка 15 прилягає тільки до чарунки 11. Ця пара теж групується; позначимо її групою 11, 15.

Етап 4. На даній карті немає октетів.

Етап 5. Чарунки 6, 7, 10 і 11 формують квартет. Помітимо, що чарунка 11 використовується вже вдруге.

Етап 6. Всі одиниці на карті вже згруповані.

Етап 7. Кожна група утворить свій мінтерм у виразі для сигналу Y . Група 4 дає просто $\overline{X_4} \overline{X_3} \overline{X_2} \overline{X_1}$. Група з чарунок 11 і 15 дає мінтерм $X_4 X_2 X_1$ (змінна x_3 скорочується). Чарунки 6, 7, 10 і 11 дають $X_3 X_1$ (скорочуються X_4 і X_2). Часто під час аналізу та синтезу логічних функцій виникає необхідність заповнити карту Карно, виходячи з аналітичного, цифрового, або мовного опису логічної функції. В цьому випадку необхідно діяти за наступним алгоритмом:

1. Отримати вираз у диз'юнктивній формі, якщо це ще не зроблено.

2. Для кожного мінтерма в отриманому виразі поставити одиницю в чарунку, яка відповідає даному членові (позначена тією ж комбінацією вхідних змінних). В усі інші чарунки поставити нулі.

Наприклад, використаємо карту Карно для спрощення виразу

$$Y(X_4, X_3, X_2, X_1) = \overline{X_2}(\overline{X_4} \overline{X_3} \overline{X_1} \vee X_1) \vee X_4 \overline{X_3} X_2 \vee \overline{X_1}:$$

а) розкривши дужки, отримаємо

$Y(X_4, X_3, X_2, X_1) = \overline{X_4} \overline{X_3} \overline{X_2} \overline{X_1} \vee \overline{X_2} X_1 \vee X_4 \overline{X_3} X_2 \vee \overline{X_1}$, тобто вираз у диз'юнктивній формі;

б) щоб врахувати член $\overline{X_4} \overline{X_3} \overline{X_2} \overline{X_1}$, поставимо у відповідну чарунку карти Карно одиницю (рис. 3.9). Для члена $\overline{X_2} X_1$ одиниці поміщаємо в усі ті чарунки, у позначенні яких присутній $\overline{X_2} X_1$: це члени $\overline{X_4} \overline{X_3} \overline{X_2} X_1$, $\overline{X_4} X_3 \overline{X_2} X_1$, $X_4 \overline{X_3} \overline{X_2} X_1$, $X_4 X_3 \overline{X_2} X_1$.

Для члена $X_4 \overline{X_3} X_2$ одиниці ставляться в ті чарунки, які містять $X_4 \overline{X_3} X_2$ у своїх позначеннях: $X_4 \overline{X_3} X_2 \overline{X_1}$, $X_4 \overline{X_3} X_2 X_1$. І нарешті, для члена $\overline{X_1}$

розставляємо одиниці в усі чарунки, у позначеннях яких є $\overline{X_1}$ – тобто в усіх чарунках крайнього лівого і самого крайнього правого стовпців.

x_4x_3	x_2x_1			
	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	1	1

$$Y(x_4, x_3, x_2, x_1) = \overline{x_4} \overline{x_3} \overline{x_2} \overline{x_1} \vee \overline{x_4} x_2 x_1 \vee x_3 x_1$$

Рис. 3.9. Порядок заповнення карти Карно

Тепер, коли карта Карно заповнена, розбиваємо її на групи, щоб спростити вираз. Можна переконатися в тому, що після правильного групування вийде вираз $Y(x_4, x_3, x_2, x_1) = \overline{x_4} \overline{x_3} \overline{x_2} \overline{x_1} \vee \overline{x_4} x_2 x_1 \vee x_3 x_1$.

Підводячи підсумок розглянутого питання мінімізації логічних функцій, можна зробити висновок, що робота з картами Карно становить упорядкований процес з чіткими та визначеними кроками виконання яких завжди дають мінімальну форму для кінцевого виразу логічної функції [1].

3.5 Синтез комбінаційних пристроїв

Метою синтезу комбінаційного пристрою є розробка його принципової схеми. Розроблена схема повинна задовольняти таким вимогам:

реалізовувати задану логічну функцію;

використовувати задані логічні елементи;

бути оптимальною з необхідної точки зору (за швидкодією, за споживаною енергією тощо).

Синтез комбінаційного пристрою описується алгоритмом:

задається логіка функціонування пристрою за допомогою таблиці істинності;

з таблиці істинності визначаються логічні функції, які мають бути реалізовані пристроєм що синтезується;

логічні функції приводяться до заданого базису логічних елементів або оптимізуються;

за отриманими логічними виразами будується функціональна схема пристрою;

для побудови принципової схеми підбираються необхідні інтегральні схеми логічних елементів;

на принципову схему наносяться умовні позначення логічних елементів і їх номери виводів, відображаються пояснюючі додаткові дані.

Розглянемо приклад реалізації комбінаційного пристрою в базисі *I-HE*, функціонування якого описується таблицею істинності (табл.3.9).

Таблиця 3.9

Таблиця істинності комбінаційного пристрою

Входи		Виходи		
x_1	x_0	$Y_2(x_1, x_0)$	$Y_1(x_1, x_0)$	$Y_0(x_1, x_0)$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Визначення логічних виразів для виходів пристрою

Для кожного виходу пристрою, що синтезується, одержують вираз логічної функції, яка має реалізовуватися цим пристроєм. Тому дані табл. 3.9 необхідно розглянути три рази – за кількістю виходів.

Для логічної функції виходу $Y_2(x_1, x_0)$ будується табл. 3.10.

Для входів майбутнього пристрою розглядаються всі можливі комбінації їх станів. Звичайно це відображення номера рядка в двійковій формі (табл. 3.11).

Початковій комбінації присвоїмо номер 0 , а кінцевій – номер $2^N - 1$ в двійковій системі числення, де N – кількість входів. Повний набір входних впливів включає 2^N комбінацій.

Як впливає з аналізу табл. 3.11, кожна комбінація є двійковим поданням номера рядка.

Мінтерм для кожного рядка одержують як добуток входних змінних. Причому, якщо входна змінна в даному рядку подана як 0 , то вона включається в добуток в інверсній формі: $x_i = 0 \Rightarrow \overline{x_i}$. А якщо входна змінна в даному рядку подана як 1 , то вона включається в добуток у прямій формі: $x_i = 1 \Rightarrow x_i$.

Таблиця 3.10

Таблиця істинності логічної
функції $Y_2(x_1, x_0)$

Входи		Мінтерми	Вихід	Добуток
x_1	x_0		$Y_2(x_1, x_0)$	
0	0	$\overline{x_1} \cdot \overline{x_0}$	0	$0 \cdot \overline{x_1} \cdot \overline{x_0}$
0	1	$\overline{x_1} \cdot x_0$	0	$0 \cdot \overline{x_1} \cdot x_0$
1	0	$x_1 \cdot \overline{x_0}$	1	$1 \cdot x_1 \cdot \overline{x_0}$
1	1	$x_1 \cdot x_0$	0	$0 \cdot x_1 \cdot x_0$

Таблиця 3.11

Приклад кодування
номера рядка

Номер рядка	x_2	x_1	x_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Логічний вираз виходить як сума добутків значення виходу на відповідний мінтерм, кількість доданків завжди дорівнює кількості рядків таблиці істинності:

$$Y_2(x_1, x_0) = 0 \cdot \overline{x_1} \cdot \overline{x_0} + 0 \cdot \overline{x_1} \cdot x_0 + 1 \cdot \overline{x_1} \cdot x_0 + 0 \cdot x_1 \cdot x_0.$$

Спростимо даний вираз. Очевидно, що добуток на 0 завжди буде дорівнювати 0. Тому частину доданків можна виключити. Також можна виключити постійний множник 1. Підсумкове значення для Y_2 :

$$Y_2(x_1, x_0) = 1 \cdot \overline{x_1} \cdot x_0 = \overline{x_1} \cdot x_0.$$

Аналогічно одержимо логічні вирази для інших виходів.

Для $Y_1(x_1, x_0)$:

$$Y_1(x_1, x_0) = 1 \cdot \overline{x_1} \cdot \overline{x_0} + 0 \cdot \overline{x_1} \cdot x_0 + 0 \cdot \overline{x_1} \cdot x_0 + 1 \cdot x_1 \cdot x_0.$$

Після спрощення: $Y_1(x_1, x_0) = \overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0$.

Для $Y_0(x_1, x_0)$:

$$Y_0(x_1, x_0) = 0 \cdot \overline{x_1} \cdot \overline{x_0} + 1 \cdot \overline{x_1} \cdot \overline{x_0} + 0 \cdot \overline{x_1} \cdot x_0 + 0 \cdot x_1 \cdot x_0.$$

Після спрощення: $Y_0(x_1, x_0) = \overline{x_1} \cdot \overline{x_0}$.

Перетворення логічних виразів у заданий базис (НІ-І)

Перетворення логічних виразів здійснюється з метою їх спрощення. Для спрощення логічних виразів використовуються аксіоми, тотожності та теореми алгебри логіки, а також різні методи мінімізації. Зазвичай для перетворення виразів досить використовувати правило де Моргана і закон подвійної інверсії.

Проілюструємо їх практичне застосування. Для одержання значення $Y_2(x_1, x_0)$ необхідно виконати операцію логічного множення I . Однак, за завданням, базовою операцією є операція $HI-I$. Тому застосування закону подвійної інверсії є вимушеною мірою і приводить до штучного ускладнення логічного виразу:

$$Y_2(x_1, x_0) = \overline{\overline{x_1} \cdot x_0} = \overline{\overline{\overline{\overline{x_1} \cdot x_0}}}$$

Розгляд рівняння $Y_1(x_1, x_0) = \overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0$ показує, що для його реалізації потрібне виконання як операцій логічного множення, так і логічного додавання. Перетворимо операцію логічного $АБО$ додаванням в операцію логічного множення I за допомогою правила де Моргана:

$$Y_1(x_1, x_0) = \overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0 = \overline{\overline{\overline{\overline{\overline{x_1} \cdot x_0} + \overline{x_1} \cdot \overline{x_0}}}}$$

Як впливає з розгляду отриманого виразу, застосування правила де Моргана не тільки дозволило змінити операцію $АБО$ на операцію $HI-I$, але і замінило окремі операції I на $HI-I$. Таким чином, всі операції, що складають заданий вираз, є операціями $HI-I$. Перетворення логічного виразу для виходу $Y_0(x_1, x_0)$ виконується аналогічно перетворенню для $Y_2(x_1, x_0)$:

$$Y_0(x_1, x_0) = x_1 \cdot \overline{x_0} = \overline{\overline{\overline{\overline{x_1} \cdot \overline{x_0}}}}$$

Побудова принципової схеми

Побудова принципової схеми полягає в графічному відображенні виконуваних операцій за допомогою умовних графічних позначень логічних елементів. Очевидно, що схема повинна мати кількість входів, що відповідає кількості вхідних змінних і кількість виходів, що відповідає кількості вихідних змінних. На рис. 3.10 показана схема, яка реалізує логічний вираз для $Y_2(x_1, x_0)$. У даній схемі використано три логічні елементи $HI-I$. Два з них (DD1, DD3) виконують операцію інверсії, тому вони можуть бути замінені логічними елементами – інверторами (рис. 2.11).

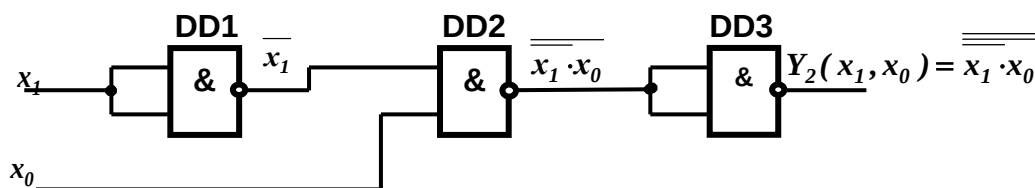


Рис. 3.10. Реалізація логічної функції $Y_2(x_1, x_0)$ на логічних елементах $HI-I$

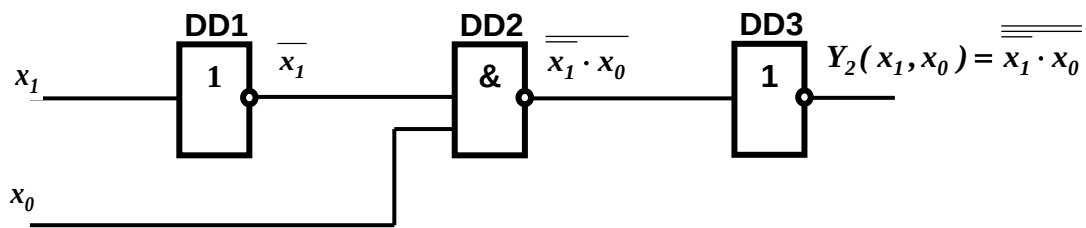


Рис. 3.11. Реалізація логічної функції $Y_2(x_1, x_0)$ на логічних елементах НІ та НІ-І

Аналогічно виходить схема для виходу $Y_0(x_1, x_0)$ (рис. 3.12).

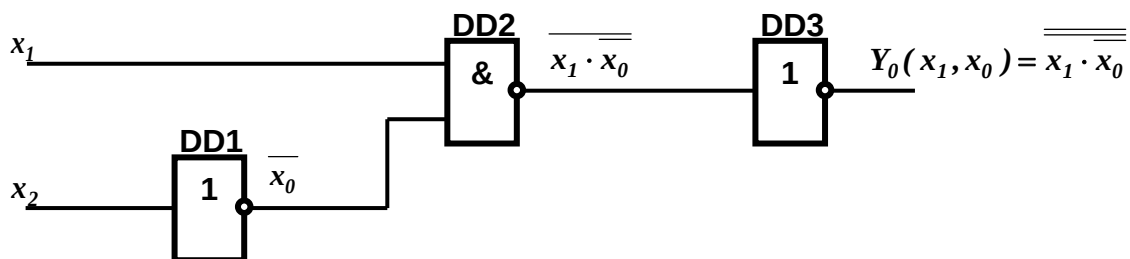


Рис. 3.12. Реалізація логічної функції $Y_0(x_1, x_0)$ на логічних елементах НІ та НІ-І

Схема для виходу $Y_1(x_1, x_0)$ ненабагато складніша (рис. 3.13).

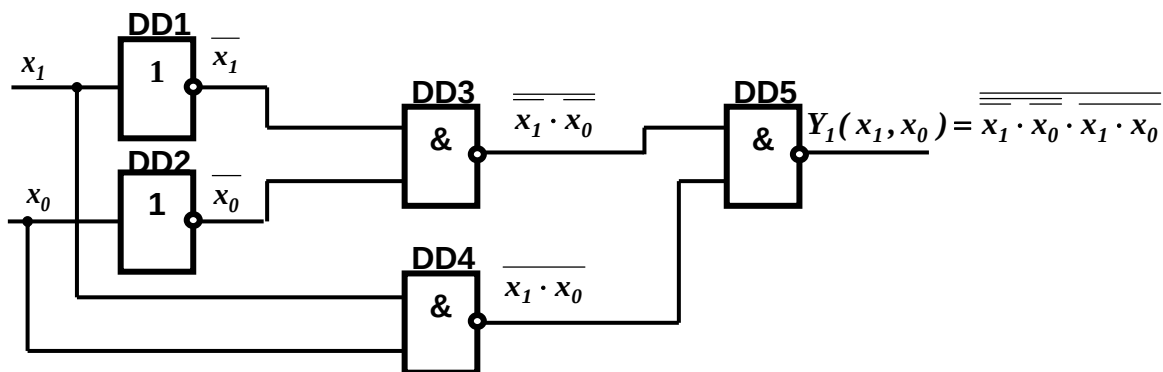


Рис. 3.13. Реалізація логічної функції $Y_1(x_1, x_0)$ на логічних елементах НІ та НІ-І

При з'єднанні цих трьох схем в одну одержимо схему на рис. 3.14.

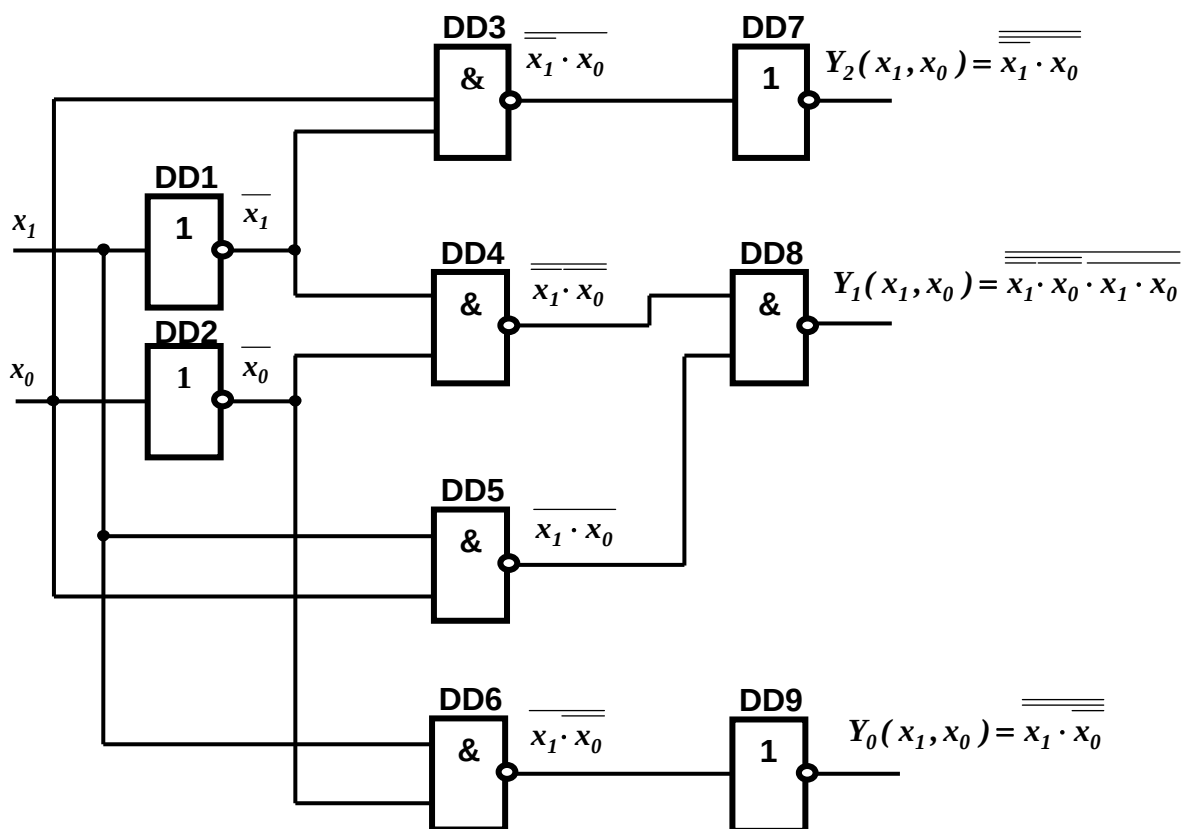


Рис. 3.14. Реалізація комбінаційного пристрою на логічних елементах

Отримана схема є функціональною. Вона відрізняється від схеми електричної принципової відсутністю конкретизації типу ІС.

Після вибору за довідником типів ІС функціональна схема доповнюється специфікацією, номерами логічних елементів та їхніх виводів [7].

КОНТРОЛЬНІ ПИТАННЯ

1. Дати визначення комбінаційному цифровому автомату та цифровому автомату з пам'яттю.
2. Дати визначення логічної змінної та логічної функції. Яка логічна функція називається не повністю визначеною?
3. Навести всі логічні функції однієї змінної.
4. Яке співвідношення між числом логічних змінних та кількістю різних наборів логічної функції?
5. Розкрити суть поняття функціонально повної системи логічних функцій. Навести приклади таких систем.
6. Скільки різних логічних функцій можна утворити від n логічних змінних?

7. Пояснити принцип побудови досконалої диз'юнктивної (кон'юнктивної) нормальної форми логічної функції.

8. За допомогою аксіом алгебри логіки отримати спрощення виразу логічної функції $f(x_3, x_2, x_1) = x_1 \vee x_2 x_3 x_2 x_1$.

9. Пояснити принцип мінімізації логічних функцій за допомогою карт Карно.

10. Логічна функція чотирьох змінних задана номерами одиничних **(1,8,10,12,13)** та нульових **(0,2,3,5,6)** наборів. За допомогою карт Карно визначити мінімальну диз'юнктивну та кон'юнктивну нормальну форму логічної функції.

11. Представити логічну функцію чотирьох змінних $f(x_4, x_3, x_2, x_1) = x_4 x_3 x_2 \vee x_3 x_2$ в табличній та досконалих диз'юнктивній та кон'юнктивній нормальних формах.

12. Для заданої логічної функції чотирьох змінних визначити всі номери чарунок карти Карно, в яких логічна функція приймає одиничні значення $f(x_4, x_3, x_2, x_1) = x_4 \vee x_4 x_3 x_1 \vee x_3 x_2 x_1$.

13. Для заданої логічної функції чотирьох змінних визначити всі номери чарунок карти Карно, в яких логічна функція приймає нульові значення $f(x_4, x_3, x_2, x_1) = x_3(x_4 \vee x_3 \vee x_2)(x_4 \vee x_2 \vee x_1)$.

14. Записати виділені контури (групи одиничних та нульових наборів) логічної функції чотирьох змінних в аналітичному вигляді.

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	0	1	1	0
01	0	0	0	0
11	1	0	1	1
10	0	1	1	0

$x_4 x_3$	$x_2 x_1$			
	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	1	1	0	0
10	0	0	1	0

15. Знайти всі мінімальні диз'юнктивні та кон'юнктивні нормальні форми не повністю заданої логічної функції, що наведена у вигляді двоходовою таблицею відповідності (картою Карно).

x_4x_3	x_2x_1			
	00	01	11	10
00	1	~	0	0
01	0	0	1	~
11	0	1	1	~
10	~	~	1	0

16. Логічна функція трьох змінних задана одновходовою таблицею відповідності. Записати її в диз'юнктивній та кон'юнктивній нормальній формі. Реалізувати її на елементах *НЕ-І*.

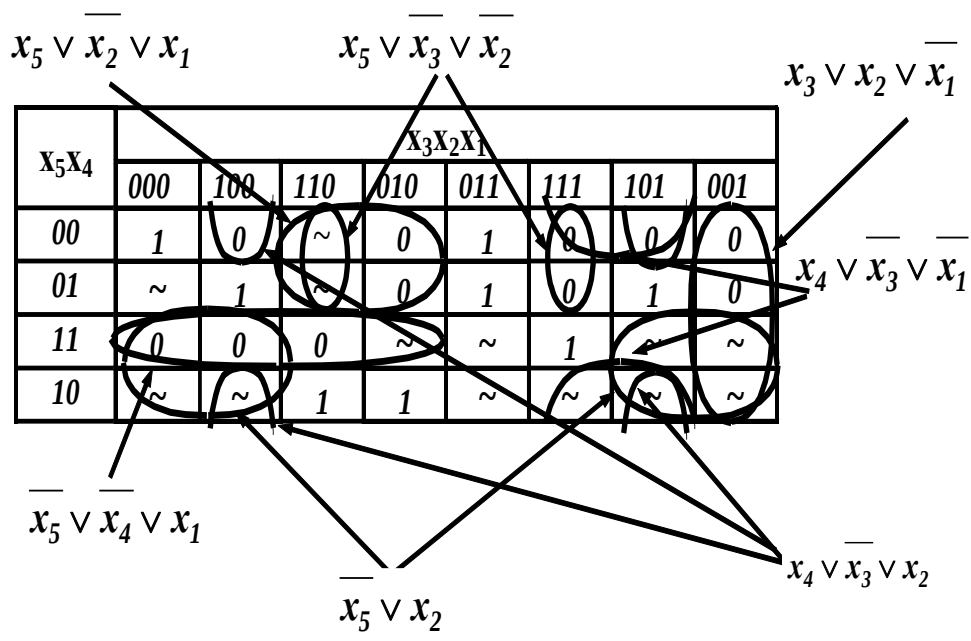
x_3	x_2	x_1	$Y(x_3, x_2, x_1)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

17. Для логічної функції, що задана в п.18, отримати спрощений вираз, використовуючи для цього карту Карно. Порівняти отриманий результат з результатом мінімізації логічної функції за допомогою аксіом алгебри логіки.

18. Неповністю визначена логічна функція п'яти змінних представлена двоходовою таблицею відповідності (картою Карно). Записати логічні вирази для виділених контурів одиничних наборів логічної функції.

x_5x_4	$x_3x_2x_1$							
	000	100	110	010	011	111	101	001
00	1	0	~	0	1	0	0	0
01	~	1	~	0	1	0	1	0
11	0	0	0	~	~	1	~	~
10	~	~	1	1	~	~	~	~

19. Перевірити правильність визначення нульових наборів логічної функції п'яти змінних. Записати мінімальну кон'юнктивну нормальну форму для цієї логічної функції.



4. ЦИФРОВІ ПРИСТРОЇ КОМБІНАЦІЙНОГО ТИПУ

На логічних елементах, виконаних у вигляді окремих *ІС*, можна створювати довільні цифрові пристрої, однак необхідно пам'ятати, що ці пристрої будуть мати значні розміри, споживати багато енергії і мати низьку надійність. Тому в мікроелектроніці пішли шляхом виділення найбільш розповсюджених класів пристроїв і реалізації цих пристроїв конструктивно та функціонально завершеними інтегральними мікроелементами часу.

Функціонування комбінаційного пристрою, як і функціонування логічного елемента, може описуватися логічним виразом і таблицею істинності.

Основою класифікації комбінаційних пристроїв є їхнє призначення. Розрізняють такі типи комбінаційних пристроїв:

- дешифратори;
- шифратори;
- мультиплексори;
- демультиплексори;
- суматори;
- компаратори;
- арифметико-логічні пристрої.

Основні характеристики *ІС* комбінаційних пристроїв збігаються з характеристиками логічних елементів:

а) електричні характеристики:

- логічні рівні одиниці U^1 і нуля U^0 ;
- вхідні/вихідні струми логічного нуля і логічної одиниці;
- споживана потужність P_{Cn} ;

б) динамічні характеристики:

- час затримки перемикання з логічного нуля в одиницю $t_{3m}^{0,1}$;
- час затримки перемикання з логічної одиниці в нуль $t_{3m}^{1,0}$;
- середній час затримки перемикання $t_{3m_{Cep}}$.

4.1. Дешифратори

Дешифратором називається пристрій комбінаційного типу, призначений для перетворення двійкового коду у будь-який інший (шістнадцятковий, десятковий та ін.).

Умовне графічне позначення дешифратора наведене на рис. 4.1.

Призначення виводів:

входи 1, 2, 4 – для подачі двійкового коду; варто підкреслити, що входи дешифратора позначаються вагами двійкових розрядів, це полегшує визначення активного виходу;

виходи – для одержання вихідного коду; у будь-який момент часу в дешифраторі може бути активним тільки єдиний вихід; якщо виходи в дешифратора прямі, то активним станом буде вважатися стан логічної одиниці, якщо виходи в дешифратора інверсні, то активним станом буде вважатися стан логічного нуля.

Двовхідний лінійний дешифратор

Приклади умовного графічного позначення лінійних дешифраторів наведені на рис. 4.1 та рис. 4.2.

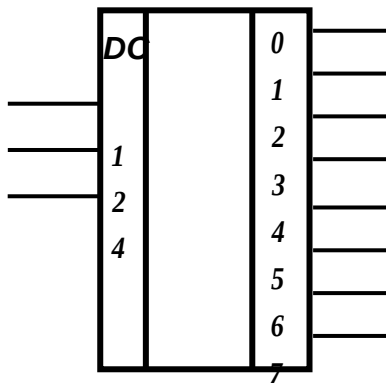


Рис. 4.1. Умовне графічне позначення трирозрядного дешифратора

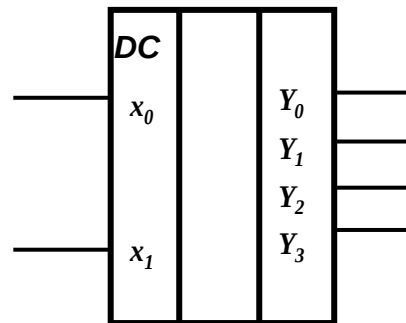


Рис. 4.2. Умовне графічне позначення двовхідного лінійного дешифратора

У таблицю істинності двовхідного лінійного дешифратора (табл. 4.1) додатково введений стовпчик десяткового значення вхідного коду – *DEC*. Порівнюючи значення вхідного коду і стану виходів можна помітити, що логічна одиниця буде присутня тільки на одному виході, номер якого збігається зі значенням вхідного коду. На всіх інших виходах будуть присутні логічні нулі [8].

Логічний вираз для дешифратора легко визначити з таблиці істинності:

$$Y_0 = \overline{x_1} \cdot \overline{x_0}; \quad Y_1 = \overline{x_1} \cdot x_0; \quad Y_2 = x_1 \cdot \overline{x_0}; \quad Y_3 = x_1 \cdot x_0.$$

За отриманими логічними виразами будується функціональна схема (рис. 4.3).

Таблиця істинності двовхідного лінійного дешифратора

x_1	x_0	DEC	$Y_3(x_1, x_0)$	$Y_2(x_1, x_0)$	$Y_1(x_1, x_0)$	$Y_0(x_1, x_0)$
0	0	0	0	0	0	1
0	1	1	0	0	1	0
1	0	2	0	1	0	0
1	1	3	1	0	0	0

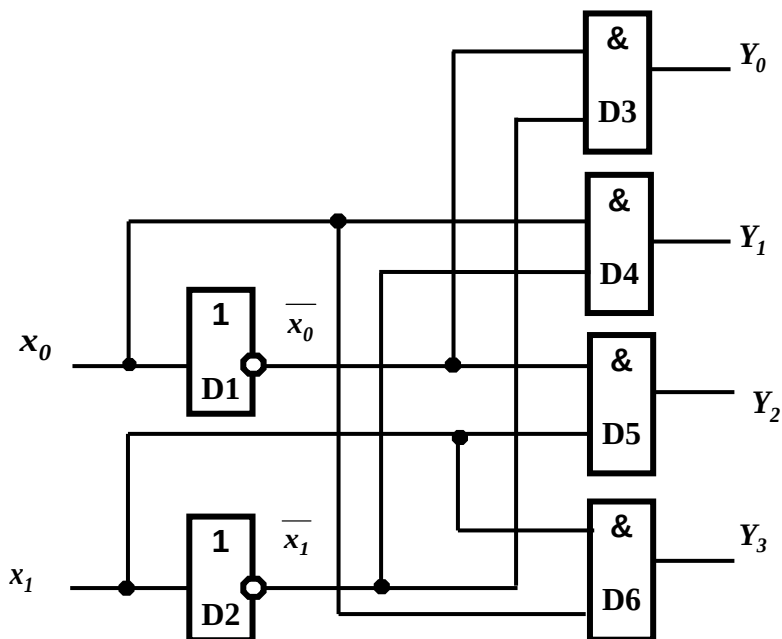


Рис. 4.3. Функціональна схема двовхідного лінійного дешифратора

Розглядаючи побудову дешифраторів з більшою, ніж 2, кількістю входів можна дійти до деяких висновків.

Логічний вираз для довільного виходу буде складатися з добутку усіх входних змінних у різній формі (інверсній або прямій).

Наявність або відсутність інверсії входної змінної буде визначатися номером логічного виразу, наприклад, для **13**-го виходу в інверсній формі буде записаний операнд x_1 , якому в двійковому коді числа **13** відповідає 0:

$$13_2 = 1 \quad 1 \quad 0 \quad 1$$

$$Y_{13} = x_3 \wedge x_2 \wedge \overline{x_1} \wedge x_0$$

Для побудови лінійного дешифратора необхідно мати інвертори (за кількістю входних операндів) та логічні елементи I , з кількістю входів, що дорівнюють кількості входних операндів. Зі збільшенням кількості входів у дешиф-

раторі зростає й кількість входів логічних елементів I . Існують технологічні обмеження з виготовлення логічних елементів з кількістю входів більше 6, тому дана схема побудови дешифраторів не може бути використана при побудові багатовхідних дешифраторів.

Швидкодія лінійного дешифратора буде визначатися затримкою переми-
кання вхідних інверторів і вихідних логічних елементів I та не залежить від кількості входів або виходів лінійного дешифратора:

$$T_{Пер} = t_{HE} + t_I = 2t_{3m_{Сер}}.$$

Перевага лінійних дешифраторів – висока швидкодія.

Недолік – неможливість нарощування розрядності через обмеження кількості входів логічних елементів.

Різновиди дешифраторів

Пірамідальний дешифратор. Перетворивши вихідний вираз для довільного виходу дешифратора

$$Y_{13} = (((x_3 \wedge x_2) \wedge \overline{x_1}) \wedge x_0)$$

можна одержати схему пірамідального дешифратора.

Функціональна схема пірамідального дешифратора зображена на рис. 4.4.

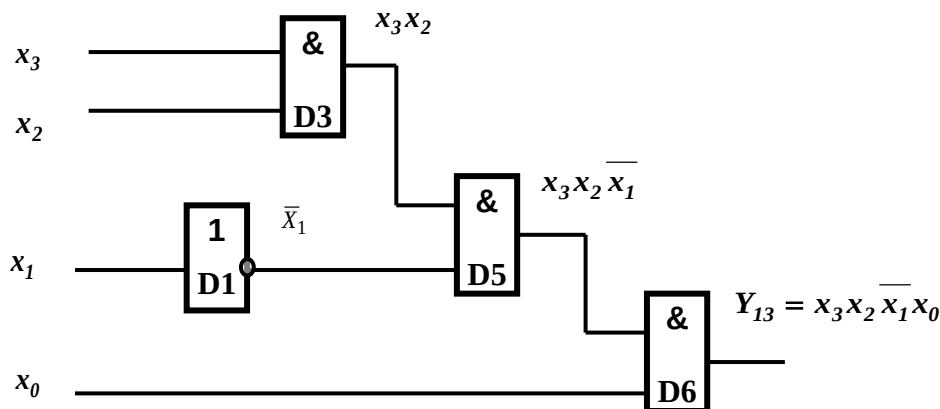


Рис. 4.4. Функціональна схема пірамідального дешифратора

Його переваги:

- використовуються однотипні логічні елементи;
- немає обмежень нарощування розрядності.

Недоліки:

- низька економічність (визначається кількістю логічних елементів);

- низька швидкодія (зі схеми видно, що зі зростанням кількості входів, зростає кількість логічних елементів, ввімкнених послідовно, і зростає час перемикачів).

Прямокутний дешифратор. Вихідний логічний вираз можна подати й в іншому вигляді:

$$Y_{13} = (x_3 \wedge x_2) \wedge (\overline{x_1} \wedge x_0),$$

$$Y_{13} = Y_{2-3} \times Y_{2-1},$$

де Y_{2-3} – вихід двовхідного дешифратора старших розрядів і Y_{2-1} – вихід 1 двовхідного дешифратора молодших розрядів.

Характерними рисами такого дешифратора, названого прямокутним, будуть два ступеня дешифрації: перший ступінь буде формуватися двома двовхідними дешифраторами (для входів x_3 та x_2 – дешифратор рядків, x_1 та x_0 – дешифратор стовпців); другий – перемножуванням вихідних значень двовхідних дешифраторів, як показано на рис. 4.5.

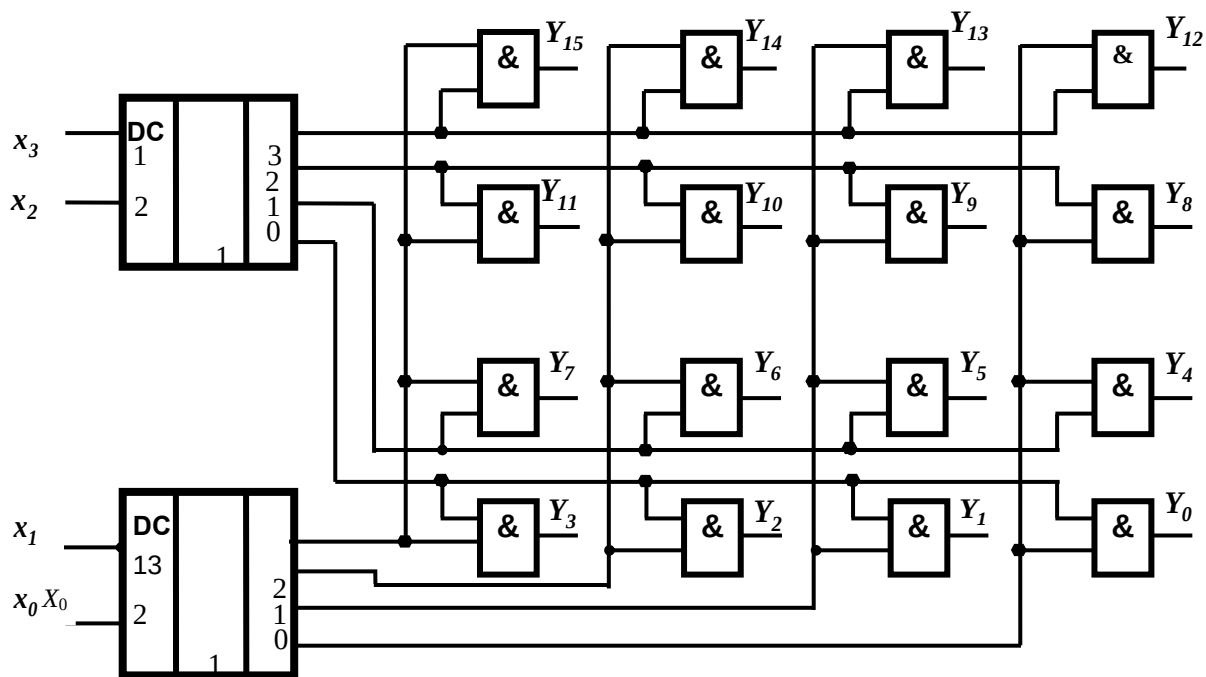


Рис. 4.5. Функціональна схема прямокутного дешифратора

Переваги:

- прийнятна швидкодія;
- немає обмежень нарощування розрядності.

Недолік – уступає за швидкістю лінійним дешифраторам.

Прикладом неповного дешифратора може служити мікросхема 555ИД10 (рис. 4.6). У ній виключене дешифрування значень, старше 9. Тому вона може використовуватися для дешифрації двійково-десятькового коду (коли кожна десяткова цифра кодується своїм чотирьохрозрядним кодом).

Дешифратори з інверсними виходами. Для деяких окремих реалізацій, наприклад, для керування індикаторними пристроями, потрібно одержувати не прямі, а інверсні коди чисел. У таких ситуаціях доцільно мати інверсну таблицю істинності для дешифратора (табл. 4.2). Активний стан виходу в цьому випадку приймають таким, що дорівнює 0. Активним, у будь-який момент часу може бути тільки один вихід, а інші повинні знаходитися в пасивному стані, тобто в стані логічної одиниці.

Дешифратори, функціонування яких описується такою таблицею істинності, називаються дешифраторами з інверсними виходами і на умовних графічних позначеннях виходи таких пристроїв мають додаткове позначення операції інверсії.

Для керування високовольтним навантаженням (з напругою живлення більшою, ніж напруга живлення використовуваної мікросхеми) вихідні каскади дешифраторів можуть будуватися за схемою з відкритим колектором.

У цьому випадку на умовному графічному позначенні дешифратора у полі виходів розташовують відповідне позначення (рис. 4.6).

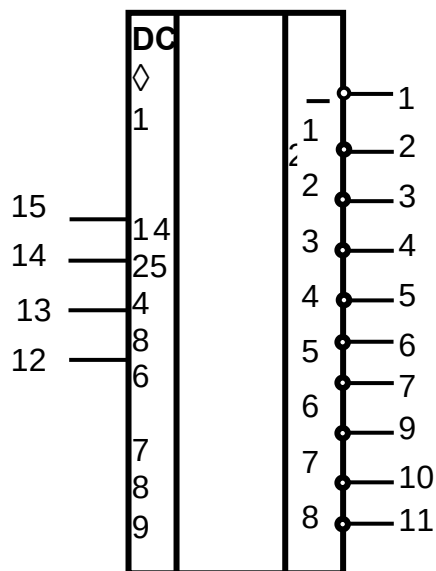


Рис. 4.6. Умовне графічне позначення ІС 555ИД10

Таблиця 4.2

Інверсна таблиця істинності дешифратора

x_1	x_0	DEC	$Y_3(x_1, x_0)$	$Y_2(x_1, x_0)$	$Y_1(x_1, x_0)$	$Y_0(x_1, x_0)$
0	0	0	1	1	1	0
0	1	1	1	1	0	1
1	0	2	1	0	1	1
1	1	3	0	1	1	1

Дешифратори для керування семисегментними індикаторами. Велике поширення в цифровій техніці одержали індикаторні пристрої. Найрозповсюдженішими серед них є семисегментні індикатори, зображення на яких форму-

ється за рахунок активізації сегментів (рис. 4.7). Залежно від активних сегментів, вони здатні відображати цифри від 0 до 9 і деякі символи.

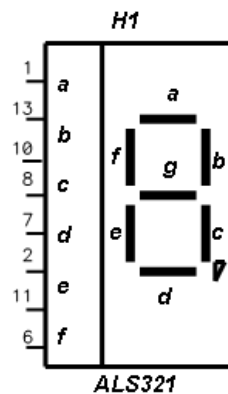


Рис. 4.7. Умовне функціональне позначення семисегментного індикатора

Для керування семисегментними індикаторами розроблені спеціальні дешифратори зі специфічною таблицею істинності (табл. 4.3).

Таблиця 4.3

Таблиця істинності дешифратора для керування семисегментним індикатором

Dec	Входи				Виходи						
	8	4	2	1	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Дешифратори для керування семисегментними індикаторами випускаються як із прямими виходами, так і з інверсними, а також з відкритим колектором. Дешифратори з інверсними виходами будуються на основі інвертованої таблиці істинності. Часто такі дешифратори мають одне найменування і відрізняються тільки буквою наприкінці умовного позначення. Тому, якщо наприкін-

ці позначення вказується буква, то обов'язково необхідно уважно ознайомитися з документацією.

Прикладом такого типу дешифраторів може служити інтегральна схема 555ИД18 (рис. 4.8). Він призначений для керування семисегментними індикаторами з загальним анодом (тобто має інверсні виходи з відкритим колектором).

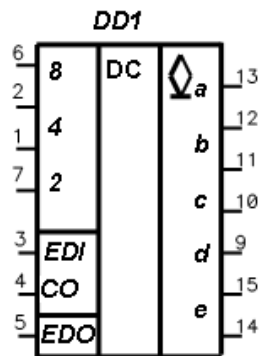


Рис. 4.8. Інтегральна схема індикатора 555ИД18

Крім входів коду дешифрувального двійкового числа і виходів керування сегментами індикатора, він має додаткові входи керування:

EDI – дозвіл дешифрації вхідного коду;

CO – контроль індикатора (активізуються всі сегменти);

EDO – керування станом виходу.

Мікросхема здатна формувати зображення на індикаторі не тільки у формі цифр, але й у формі знаків, відповідно до рис. 4.9.

0	1	2	3	4	5	6	7	8	9	c	3	4	5	6	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Рис. 4.9. Перелік символів, формованих дешифратором 555ИД18

Вона може випускатися в керамічних або пластикових корпусах і забезпечувати безпосереднє керування семисегментними індикаторами із загальним анодом. Максимальний струм на будь-якому виході може складати 24 мА при напрузі живлення навантаження до 15 В.

4.2. Шифратори

Шифратором називається пристрій комбінаційного типу, призначений для перетворення вхідного довільного коду в двійковий.

Умовне графічне позначення шифратора зображене на рис. 4.10. На входних виводах $x_7 \dots x_0$ задається код шифрувального числа. Звичайно вхідний код подається позиційним десятимадвійковим кодом. Особливістю такого способу кодування є активізація тільки єдиного входу, двійковий код якого потрібно одержати на виході. Наприклад, якщо потрібно на виході одержати двійковий код числа **6** (**110**), то активізується (подається логічна одиниця) тільки вхід x_6 . Інші входи залишаються в пасивному стані (в логічному нулі). Може застосовуватися і пріоритетне кодування, коли активізація старшого входу робить байдужим стан молодших входів. Наприклад, якщо старшим активним входом є x_5 , то стан всіх молодших входів ($x_4 \dots x_0$) ігнорується і на виході формується двійковий код **5** (**101**).

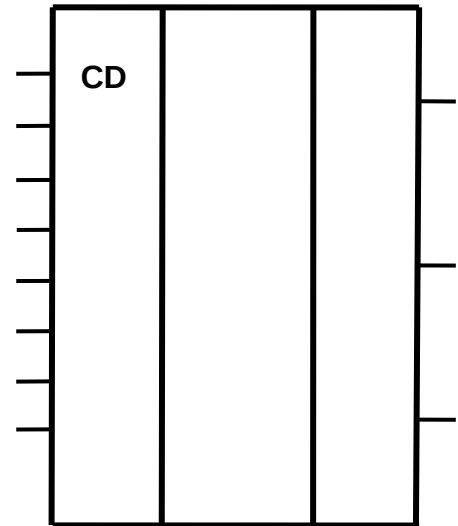


Рис. 4.10. Умовне графічне позначення шифратора

Для шифратора використовується умовне позначення *CD* [1].

Вихідне значення формується відповідно до необхідного кодування чисел: прямий код, обернений код, доповняльний код. Виходи позначаються вагами двійкових розрядів.

Значення таблиці істинності (табл. 4.4) отримані за правилами пріоритетного шифратора.

Умовні позначення “×” використані в таблиці для позначення байдужого стану даного входу в даній ситуації. У таблицю істинності додатково введений стовпчик десятичного значення вихідного коду Y_{dec} для підкреслення відповідності вхідного впливу вихідному двійковому коду.

За заданою таблицею істинності отримують логічні вирази для виходів:

$$Y_4 = x_7 + \overline{x_7} \cdot x_6 + \overline{x_7} \cdot \overline{x_6} \cdot x_5 + \overline{x_7} \cdot \overline{x_6} \cdot \overline{x_5} \cdot x_4;$$

$$Y_2 = x_7 + \overline{x_7} \cdot x_6 + \overline{x_7} \cdot \overline{x_6} \cdot x_5 \cdot x_4 \cdot x_3 + \overline{x_7} \cdot \overline{x_6} \cdot \overline{x_5} \cdot x_4 \cdot x_3 \cdot x_2;$$

$$Y_1 = x_7 + \overline{x_7} \cdot \overline{x_6} \cdot x_5 + \overline{x_7} \cdot \overline{x_6} \cdot \overline{x_5} \cdot x_4 \cdot x_3 + \overline{x_7} \cdot \overline{x_5} \cdot \overline{x_6} \cdot \overline{x_4} \cdot x_3 \cdot x_2 \cdot x_1.$$

За отриманими логічними виразами будується функціональна схема (рис. 4.11).

Таблиця 4.4

Таблиця істинності пріоритетного шифратора

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	$Y_{\text{дес}}$	Y_4	Y_2	Y_1
1	×	×	×	×	×	×	×	7	1	1	1
0	1	×	×	×	×	×	×	6	1	1	0
0	0	1	×	×	×	×	×	5	1	0	1
0	0	0	1	×	×	×	×	4	1	0	0
0	0	0	0	1	×	×	×	3	0	1	1
0	0	0	0	0	1	×	×	2	0	1	0
0	0	0	0	0	0	1	×	1	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0

Випускаються кілька типів шифраторів у вигляді ІС. Найбільш показовою є ІС **КМ555ІВ1**. Дана мікросхема є пріоритетним шифратором. Її умовне графічне позначення зображене на рис. 4.12.

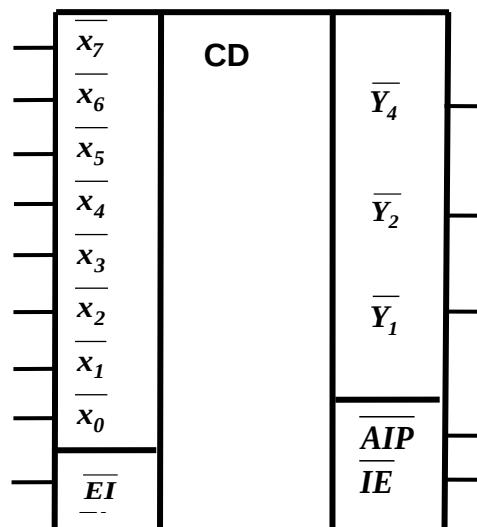
555ІВ1

Рис. 4.12. Шифратор 555ІВ1

Дана ІС має ряд особливостей. Входи вибору вихідного коду $x_7...x_0$ інверсні. Тобто, для вибору необхідного коду на відповідний вхід необхідно подати логічний нуль. При цьому на всіх старших входах повинні бути логічні одиниці. Стан входів, молодших необхідного, є байдужим унаслідок пріоритетного характеру його логіки функціонування.

Вхід дозволу аналізу вхідного коду EI є інверсним. При подачі на нього логічного нуля дозволяється функціонування шифратора, а при подачі на нього логічної одиниці на усіх виходах формується логічна одиниця.

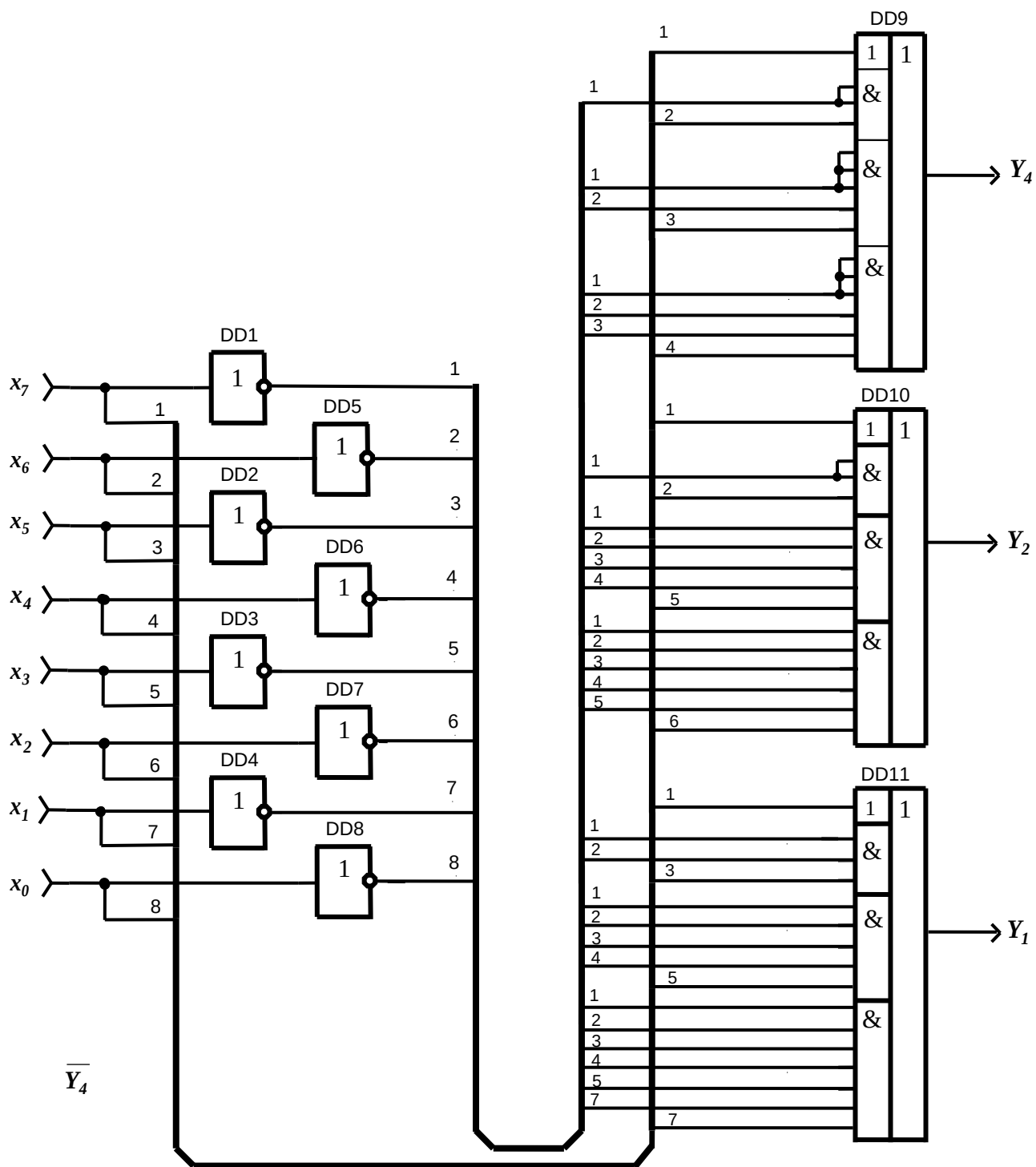


Рис. 4.11. Функціональна схема шифратора

На виходах Y_4, Y_2, Y_1 формується вихідний код, що відповідає активному входу.

Є два додаткові виходи. AIP (*All Input Passive*) – пасивність усіх входів. Цей вихід формує логічний нуль, якщо усі входи $x_7...x_0$ знаходяться в стані логічної одиниці. Вихід IE (*Input Enable*) – активність входів формує логічну оди-

ницю, коли хоча б один вхід $x_7...x_0$ активізується. Додаткові виходи використовуються для каскадування ІС шифраторів.

4.3. Мультиплексори

Мультиплексором називається цифровий пристрій комбінаційного типу, призначений для з'єднання однієї з декількох вхідних інформаційних ліній з єдиною вихідною інформаційною лінією відповідно до заданої адреси [9].

Умовне графічне позначення мультиплексора наведене на рис. 4.13. Воно містить поле інформаційних входів $IN_3...IN_0$, поле входів адреси $A_1...A_0$ і єдиний інформаційний вихід **OUT**. Виконувана функція позначається **MUX** (англ. *multiplexor* – комутатор).

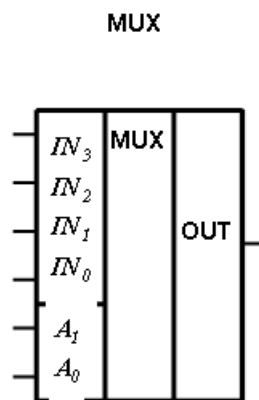


Рис. 4.13. Умовне графічне позначення мультиплексора

Внутрішню структуру мультиплексора можна відобразити, як показано на рис. 4.14. У структурі можна виділити кілька функціональних вузлів. *Комутуючі елементи* – ключі $S_3...S_0$ – для комутації інформаційних сигналів з декількох входів на один вихід. *Дешифратор адреси* – для перетворення коду адреси в керуючі сигнали ключів.

Мультиплексор працює наступним чином. На інформаційних входах присутні двійкові сигнали. На вхід адреси подається код адреси. Дешифратор адреси його дешифрує і замикає відповідний цьому коду комутуючий елемент S , при цьому інформація з даного входу надходить на вихід мультиплексора. У будь-який момент часу з інформаційним виходом може бути з'єднаний тільки єдиний інформаційний вхід. Логічні сигнали на обраному інформаційному вході й інформаційному виході завжди збігаються.

Оскільки кількість входів у мультиплексорів значна, то для них зазвичай не складають повну таблицю істинності, а записують її в скороченій формі. Значення інформаційних входів взагалі не розглядають, а як значення вихідної

функції записуються не логічні значення (0 або 1), а позначення вхідних змінних IN_j .

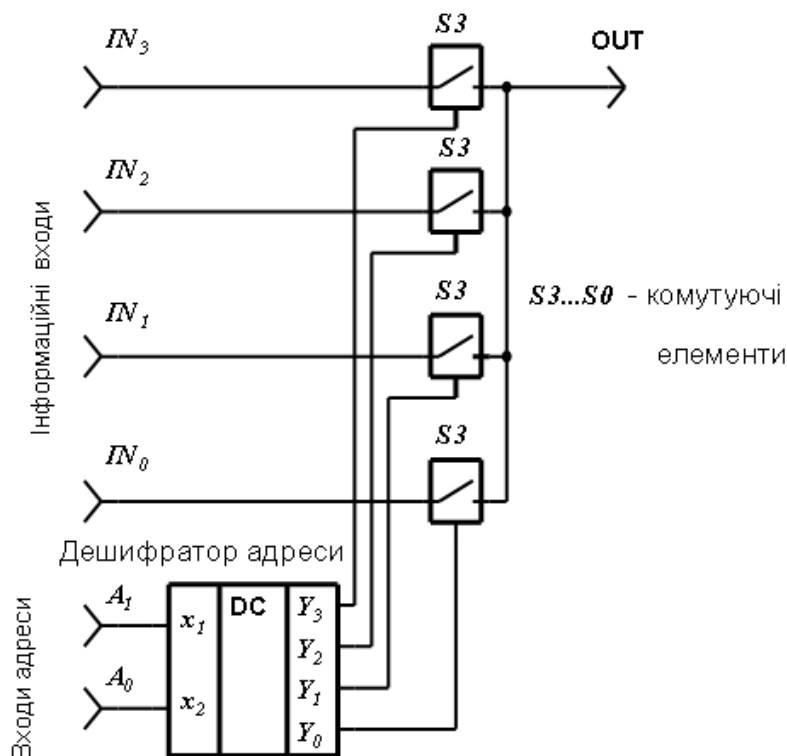


Рис. 4.14. Структурна схема мультиплексора

Мультиплексор $2 \rightarrow 1$

Умовне графічне позначення мультиплексора $2 \rightarrow 1$ показано на рис. 4.15. Він призначений для комутації двох інформаційних входів IN_1 або IN_0 на один інформаційний вихід OUT у відповідності зі значенням адреси, яка подається на адресний вхід A_0 . Якщо на вхід адреси подається логічна одиниця, то на вихід комутується інформаційний вхід IN_1 . Якщо на вхід адреси подається логічний нуль, то на вихід комутується інформаційний вхід IN_0 .

Таблиця істинності для цього пристрою (у скороченій формі) буде мати вигляд:

Таблиця 4.15

Таблиця істинності мультиплексора $2 \rightarrow 1$

A_0	$OUT (Y)$
1	IN_1
0	IN_0

За даною таблицею істинності записується логічний вираз для виходу **OUT (Y)**:

$$Y = A_0 \cdot IN_1 + \overline{A_0} \cdot IN_0.$$

Правила запису логічного виразу залишаються незмінними. Якщо вхідний операнд поданий як логічна одиниця, то він записується в прямій формі. Якщо вхідний операнд поданий логічним нулем – в інверсній формі.

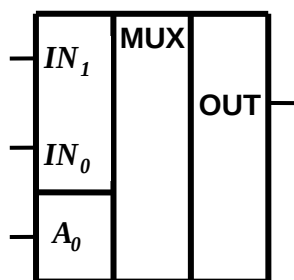


Рис. 4.15. Умовне графічне позначення мультимплексора 2 в 1

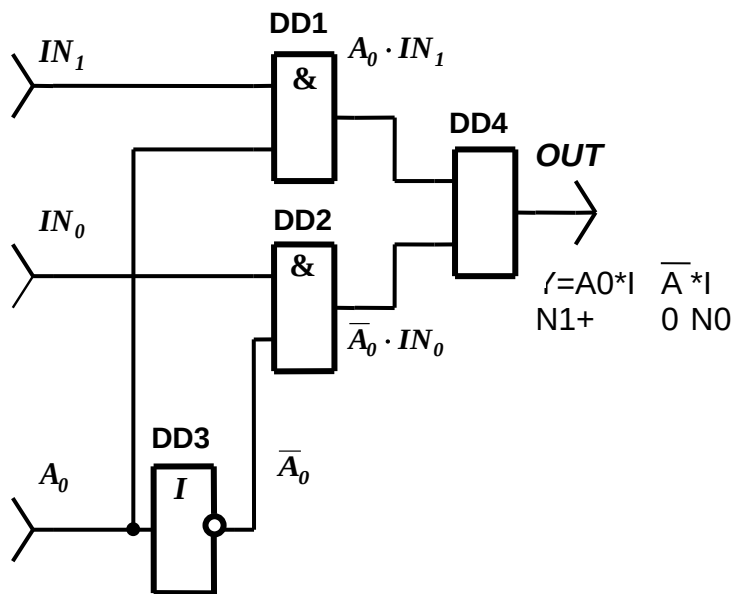


Рис. 4.16. Структурна схема мультимплексора 2 в 1

За отриманим логічним виразом будується функціональна схема даного пристрою (рис. 4.16).

Різновиди мультимплексорів

Мультимплексори з інверсними виходами. Для вирішення окремих задач може знадобитися подача даних в інверсній формі. З цією метою випускаються мультимплексори з інверсними виходами. Умовне графічне позначення наведено на рис. 4.17. Його таблиця істинності – табл. 4.6.

Логічний вираз для його виходу буде мати вигляд

$$Y = \overline{A_1} \cdot \overline{A_0} \cdot IN_0 + \overline{A_1} \cdot A_0 \cdot IN_1 + A_1 \cdot \overline{A_0} \cdot IN_2 + A_1 \cdot A_0 \cdot IN_3.$$

Розгляд логічного виразу показує, що співмножники при вхідних змінних є не що інше, як логічні вирази для виходів дешифратора. Тому, позначивши їх відповідними змінними, можна його спростити:

$$Y = Y_0 \cdot IN_0 + Y_1 \cdot IN_1 + Y_2 \cdot IN_2 + Y_3 \cdot IN_3.$$

Таблиця 4.6

Таблиця істинності мультиплексора з інверсними входами

A_1	A_0	$OUT (Y)$
0	0	$\overline{IN_0}$
0	1	$\overline{IN_1}$
1	0	$\overline{IN_2}$
1	1	$\overline{IN_3}$

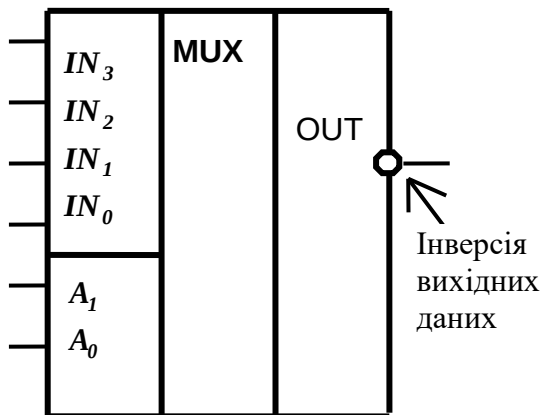


Рис. 4.17. Умовне графічне позначення мультиплексора 4 в 1 з інверсією

Дане правило має місце не тільки для мультиплексорів з інверсними виходами, але і для будь-яких інших типів.

За отриманим логічним виразом будується структурна схема (рис. 4.18).

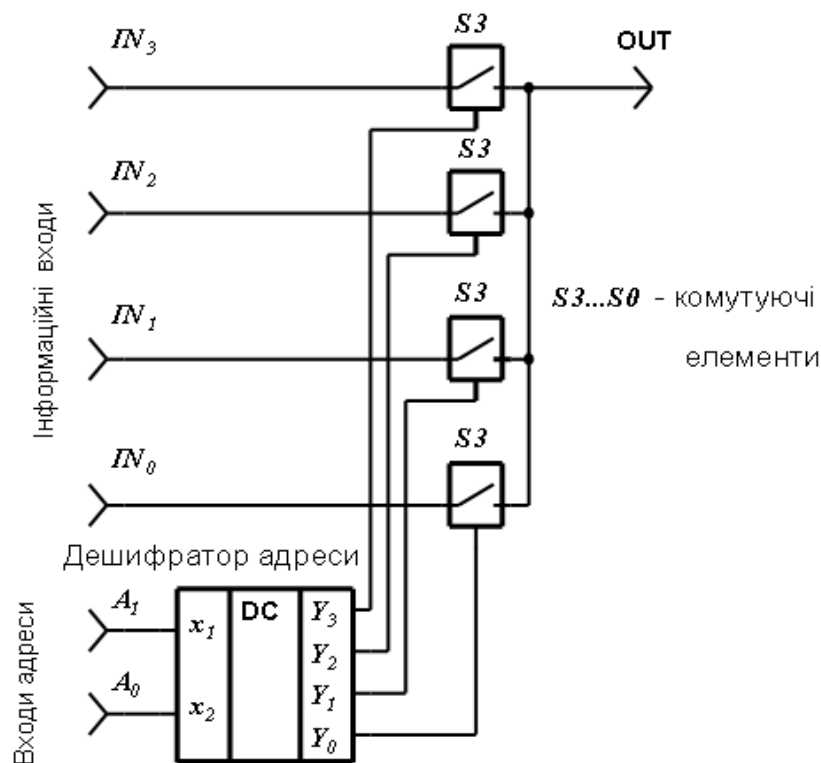


Рис. 4.18. Структурна схема мультиплексора з інверсією вхідних даних

Багатоканальні мультиплексори. У зв'язку з тим, що цифрові пристрої часто бувають багаторозрядними, виникає необхідність комутувати потоки багаторозрядних даних. Для вирішення таких задач були розроблені мікросхеми

багатоканальних (багаторозрядних) мультиплексорів. Одна інтегральна мікросхема може містити в собі декілька мультиплексорів.

Входи адреси і керування в цих мультиплексорах сполучені, інформаційні й розрядні входи/виходи різні.

Прикладом такого мультиплексора є мікросхема **555КП11** (рис. 4.19). Вона становить чотирьохканальний мультиплексор $2 \rightarrow 1$. Кожний з чотирьох каналів (A, B, C, D) дозволяє комутувати дані від двох джерел до одного споживача. Вихідні каскади мультиплексора побудовані за схемою з трьома станами. Виходи можуть переводитися в **Z**-стан (вимкнено) при подачі одиниці на вхід дозволу **EO** (*enable output*). У робочому режимі на цьому вході повинний бути логічний нуль. За даною функцією вирішується задача нарощування розрядності мультиплексорів.

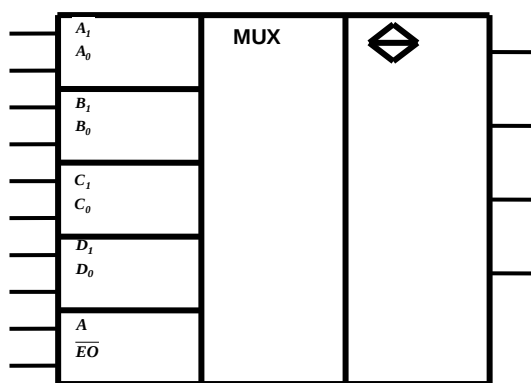


Рис. 4.19. Умовне графічне позначення мультиплексора 555КП11

Нарощування розрядності мультиплексорів забезпечується їх пірамідальним з'єднанням (рис. 4.20).

Схема складається з двох етапів: у перший входять мультиплексори **DD1...DD4**, а в другий – мультиплексор **DD5**. На мультиплексори першого ступеня подаються молодші адреси (A_1, A_0), а на **DD5** – старші (A_3, A_2).

Нарощування розрядності мультиплексорів

Комутація з'єднання відбувається за двома ступенями: мультиплексорами першого ступеня вибираються чотири входи, наприклад X_{15}, X_{11}, X_7, X_3 ($A_1 = 1, A_0 = 1$), а мультиплексором другого ступеня вибирається один з першої четвірки, наприклад $Y_{1.1}$ ($A_3 = 0, A_2 = 1$). Збільшуючи кількість розрядів адреси можна використовувати цей принцип побудови і для більшої кількості входів. Застосування такого способу з'єднання не вимагає використання ніяких додаткових елементів. Перевагою такої схеми є її простота. Однак, при збільшенні кількості

ступенів мультиплексування знижується швидкодія за рахунок введення додаткових мікросхем (наприклад, **DD5**), що можна віднести до її недоліків.

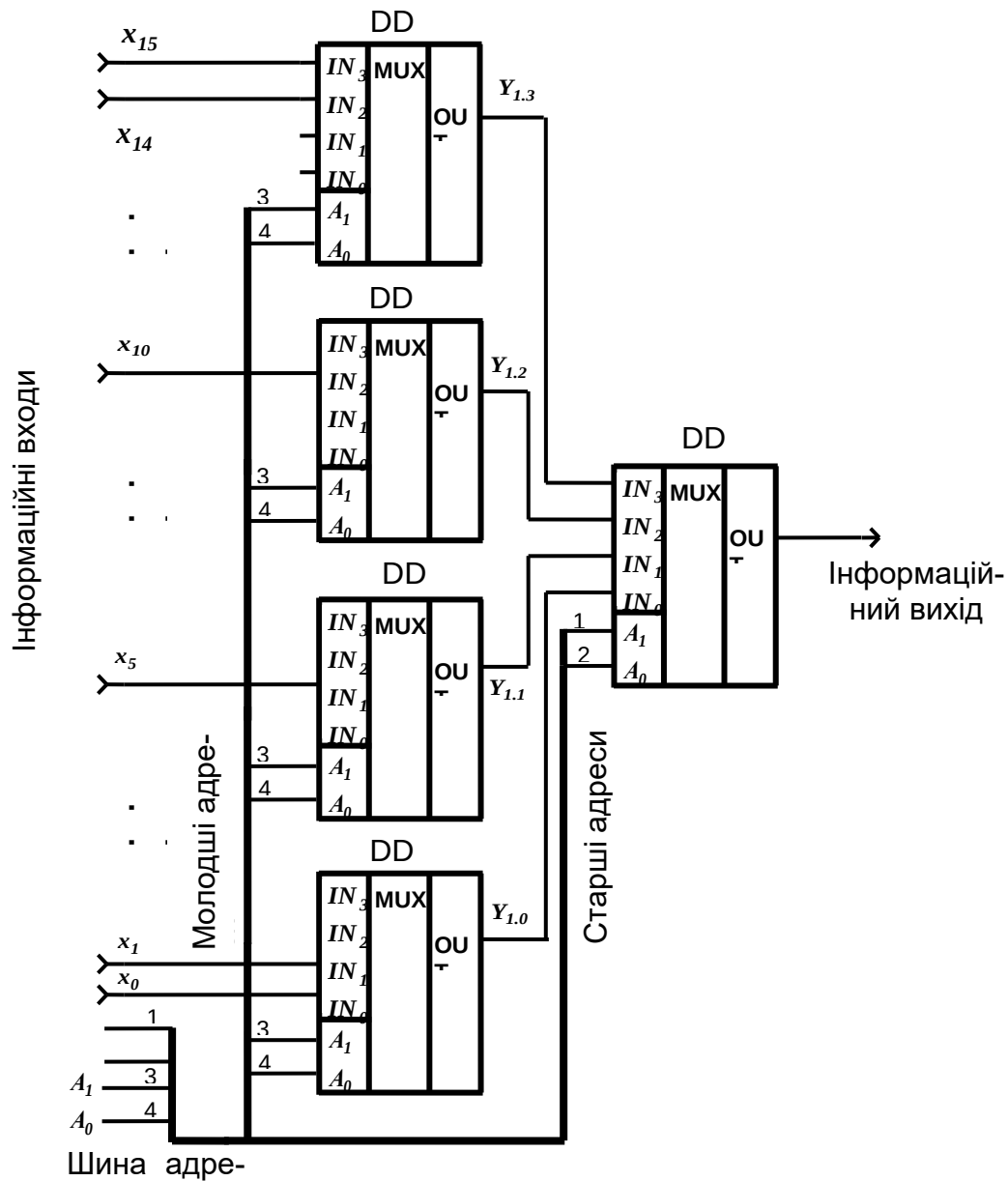


Рис. 4.20. Пірамідальне з'єднання мультиплексора

Частково ці недоліки усуває паралельна схема з'єднання мультиплексорів (рис. 4.21). Дане з'єднання вимагає використання додаткових логічних елементів **DD3** і **DD4**. Старший розряд адреси A_3 вибирає мультиплексор, який повинен бути активним. Якщо $A_3 = 0$, то дозволяється функціонування мультиплексора **DD2** ($EI_{DD2} = 0$ (EI – enable input)), а забороняється **DD1** ($EI_{DD1} = 1$ через інверсію на **DD4**).

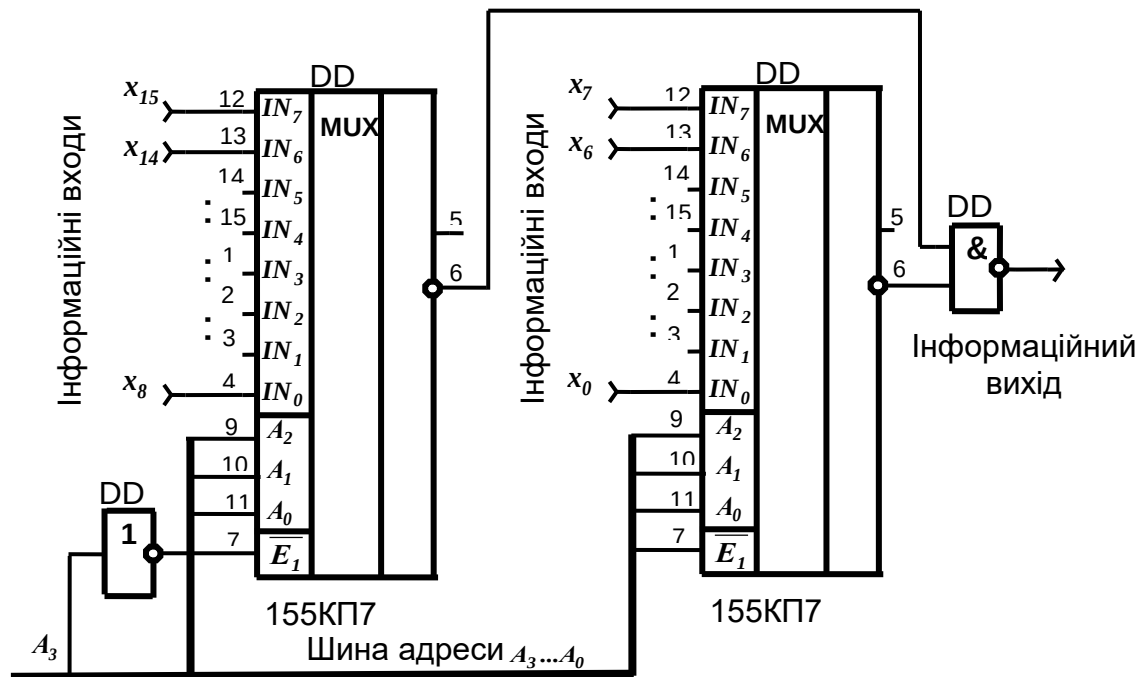


Рис. 4.21. Паралельне з'єднання мультиплексора

Швидкодія такої схеми виявляється вищою, ніж при пірамідальному з'єднанні мультиплексорів. Однак, схема виходить складнішою. Якщо мультиплексори мають виходи з трьома станами, то вихідний логічний елемент не потрібний і схема спрощується та стає більш швидкодіючою (рис. 4.22).

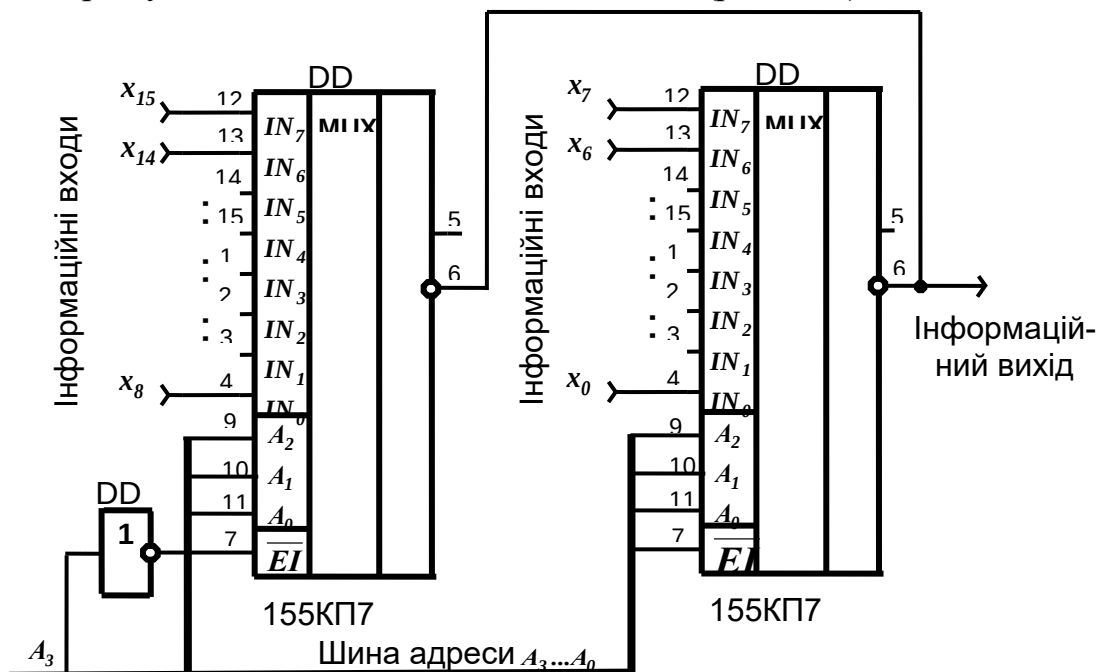


Рис. 4.22. Паралельне з'єднання мультиплексорів із виходами з Z-станами

Інтегральні мікросхеми мультиплексорів випускаються в різних серіях як ТТЛ, так і МОН і мають функціональне позначення "КП".

4.4. Демультимплексори

Демультимплексором називається цифровий пристрій комбінаційного типу, призначений для комутації єдиної вхідної інформаційної лінії з однією з декількох вихідних інформаційних ліній відповідно до заданої адреси.

Умовне графічне позначення чотиривихідного демультимплексора (рис. 4.23) містить у собі три поля. Поле входів складається з інформаційного входу X та ліній адреси A_1, A_0 . Функціональне позначення демультимплексора задається абревіатурою **DMX** (англ. *demultiplexer* – пристрій роз'єднання каналів). У поле виходів відображаються інформаційні виходи $Y_3...Y_0$. Функціонування демультимплексора полягає в комутації інформаційного входу на один з виходів відповідно до заданої адреси. Всі інші виходи будуть знаходитися в статичному стані. Якщо виходи демультимплексора прямі, то статичний стан вимкнених виходів дорівнює логічному 0, якщо виходи інверсні, то логічний 1.

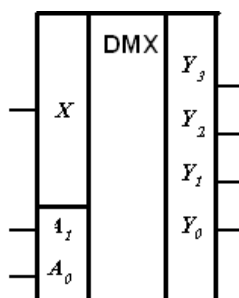


Рис. 4.23. Умовне графічне позначення демультимплексора

Структурно (рис. 4.24) демультимплексор включає у свій склад дешифратор адреси (для перетворення коду адреси в сигнали керування комутуючими елементами) і самі комутуючі елементи (комутуючими елементами є логічні елементи), які здійснюють передачу двійкових даних із входу на вихід.

Демультимплексор $1 \rightarrow 2$

Умовне графічне позначення демультимплексора $1 \rightarrow 2$ наведене на рис. 4.25. Його функціонування, як і функціонування мультиплексора, описується таблицею істинності в скороченій формі (табл. 4.7).

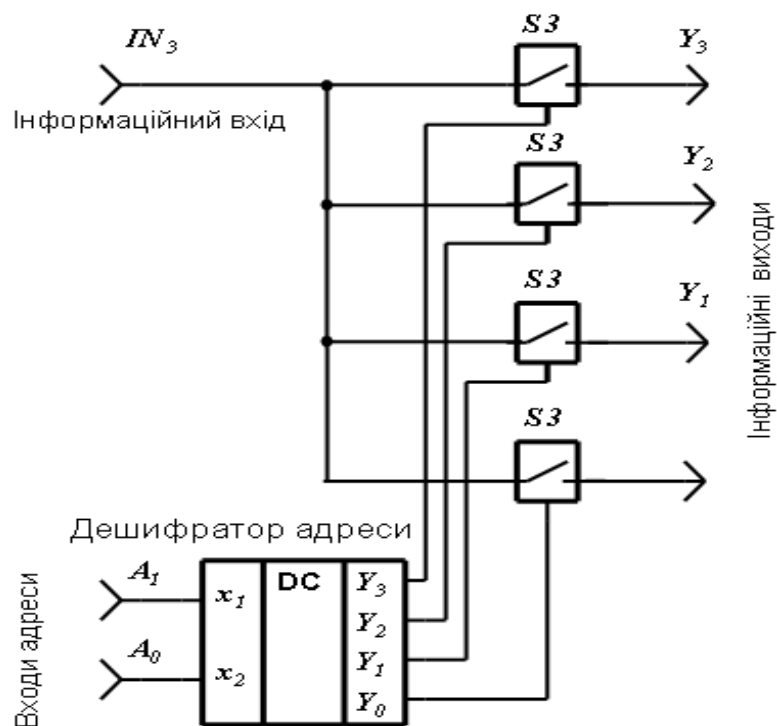


Рис. 4.24. Структурна схема демультиплексора

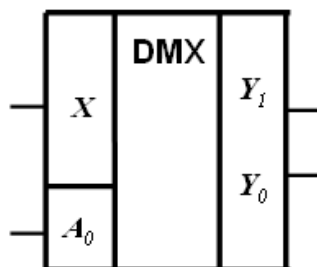


Рис. 4.25. Умовне графічне позначення демультиплексора

Таблиця 4.7

Таблиця істинності демультиплексора $1 \rightarrow 2$

A_1	Y_1	Y_0
0	0	X
1	X	0

Для кожного виходу знаходиться свій логічний вираз:

$$Y_1 = A_0 \cdot X ;$$

$$Y_0 = \overline{A_0} \cdot X .$$

За отриманими логічними виразами будується структурна схема демультиплексора (рис. 4.26).

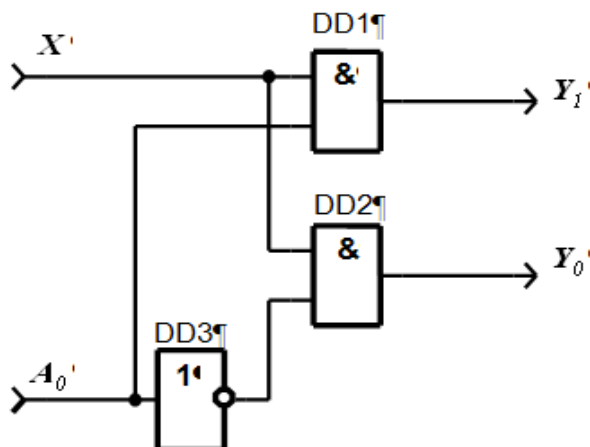


Рис. 4.26. Структурна схема демультиплексора 1 в 2

Розгляд структурної схеми показує, що в ролі одновхідного дешифратора використовується логічний елемент ЧИ, а в ролі комутуючих елементів – логічні елементи І.

Різновиди демультиплексорів

Демультиплексори з інверсними виходами. Для вирішення окремих задач буває необхідним надання даних в інверсній формі. З цією метою випускаються демультиплексори з інверсними виходами. Одержати структурну схему демультиплексора з інверсними виходами легко. Для цього необхідно тільки проінвертувати вихідні значення інформаційних виходів (рис. 4.27).

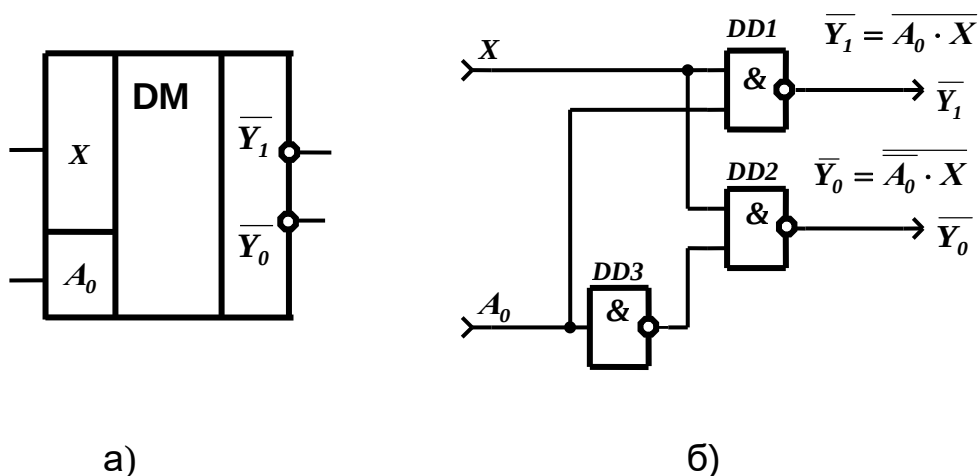


Рис. 4.27. Умовне графічне позначення (а) і структурна схема (б) демультиплексора $1 \rightarrow 2$ з інверсними виходами

Демультимплексори із входами дозволу. Для розширення функціональних можливостей інтегральних мікросхем демультимплексорів їх випускають з додатковими входами: ЕІ – вхід дозволу дії інформаційних входів та ЕО – вхід керування станом вихідних каскадів (рис. 4.28).

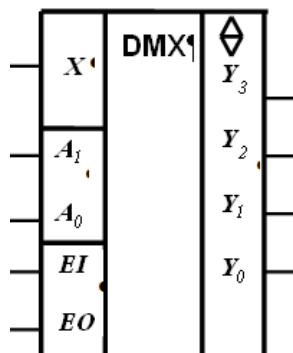


Рис. 4.28. Демультимплексатор із входами дозволу

Вхід ЕІ в активному стані (для розглянутого прикладу $EI = 1$) дозволяє зміну стану логічних елементів, пов'язаних із входом X . В пасивному стані (у даному випадку $EI = 0$) фіксує їх стан і унеможливорює проходження інформаційного потоку через демультимплексор.

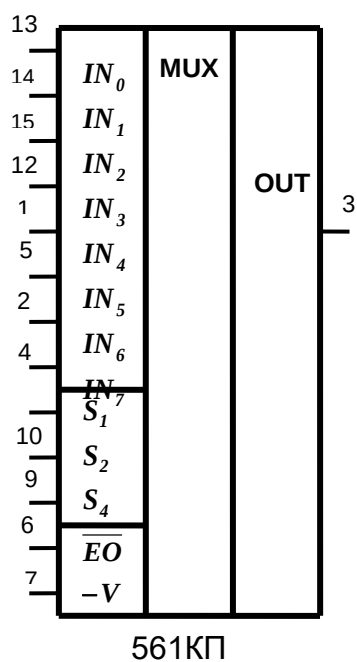
Ця можливість використовується для запобігання помилкової зміни стану виходів демультимплексора на час дешифрації нової адреси.

Використання вихідних каскадів із трьома станами в демультимплексорах призводить до появи додаткового входу ЕО – дозволу виходу. Цей вхід дозволяє перевести вихідні каскади в **Z**-стан (вимкнений). Використання такої можливості дозволяє з'єднувати виходи декількох демультимплексорів разом, що спрощує побудову комутаторів інформаційних потоків.

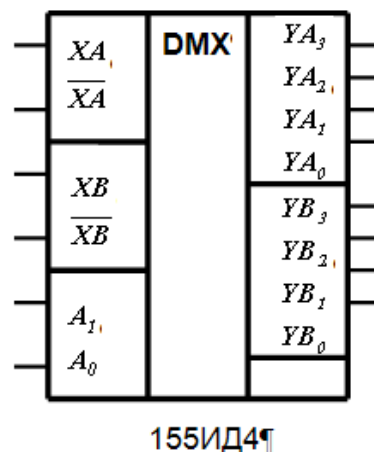
Сумісні демультимплексори. Одна інтегральна мікросхема може містити в собі декілька демультимплексорів. Входи адреси і керування вихідними каскадами в цих демультимплексорах сумісні, інформаційні входи/виходи різні. Прикладом такої мікросхеми може служити дешифратор-демультиплексор **155ИД4** (рис. 4.29). Він становить двоканальний демультимплексор $1 \rightarrow 4$. Вибір інформаційних виходів здійснюється адресними входами A_1, A_0 . Інформація може подаватися на чотири інформаційних входи ($XA, \overline{XA}, XB, \overline{XB}$), по два (прямий та інверсний) на кожен канал.

Мультимплексори/демультиплексори. **КМОН**-технологія дозволяє створювати мутуючі канали передачі даних з байдужим напрямком передачі. Такі пристрої можна використовувати в довільному функціональному призначенні: як му-

льтиплексор, або як демультимплексор. Напрямок передачі даних для таких мікросхем не має значення. Двонаправленість каналу передачі даних дозволяє так само комутувати аналогові сигнали з рівнями, які не перевищують використовуваний діапазон напруг живлення.



561КП2
Рис. 4.30. IC 561КП2



155ИД4
Рис. 4.29. Дешифратор-демультиплексор 155ИД4

Прикладом такої мікросхеми може служити мультиплексор/демультиплексор **561КП2** (рис. 4.30). Вона становить комутатор $8 \rightarrow 1$ або $1 \rightarrow 8$, залежно від обраного напрямку передачі даних. Живлення мікросхеми може бути однополярним або двополярним.

Однополярна напруга живлення може бути обрана, як для будь-яких КМОН ІС, у межах $5...15\text{ В}$. У цьому випадку вивід 7 ($-V$) замикається на корпус. Двополярне живлення використовується для забезпечення комутації двополярних сигналів. Джерело негативної напруги підключається до виводу 7 ($-V$). Для усунення спотворень амплітуда сигналів, що комутуються, у будь-якому випадку не повинна перевищувати $0,75$ від напруги живлення. Вибір напрямку комутації задається за допомогою виводів адреси $-S_4, S_2, S_1$.

Замкнутим каналом здійснюється передача даних, а інші канали залишаються розімкнутими (**Z**-стан). Опір замкнутого каналу знаходиться в межах від $0,13\text{ кОм}$ (для напруги живлення 15 В) до $2,5\text{ кОм}$ (для напруги живлення 5 В). Передбачена можливість примусового розмикання каналів у разі потреби. Для цього використовується вхід **EO** (інверсний). Якщо на нього подати логічну одиницю, то всі канали розімкнуться. Якщо на нього подати логічний нуль, то замкнеться єдиний канал, заданий по входах S_4, S_2, S_1 .

Крім готових мультиплексорів/демультиплексорів – комутаторів сигналів, випускається ряд однорозрядних комутаторів, які мають єдиний канал, який комутується, і вхід керування цим каналом. Прикладом такої мікросхеми може служити мікросхема **561КТЗ**.

Нарощування розрядності мультиплексорів

Використовується два ступеня демультиплексування (рис. 4.31). Перший ступінь демультиплексування забезпечується демультиплексором **DD1** за адресою, яка задається старшими розрядами адреси A_3, A_2 . Він направляє потік даних на один з демультиплексорів другого ступеня **DD2...DD5**. Другий ступінь демультиплексування використовує молодші розряди адреси A_1, A_0 і направляє потік даних на заданий вихід. На рис. 4.31 показаний приклад проходження потоку даних через схему. За заданою адресою ($A = 0101_2 (5_{10})$) демультиплексор першого ступеня **DD1** комутує потік даних у відповідності зі значеннями старших розрядів адреси ($A_3 A_2 = 01_2 (1_{10})$) на свій перший вихід, який підключений до інформаційного входу **DD4**. Демультиплексор другого ступеня використовує для комутації молодші розряди адреси ($A_1 A_0 = 01_2 (1_{10})$) і направляє потік даних на свій перший вихід. Цим виходом у структурі комутатора є Y_5 і він відповідає повній адресі, заданій демультиплексору.

Дане з'єднання демультиплексорів не використовує ніяких додаткових елементів і є дуже простим. Однак, через послідовне з'єднання демультиплексорів зростає час затримки поширення сигналу.

Використання дешифраторів як демультиплексорів

Аналізуючи таблицю істинності демультиплексора, наприклад $1 \rightarrow 4$, і логічні вирази для виходів демультиплексора (табл. 4.8), можна помітити, що логічні вирази містять у собі формули для виходів дешифратора ($A_1 A_0$) і їх добуток на значення інформаційного входу X . Якщо перейменувати інформаційний вхід X у вхід дозволу EO , то отримаємо логічні вирази для дешифратора із входом дозволу виходів:

$$Y_0 = \overline{A_1} \cdot \overline{A_0} \cdot EO;$$

$$Y_1 = \overline{A_1} \cdot A_0 \cdot EO;$$

$$Y_2 = A_1 \cdot \overline{A_0} \cdot EO;$$

$$Y_3 = A_1 \cdot A_0 \cdot EO$$

Таблиця істинності демультиплексора $1 \rightarrow 4$

A_1	A_0	Y_0	Y_1	Y_2	Y_3	Логічні вирази
0	0	X	0	0	0	$Y_0 = \overline{A_1} \cdot \overline{A_0} \cdot X$
0	1	0	X	0	0	$Y_0 = \overline{A_1} \cdot A_0 \cdot X$
1	0	0	0	X	0	$Y_0 = A_1 \cdot \overline{A_0} \cdot X$
1	1	0	0	0	X	$Y_0 = A_1 \cdot A_0 \cdot X$

Ця властивість широко використовується при розробці ІС дешифраторів. Прикладами використання дешифраторів як демультиплексорів можуть служити ІС **155ИД4** (рис. 4.29), або **155ИД3** (рис. 4.32).

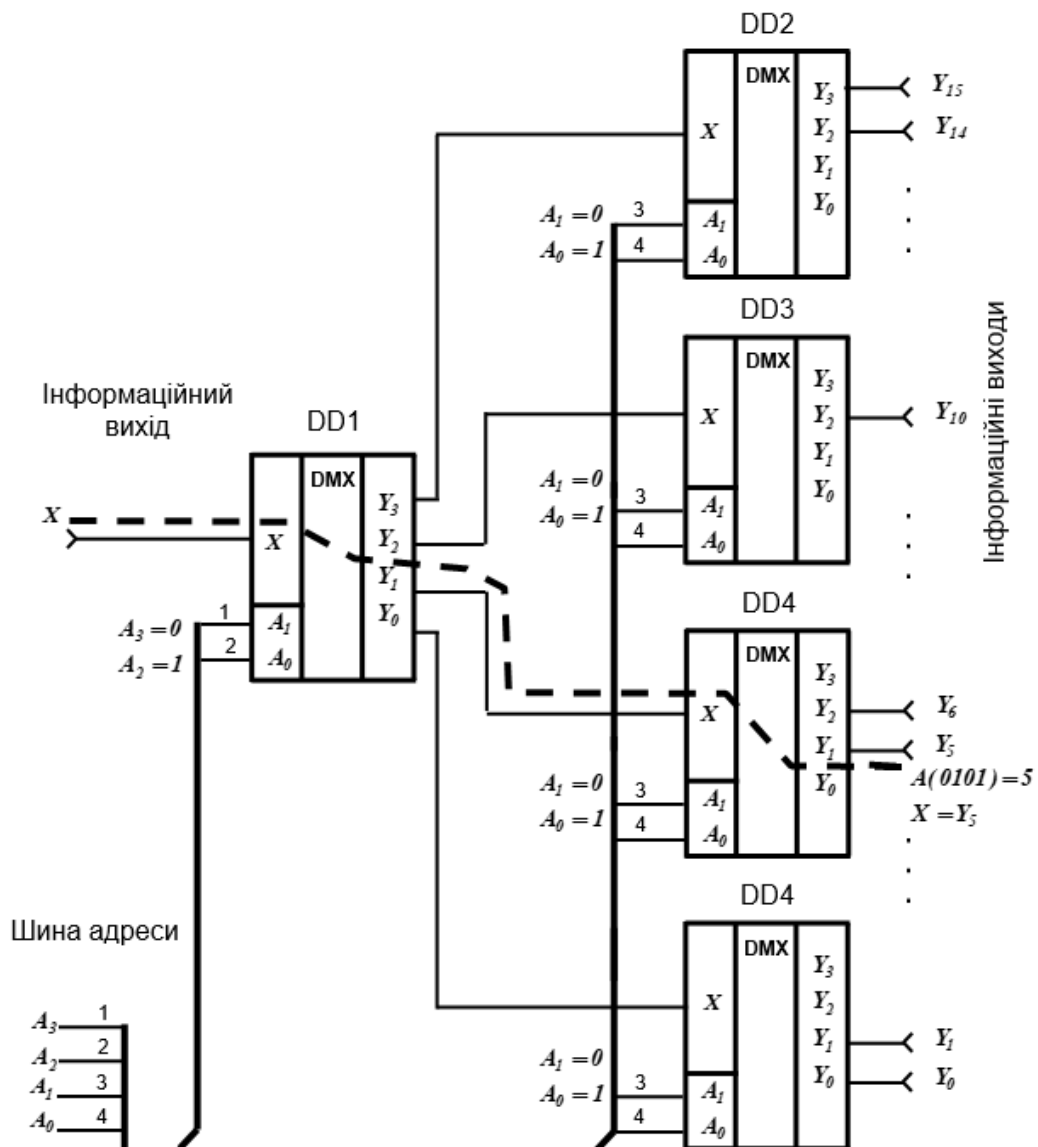
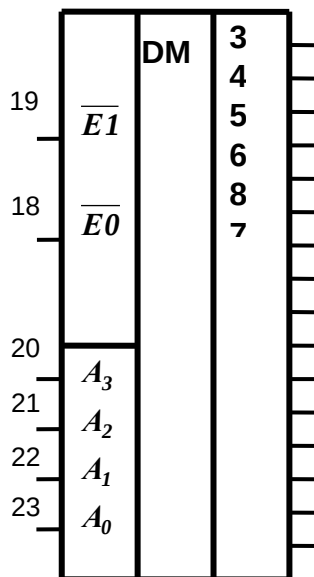


Рис. 4.31. Пірамідальне з'єднання мультиплексора

Дешифратор **155ИДЗ** може використовуватися як демультіплексор таким чином. На вході адреси $A_3...A_0$ задається адреса комутації (активний вихід Y_i). Стани виходу будуть визначатися виразом $\overline{Y}_i = \overline{E}_1 + \overline{E}_0$, або таблицею істинності (табл. 4.9).



Таблиця 4.9

Стани виходу 155ИДЗ

$\overline{E}_1$	$\overline{E}_0$	$\overline{Y}_i$
0	0	0
0	1	1
1	0	1
1	1	0

Рис. 4.32. IC 155ИДЗ

4.5. Суматори

Суматором називається пристрій комбінаційного типу, призначений для знаходження суми двох двійкових чисел.

Виконання операції додавання має передбачати знаходження арифметичної суми чисел відповідно до законів математики. Основними умовами процесу додавання є наявність двох N -розрядних операндів на вході і, як наслідок, отриманий результат додавання на виході. Варто помітити, що розрядність суми може виявитися на один розряд більшою, ніж розрядність вхідних операндів. Цей додатковий розряд називається переносом (в англійській мові для позначення переносу застосовується термін “*carry*”, тому на схемах цей вихід позначається латинською буквою *C* або *CR*; для відображення виконуваної функції (введення/виведення) до умовного позначення може додаватися суфікс *in* – для введення і *out* – для виведення).

Півсуматори

Півсуматором називається цифровий пристрій комбінаційного типу, призначений для знаходження суми двох однорозрядних двійкових чисел.

Умовне графічне позначення півсуматора (рис. 4.33) включає: **A0** – вхід нульового розряду A_0 числа A ; **B0** – вхід нульового розряду B_0 числа B ; **HSM** – умовне графічне позначення виконуваної операції; **SM** – вихід суми; **CR** – вихід переносу (може позначатися як **C** або **Crout**). Домовимось, що номери виводів будуть позначатись великими цифрами, а номери сигналів, які обробляються в будь-якому пристрої, – індексами.



Рис. 4.33. Умовне графічне позначення півсуматора

Функціонування півсуматора описується таблицею істинності (табл. 4.10).

Таблиця 4.10

Таблиця істинності півсуматора

Входи		Значення суми		Виходи	
A_0	B_0	<i>Decimal</i>	<i>Binary</i>	<i>Crout</i>	<i>SM</i>
0	0	0	00	0	0
0	1	1	01	0	1
1	0	1	01	1	0
1	1	2	10	1	1

У таблицю додатково введені стовпці “Значення суми” у двох системах числення: десятковій (*Decimal*) і двійковій (*Binary*). Після розгляду результату підсумовування в двійковій системі числення однозначно впливають значення виходів **Crout** та **SM**.

Логічні вирази, що описують роботу півсуматора, мають вигляд:

$$SM = \overline{A_0} \cdot B_0 + A_0 \cdot \overline{B_0}, \quad Crout = A_0 \cdot B_0.$$

Функціональна схема півсуматора зображена на рис. 4.34.

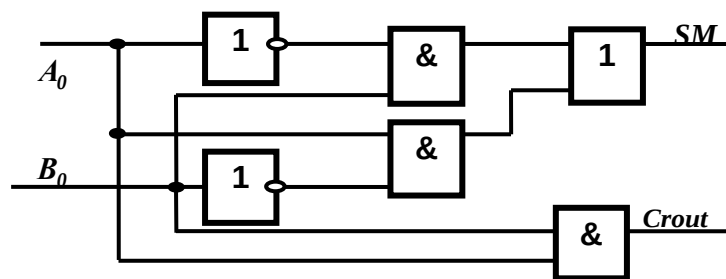


Рис. 4.34. Функціональна схема півсуматора

Для функції типу $SM = \overline{A_0} \cdot B_0 + A_0 \cdot \overline{B_0}$, якій відповідає таблиця істинності (табл. 4.11), існує спеціальна назва “ВИКЛЮЧАЛЬНЕ ЧІ” або операція нерівнозначності. Також існує логічний елемент, здатний її реалізовувати (рис. 4.35). Його використання дозволяє значно спростити схему півсуматора [1].

Таблиця 4.11

Таблиця істинності елемента “ВИКЛЮЧАЛЬНЕ ЧІ”

x_1	x_0	$Y_2(x_1, x_0)$
0	0	0
0	1	1
1	0	1
1	1	0

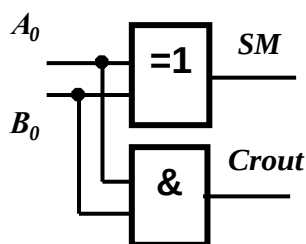


Рис. 4.35. Півсуматор на основі елемента нерівнозначності

На рис. 4.35 елемент з позначкою “= 1” – це приклад умовного графічного позначення логічного елемента “ВИКЛЮЧАЛЬНЕ ЧІ”.

Однорозрядний суматор

Однорозрядним суматором називається пристрій комбінаційного типу, призначений для знаходження суми одного розряду багаторозрядного двійкового числа. Відмінністю однорозрядного суматора від півсуматора є наявність у

ньому додаткового входу переносу **Crin** (може позначатися **C**, **Crin** або **CR**). Цей вхід з'єднується з виходом переносу суматора попереднього розряду.

Умовне графічне позначення однорозрядного суматора (рис. 4.36) включає: **CR (Crin)** – вхід значення переносу із суматора попереднього розряду; **A** – вхід розряду числа А, **B** – вхід розряду числа В; **SM** – позначення виконуваної функції; виходи: **SM** – вихід значення суми розряду чисел А і В; **CR (Crout)** – вихід переносу суми розряду чисел А і В.

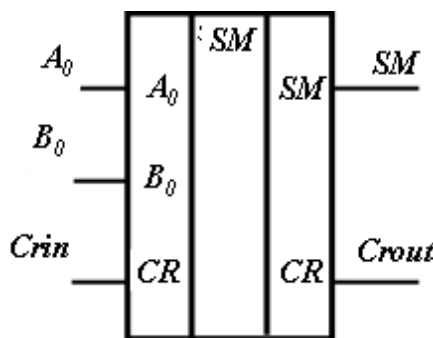


Рис. 4.36. Умовне графічне позначення однорозрядного суматора

Як і в десятковій системі числення, операція додавання двох *n*-розрядних чисел $A\{a_{n-1}...a_1a_0\}$ та $B\{b_{n-1}...b_1b_0\}$ виконується порозрядно, починаючи з молодшого. Розрядні числа a_i та b_i додаються "за модулем 2" з урахуванням переносу з попереднього (молодшого) розряду CR_{i-1} відповідно до табл. 4.12. Сума $SM_i = 1$, якщо кількість доданків зі значенням 1 непарна, а перенос у наступний розряд $CR_i = 1$, якщо кількість таких доданків не менше двох. За цих умов додавання числа $A_{10} = 7$ до числа $B_{10} = 19$ буде виглядати наступним чином:

$$\begin{array}{r}
 A_2 = 0. \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 = A_{10} = 7 \\
 + \\
 B_2 = 0. \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 = B_{10} = 19 \\
 \hline
 \phantom{= A_{10}} \phantom{= B_{10}} \\
 \phantom{= A_{10}} \phantom{= B_{10}} \\
 \phantom{= A_{10}} \phantom{= B_{10}} \\
 S_2 = 0. \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 = S_{10} = 26
 \end{array}$$

Відповідно до таблиці істинності отримані логічні вирази для виходів однорозрядного суматора:

$$\begin{aligned}
 SM &= \overline{Crin} \cdot \overline{A} \cdot B + \overline{Crin} \cdot A \cdot \overline{B} + Crin \cdot \overline{A} \cdot \overline{B} + Crin \cdot A \cdot B; \\
 CR &= \overline{Crin} \cdot A \cdot B + Crin \cdot \overline{A} \cdot B + Crin \cdot A \cdot \overline{B} + Crin \cdot A \cdot B
 \end{aligned}$$

Таблиця 4.12

Таблиця істинності однорозрядного суматора

Входи			Значення суми		Виходи	
<i>Crin</i>	<i>A</i>	<i>B</i>	<i>Decimal</i>	<i>Binary</i>	<i>Crout</i>	<i>SM</i>
0	0	0	0	00	0	0
0	0	1	1	01	0	1
0	1	0	1	01	0	1
0	1	1	2	10	1	0
1	0	0	1	01	0	1
1	0	1	2	10	1	0
1	1	0	2	10	1	0
1	1	1	3	11	1	1

Можна помітити, що таблиця істинності (табл. 4.12) і відповідні їй логічні вирази складаються з двох частин: таблиці істинності півсуматора (перші чотири рядки) і виконання операції підсумовування отриманого результату зі значенням переносу *Crin*. Логічні вирази в цьому випадку будуть мати вигляд:

$$SM = \overline{SM'} \cdot Crin + SM' \cdot \overline{Crin};$$

$$Crout = CR' \cdot CR'';$$

а схема суматора, яка реалізується згідно з цими формулами, наведена на рис. 4.37.

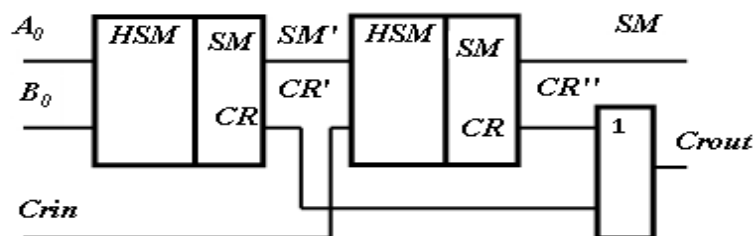


Рис. 4.37. Однорозрядний суматор на двох півсуматорах

Нарощування розрядності в суматорах

Однорозрядні цифрові системи, як правило, не знаходять практичного застосування у зв'язку з низькою забезпеченою точністю. Звичайно розрядність цифрових систем кратна чотирьом або восьми. У цьому випадку виникає необхідність виконувати операцію підсумовування над багаторозрядними числами. Використовувати з цією метою однорозрядні суматори можливо шляхом їх об'єднання. Таке об'єднання називається нарощуванням розрядності.

Чотирьохрозрядний суматор з послідовним переносом

Методика нарощування розрядності полягає в послідовному з'єднанні однорозрядних суматорів з послідовною передачею значення переносу з поперед-

нього розряду до наступного (рис. 4.38). Тому така схема з'єднання одержала назву схеми з послідовним переносом. Використання можливостей мікроелектроніки дозволяє однорозрядні суматори об'єднати в один закінчений функціональний пристрій – чотирьохрозрядний суматор (рис. 4.39).

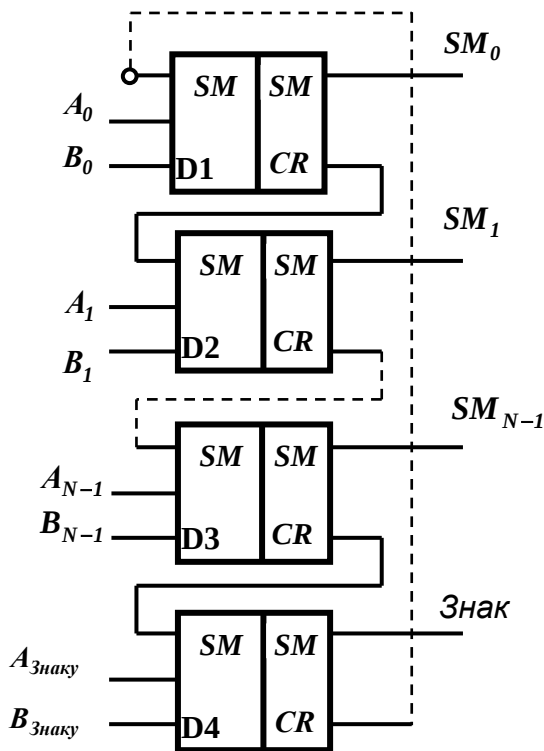


Рис. 4.38. Багаторозрядний суматор з послідовним перенесенням

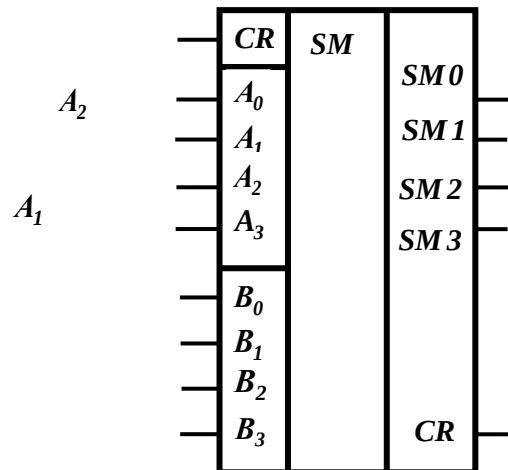


Рис. 4.39. Умовне графічне позначення чотирьохрозрядного суматора

Такий підхід не має явних обмежень щодо кількості розрядів і є дуже простим. Однак, з погляду швидкодії, така схема має істотний недолік. Так, прийнявши за час формування значення переносу одного розряду величину t_{CR} одержимо $T_{CR\ SM} = N \cdot t_{CR}$. З формули випливає, що час виконання операції підсумовування зростає пропорційно кількості розрядів N і часу формування значення переносу одного розряду t_{CR} . Це призводить до швидкого наростання затримки і робить таку схему непридатною в багаторозрядних пристроях.

Суматор з паралельним переносом

Для підвищення швидкодії багаторозрядних суматорів необхідно формувати значення переносу одночасно з обчисленням значення суми. Для цього використовуються спеціальні комбінаційні пристрої – вузли прискореного переносу – **CRU**.

Для нульового (молодшого) розряду значення переносу формується самостійно і не потрібно ніякого прискорення $CR_0 = CR_0'$.

Для першого розряду $CR_1 = CR_1' + CR_0' \cdot SM_1'$.

Для другого розряду $CR_2 = CR_2' + CR_1' \cdot SM_2' + CR_0' \cdot SM_1' \cdot SM_2'$.

Для третього розряду

$$Crout = CR_3 = CR_3' + CR_2' \cdot SM_3' + CR_1' \cdot SM_2' \cdot SM_1' + CR_0' \cdot SM_3' \cdot SM_2' \cdot SM_1'$$

Реалізація вузлів прискореного переносу наведена на рис. 4.40.

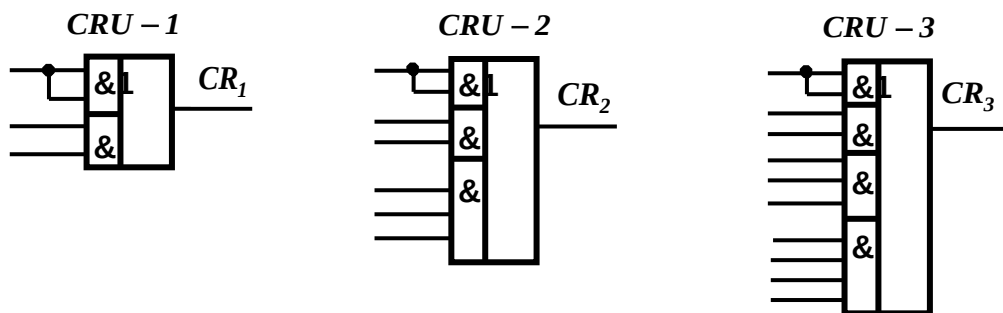


Рис. 4.40. Варіанти побудови вузлів прискореного переносу

Застосування вузлів прискореного переносу змінить побудову чотирьохрозрядного суматора на півсуматорах. Така схема виявляється прийнятною за швидкістю і досить складною за конструктивним виконанням та практично не застосовується при реалізації на ІС малого ступеня інтеграції. Компромісне рішення може бути знайдене при обмеженні вимоги щодо швидкодії. У цьому випадку для чотирьох розрядів перенесення формується послідовно, а зовнішнє перенесення паралельно. При проектуванні суматорів у мікроелектронному виконанні є можливість реалізовувати практично будь-які схеми, не звертаючи уваги на кількість використовуваних елементів. Тому, з метою досягнення максимальної швидкодії, схеми багаторозрядних суматорів будуються безпосередньо за логічними виразами, одержуваними з таблиць істинності.

Інтегральні схеми суматорів

Мікросхеми суматорів позначаються літерами “ІМ” в умовному позначенні. Наприклад, **K555ІМ2** (рис. 4.41).

Даний суматор становить ІС, яка виконує операцію підсумовування двох розрядів двох двійкових чисел. Для його використання в цифрових схемах з розрядністю більше двох суматор має необхідні входи/виходи розширення: **Crin** – вхід для подачі значення переносу із суматора попередніх розрядів; **Crin + 1** – вихід для передачі значення переносу в суматор старших розрядів.

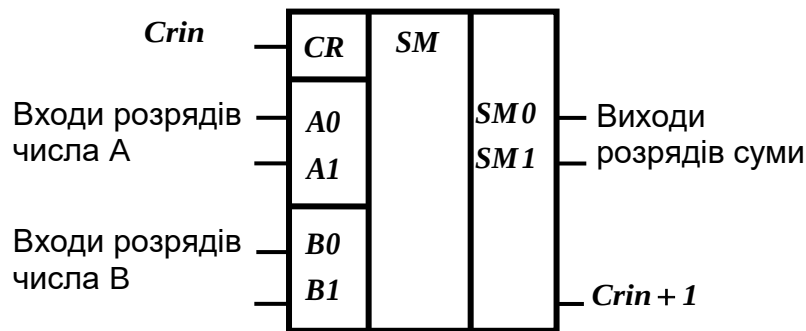


Рис. 4.41. Умовне графічне позначення дворозрядного повного суматора К555ІМ2

Висновки:

суматори бувають багаторозрядними й однорозрядними (півсуматори);
 для нарощування розрядності використовується послідовне і паралельне з'єднання однорозрядних суматорів;
 для паралельного з'єднання необхідні вузли прискореного переносу;
 найбільшу швидкодію мають суматори з паралельним переносом.

4.6. Компаратори

Компаратором називається пристрій комбінаційного типу, призначений для порівняння двох двійкових чисел А і В.

Результатом порівняння може бути три результати: $A < B$; $A = B$; $A > B$.

Умовне графічне позначення компаратора містить у собі такі поля: два поля входів розрядів чисел А і В; виходи результатів порівняння (рис. 4.42, 4.43).

Результати порівняння збігаються з результатами порівняння в десятковій арифметиці. Залежно від розрядності вхідних чисел розрізняють однорозрядні та багаторозрядні компаратори. Кількість виходів при цьому залишається постійною. Крім цього необхідно знати, що для розв'язання деяких конкретних задач може знадобитися визначення тільки одного стану з трьох, наприклад, $A = B$. У цьому випадку надлишкові виходи можуть бути виключені (що спростить схему компаратора). Такі компаратори називаються неповними.

Функціонування і синтез однорозрядного компаратора

Умовне графічне позначення однорозрядного компаратора наведене на рис. 4.43.

Відповідно до таблиці істинності однорозрядного компаратора (табл. 4.13) отримаємо логічні вирази для виходів компаратора. Оскільки компаратор

має три виходи, то кількість логічних виразів також буде дорівнювати трьом. Для кожного виходу має місце свій логічний вираз.

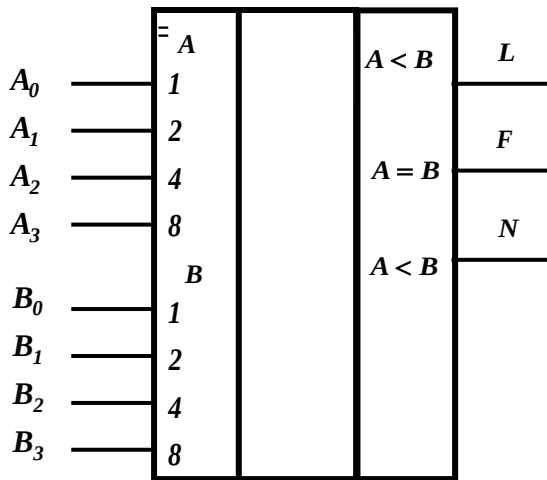


Рис. 4.42. Умовне графічне позначення компаратора

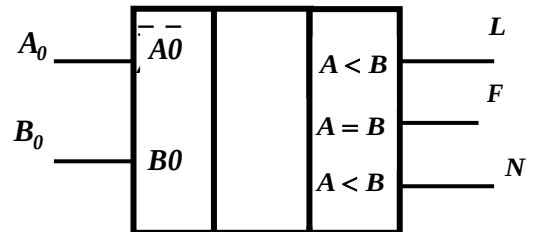


Рис. 4.43. Умовне графічне позначення

Таблиця 4.13

Таблиця істинності однорозрядного компаратора

Входи		Виходи		
$A0$	$B0$	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Для зручності умовно позначимо виходи компаратора: $A > B$ як $Y_2(L)$; $A = B$ як $Y_1(F)$; $A < B$ як $Y_0(N)$.

Тоді на виходах компаратора буде:

$$Y_2 = A_0 \wedge \overline{B_0}; Y_1 = A_0 \wedge B_0 + \overline{A_0} \wedge \overline{B_0}; Y_0 = \overline{A_0} \wedge B_0.$$

Синтез багаторозрядного компаратора розглядається нижче. Багаторозрядні компаратори мають більш складну таблицю істинності, а отже і структуру, ніж однорозрядні компаратори. Розглянемо лише кілька простих, але важливих прикладів [10].

Багаторозрядні компаратори

Компаратор збігу (рівнозначності). Найпростішу структуру має багаторозрядний компаратор рівнозначності (збігу), який дуже широко використовується в цифровій техніці.

Для перевірки рівності двох чисел необхідно перевірити збіг значень кожного i -го розряду (табл. 4.14).

Таблиця 4.14

Таблиця істинності компаратора збігу

Входи		Вихід
A_j	B_j	$A = B(F)$
0	0	1
0	1	0
1	0	0
1	1	1

Для компараторів нерівнозначності двох чисел необхідно враховувати не тільки співвідношення значень окремих розрядів чисел, але і старшинство розрядів. Перевага старшого розряду однозначно визначає старшинство всього числа. Тому логічні вирази для компараторів нерівнозначності виходять досить складними.

Нарощування розрядності компараторів

Часто для багаторозрядних пристроїв неможливо знайти або підібрати для відповідної задачі багаторозрядний компаратор. У цьому випадку діють так: використовують кілька компараторів малої розрядності і поєднують їх певним чином. Для вирішення поставленої задачі в структурі компараторів передбачаються додаткові входи/виходи розширення/нарощування розрядності.

Дана таблиця істинності збігається з таблицею істинності логічного елемента **РІВНОЗНАЧНОСТІ** або **ІНВЕРСНОГО ВИКЛЮЧАЛЬНЕ ЧИ**. Виконання даної операції позначається значком $\sim$.

Очевидно, щоб два багаторозрядних числа були рівні, необхідно, щоб були рівні значення всіх розрядів обох чисел $\forall i(A_i = B_i)$. Цю операцію можна виконати за допомогою логічного елемента **I**. Тоді логічний вираз для багаторозрядного компаратора збігу буде мати вигляд:

$$F = A_0 \sim B_0 \cdot A_1 \sim B_1 \dots A_n \sim B_n.$$

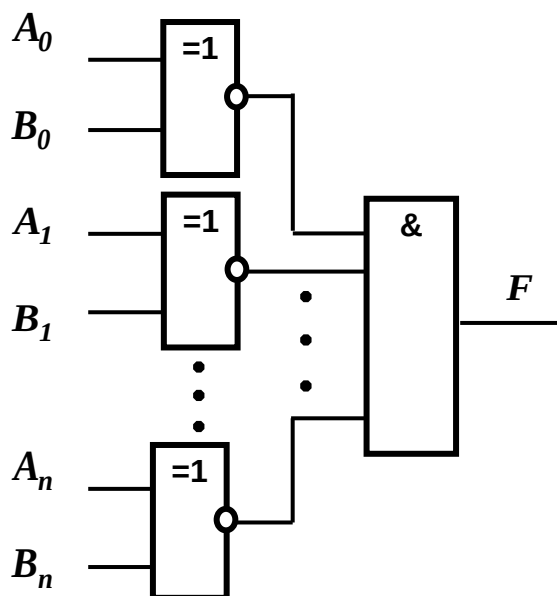


Рис. 4.44. Структурна схема компаратора рівнозначності на елементах **ВИКЛЮЧАЛЬНЕ ЧИ**

Відповідно до даного логічного виразу побудована функціональна схема багаторозрядного компаратора рівнозначності (рис. 4.44).

Як приклад можна розглянути компаратор **K555СП1** (рис. 4.45). Він, поряд з чотирма парами входів, які приймають для аналізу два чотирьохрозрядні числа A і B , має три входи $EX(A > B)$, $EX(A = B)$, $EX(A < B)$, призначені для розширення (нарощування) розрядності компаратора.

Для збільшення розрядності компаратори можна з'єднувати послідовно (рис. 4.46). Варто помітити, що на входи розширення компаратора молод-

ших розрядів (**D1**) необхідно подати визначені логічні рівні: на входи $A > B$ і $A < B$ подається потенціал логічної одиниці U^1 , а на вхід $A = B$ – логічного нуля U^0 . У протилежному випадку компаратор не зможе правильно формувати вихідні значення.

Крім послідовного з'єднання на практиці при розрядності вхідних операндів менше **24** знаходить застосування і пірамідальне з'єднання компараторів (рис. 4.47).

Перевагою послідовного з'єднання є можливість необмеженого нарощування розрядів за рахунок послідовного з'єднання компараторів, однак у такому випадку знижується швидкодія усього вузла, оскільки при зміні значення на вході, компаратори будуть перемикатися послідовно від молодшого розряду до старшого. Таким чином, швидкодія (час перемикання) $T_{Пер}$ буде визначатися як швидкодією кожного компаратора (T_0), так і їх кількістю n : $T_{Пер} = nT_0$.

Зі структури схеми пірамідального з'єднання випливає, що її час перемикання буде складати $T_{Пер} = 2T_0$. Отже, швидкодія даної схеми буде вище, ніж швидкодія схеми з послідовним з'єднанням.

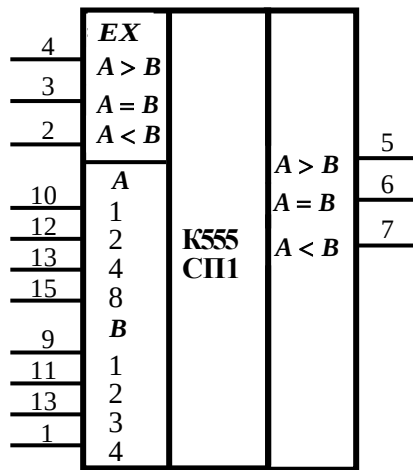


Рис. 4.45. Умове графічне позначення компаратора K555СП1 (K531СП1)

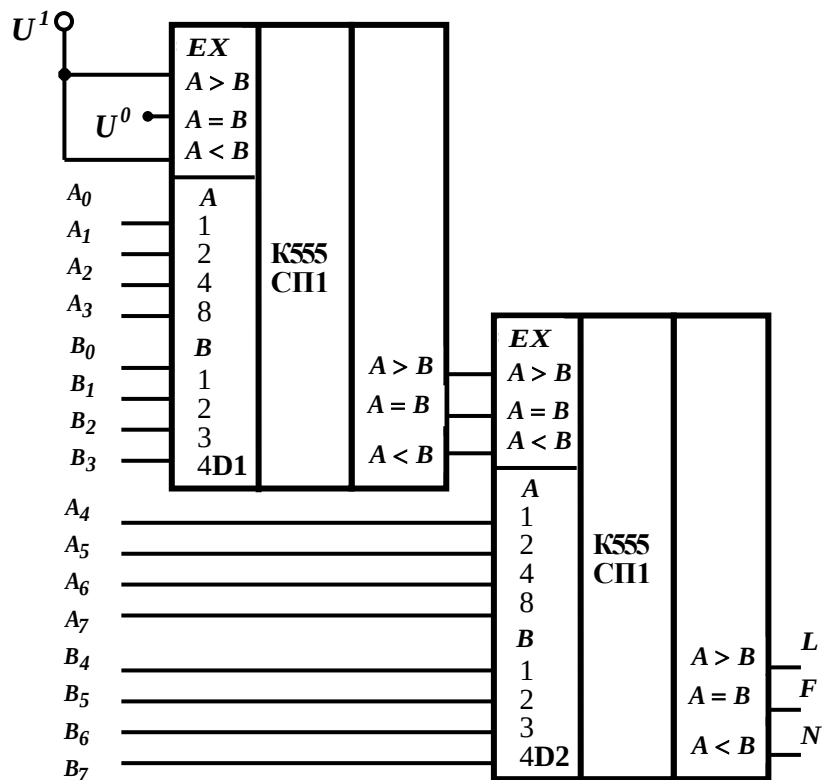


Рис. 4.46. Послідовне нарощування розрядності за допомогою компараторів K555СП1

Висновки:

- компаратори бувають багаторозрядними й однорозрядними;
- деякі виходи у компараторів можуть бути відсутні;
- для нарощування розрядності використовується послідовне і пірамідальне з'єднання компараторів;
- найбільшу швидкодію мають багаторозрядні компаратори з пірамідальним з'єднанням.

4.7. Програмовані логічні матриці

Програмовані логічні матриці (ПЛМ) становлять функціональний блок, виконаний у вигляді великої інтегральної схеми на базі напівпровідникової технології і призначені для реалізації цифрових пристроїв.

ПЛМ є невід'ємною частиною мікропроцесорів, в яких вони виконують частину функцій пристрою управління, який формує сигнали мікрооперацій у відповідності з кодом операцій, що поступає на вхід мікропроцесора.

Умовне графічне позначення ПЛМ (рис.4.48) передбачає прямокутник, в основному полі якого зображується найменування повзводної функції **PLM** (від англійського “**program logic memory**”).

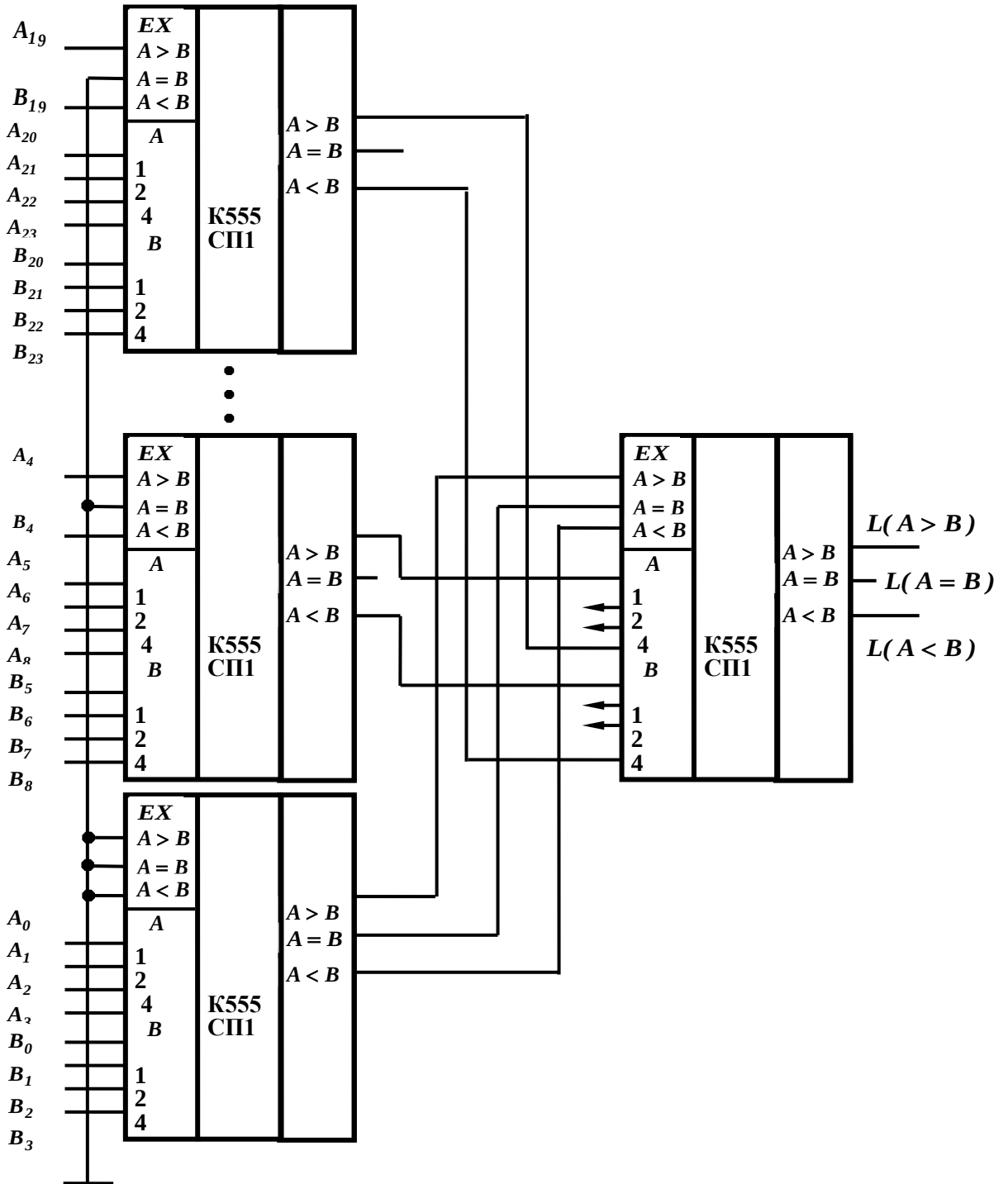


Рис. 4.47. Схема пірамідального 24-розрядного компаратора

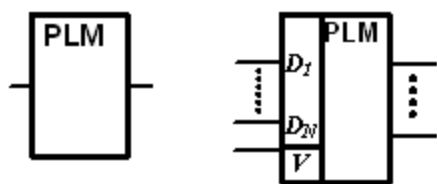


Рис. 4.48. Умовне графічне позначення ПЛМ

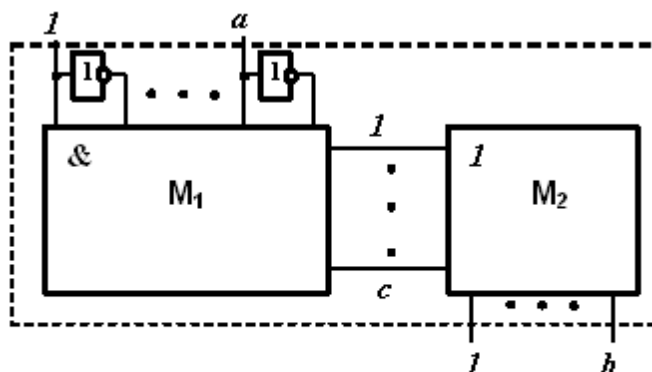


Рис. 4.49. Структурна схема комбінаційної ПЛМ

Комбінаційні **ПЛМ** складаються з двох матриць **M₁** та **M₂** (рис.4.49), перша з яких виконує функцію формування кон'юнкцій вхідних змінних (термів), а друга – функцію формування диз'юнкцій термів, отриманих в матриці **M₁**. Кожна **ПЛМ** характеризується числом входів – **a**, виходів – **b** та термів – **c**. Тому як правило комбінаційні матриці позначаються **ПЛМ (a, b, c)**. Навпаки, від комбінаційних **ПЛМ** програмуємі логічні матриці з пам'яттю вміщують **d** – розрядний регістр в колі зворотного зв'язку між – матрицями **M₁** та **M₂**.

У матриці **M₁** може бути сформовано **c** термів від **a + d** змінних, а в матриці **M₂** – **b + d** диз'юнкцій цих термів.

Розглянемо особливості технічної реалізації комбінаційних **ПЛМ** на біполярних елементах.

Припустимо, що необхідно реалізувати цифровий пристрій, який опису-

ється системою логічних функцій:

$$\begin{cases} Y_1 = \overline{x_3} \overline{x_2} x_1 \vee \overline{x_4} x_2 \\ Y_2 = x_4 x_3 \overline{x_2} \vee x_3 \overline{x_2} x_1 \\ Y_3 = \overline{x_4} x_2 \vee x_4 \overline{x_3} \overline{x_2} \end{cases}$$

Для матричної реалізації цієї системи потребується **ПЛМ**, яка має чотири входи і три терми ($\overline{x_3} \overline{x_2} x_1$, $\overline{x_4} x_2$, $x_4 \overline{x_3} \overline{x_2}$), тобто (**a** = 4, **b** = 3, **c** = 3). На рис. 4.50 наведена схема **ПЛМ**, яка реалізує задану систему логічних функцій.

На трьох горизонтальних шинах цієї **ПЛМ** реалізуються терми $\overline{x_3} \overline{x_2} x_1$, $\overline{x_4} x_2$, $x_4 \overline{x_3} \overline{x_2}$. Якщо змінна входить до терму, який необхідно сформувати, то діод підключається між вертикальною шиною цієї змінної або її

інверсії та горизонтальною шиною, на якій реалізується даний терм. Умовно це з'єднання показана точкою.

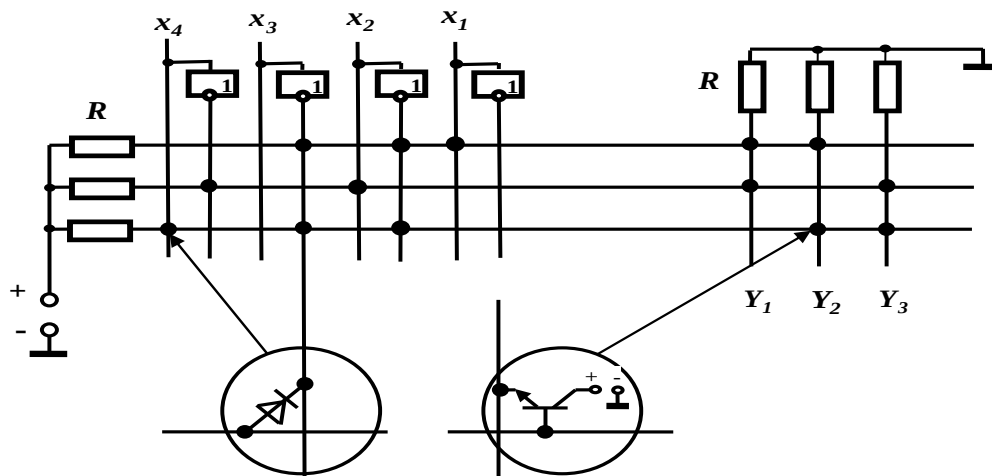


Рис. 4.50. Функціональна схема ПЛМ

На вертикальних шинах диз'юнктивної матриці реалізується диз'юнкція термів $\overline{x_3 x_2 x_1}$ та $\overline{x_4 x_2}$, $\overline{x_4 x_3 x_2}$ та $\overline{x_3 x_2 x_1}$, $\overline{x_4 x_3 x_2}$ та $\overline{x_4 x_2}$, що входять до виразу для змінних Y_1 , Y_2 , Y_3 відповідно. Якщо терм входить до диз'юнкції, то до матриці M_2 на перехресті горизонтальної шини, яка відповідає терму, і вертикальної шини, яка відповідає вихідній змінній Y_i , до якої входить цей терм, ставиться точка, що позначає наявність з'єднання. Це з'єднання реалізується шляхом підключення емітера біполярного транзистора $n-p-n$ типу до вертикальної шини, бази цього транзистора до горизонтальної шини, а колектора до "плюсу" джерела живлення.

У ланцюгах живлення, які з'єднують "плюс" джерела живлення з горизонтальними шинами та "мінус" джерела з вертикальними шинами матриці M_2 , стоять обмежуючі резистори.

Таким чином, точки на перехресті вертикальних шин матриці M_1 з горизонтальними шинами відповідає кон'юнкціям вхідних змінних, що входять до різних логічних функцій, які реалізує ПЛМ, а точки на перехресті матриці M_2 з горизонтальними шинами відповідає диз'юнкції термів, від яких залежить вихідні змінні.

При використанні ПЛМ, виконаних на МОН-елементах, кожна горизонтальна шина матриці M_1 і кожна вертикальна шина матриці M_2 виконують логічну функцію ЧИ-НІ. В цьому випадку система логічних функцій, що описує цифровий пристрій, переводиться до базису ЧИ-НІ.

Для прикладу, що розглядається, таке перетворення буде виглядати таким чином:

$$\begin{cases} Y_1 = \overline{x_3 \vee x_2 \vee x_1 \vee x_4 \vee x_2} \\ Y_2 = \overline{x_4 \vee x_3 \vee x_2 \vee x_3 \vee x_2 \vee x_1} \\ Y_3 = \overline{x_4 \vee x_2 \vee x_4 \vee x_3 \vee x_2} \end{cases}$$

У ПЛМ побудованих на МОН елементах, значення вихідних змінних отримується в інверсному вигляді, цьому дану систему логічних функцій доцільно представити у наступному вигляді:

$$\begin{cases} \overline{Y_1} = \overline{x_3 \vee x_2 \vee x_1 \vee x_4 \vee x_2} \\ \overline{Y_2} = \overline{x_4 \vee x_3 \vee x_2 \vee x_3 \vee x_2 \vee x_1} \\ \overline{Y_3} = \overline{x_4 \vee x_2 \vee x_4 \vee x_3 \vee x_2} \end{cases}$$

Функціональна схема ПЛМ, яка реалізує дану систему логічних функцій, наведена на рис. 4.51.

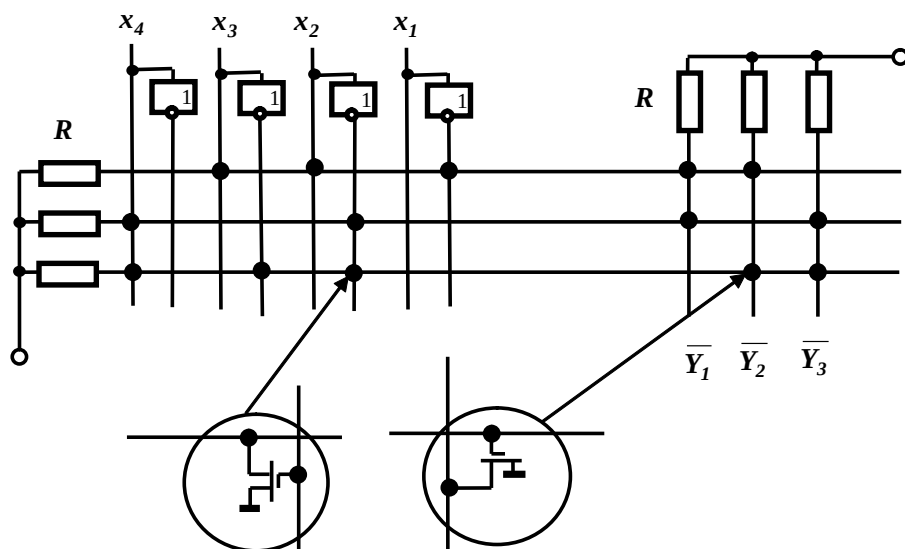


Рис. 4.51. Функціональна схема ПЛМ на МОН-транзисторах

КОНТРОЛЬНІ ПИТАННЯ

1. Дати визначення дешифратору (шифратору).

2. Охарактеризувати існуючу залежність між кількістю входів та виходів дешифратора (шифратора).

3. Дати характеристику особливостей побудови пірамідального дешифратора. Навести його переваги та недоліки.

4. Скільки виходів повинні мати восьми – та тринадцятиходові шифратори? Чому?

5. Пояснити принцип роботи мультиплексора (демультиплексора).

6. Скільки адресних входів повинен мати шістнадцятиходовий мультиплексор. Чому?

7. Визначити вираз, який правильно описує роботу чотирьох-ходового мультиплексора з інверсним виходом (A_i – позначення адресних входів, IN_i – позначення інформаційних входів)

$$Y = \overline{A_1} \cdot \overline{A_0} \cdot IN_0 + \overline{A_1} \cdot A_0 \cdot IN_1 + A_1 \cdot \overline{A_0} \cdot IN_2 + A_1 \cdot A_0 \cdot IN_3$$

$$Y = \overline{A_1} \cdot \overline{A_0} \cdot IN_0 + \overline{A_1} \cdot A_0 \cdot IN_1 + A_1 \cdot \overline{A_0} \cdot IN_2 + A_1 \cdot A_0 \cdot IN_3$$

$$Y = \overline{A_1} \cdot \overline{A_0} \cdot IN_0 + \overline{A_1} \cdot A_0 \cdot IN_1 + A_1 \cdot \overline{A_0} \cdot IN_2 + A_1 \cdot A_0 \cdot IN_3.$$

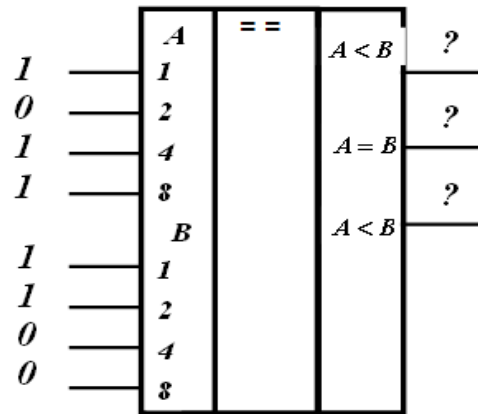
8. Реалізувати повністю визначену логічну функцію двох змінних на мультиплексорі $Y(x_2, x_1) = x_1 \cdot x_0 + x_1 \cdot x_0$.

9. Заповнити всі чарунки таблиці істинності однорозрядного суматора.

Входи			Значення суми	Виходи	
<i>Crin</i>	<i>A</i>	<i>B</i>	<i>Binary</i>	<i>Crout</i>	<i>SM</i>
0	0	0	00	0	0
0		1	01	0	1
0	1	0		0	1
	1	1	10	1	0
1	0	0	01		1
1	0		10	1	0
1	1	0	10	1	0
1	1	1	11	1	

10. Визначити додток двох чисел $A_{10} = 9$ та $B_{10} = 14$ за правилами двійкової арифметики.

11. Визначити значення сигналу на виході компаратора.



12. Реалізувати на ПЛМ логічний пристрій, який описується системою логічних функцій:
$$\begin{cases} Y_1 = x_3 \underline{x_1} \vee \overline{x_4} x_2 \\ Y_2 = x_4 x_3 \underline{x_1} \vee x_4 \end{cases}.$$

5. СХЕМОТЕХНІКА ЦИФРОВИХ ЕЛЕМЕНТІВ

5.1. Поняття про цифрові елементи з пам'яттю

Функціональні інтегральні схеми та пристрої накопичувального типу містять елементи пам'яті, здатні зберігати двійкову інформацію тривалий час. Завдяки елементам пам'яті, роль яких зазвичай виконують тригери, стан накопичувальних ІС і пристроїв визначається не тільки конкретною комбінацією вхідних змінних у даний момент часу, але і станом пристрою до подачі цієї комбінації, тобто визначається послідовністю надходження вхідних сигналів. Тому поряд з терміном “пристрій накопичувального типу” застосовується термін “пристрій послідовного типу”.

До накопичувальних пристроїв відносяться тригери, лічильники, регістри, запам'ятовуючі пристрої, суматори тощо.

Їх роботу прийнято розглядати в безрозмірному дискретному часі, для чого реальний час розбивається на інтервали, які нумеруються, починаючи з деякого моменту. Кожен такий проміжок часу називають тактовим інтервалом. Тривалість його для характеристики роботи пристрою не має значення. Часові затримки, обумовлені перехідними процесами, зазвичай не враховують, однак, коли частота зміни тактів велика і порівнянна зі швидкістю пристрою, з ними доводиться рахуватися.

Більшість накопичувальних пристроїв є *синхронними*, тобто мають спеціальні входи, на які подають синхронізуючі сигнали (тактові), які є своєрідними виконавчими командами. Вони визначають момент початку роботи пристрою, отже, і момент початку тактового інтервалу. Якщо пристрій *асинхронний*, то початок тактового інтервалу прив'язують до моменту надходження будь-якого інформаційного сигналу.

Роботу накопичувальних пристроїв прийнято описувати, пов'язуючи їхній стан між двома суміжними тактами та t^{n+1} або t^{n-1} та t^n . Відповідно до позначень вхідних і вихідних сигналів, які діють у межах такту, додають номер такту. Так, запис $Q^n = 1$ означає, що на виводі Q в інтервалі $t^n \leq t < t^{n+1}$ діє сигнал логічної одиниці.

Способи опису роботи накопичувальних пристроїв подібні способам опису роботи комбінаційних пристроїв – у вигляді таблиць і виразів, які одержали назву таблиць станів і функцій переходів. Ті й інші виражають зв'язок між вихідними та вхідними сигналами в $(n + 1)$ -му такті з урахуванням стану пристрою в попередньому n -му такті. При великій кількості входів і виходів повні

таблиці станів виходять громіздкими. Тому на практиці частіше користуються скороченими таблицями, у яких стовпці з вихідними сигналами в n -му такті виключені, а стан в $(n + 1)$ -му такті записуються в узагальненому вигляді, наприклад $Q^{n+1} = \overline{Q^n}$. Це означає, що вихідний сигнал змінив своє значення на протилежне (0 на 1 або 1 на 0).

Як відзначалося вище, накопичувальні пристрої можуть бути синхронними й асинхронними. *Асинхронні* переходять у новий стан відразу після надходження на входи визначеної комбінації інформаційних сигналів. *Синхронні* пристрої мають вхід синхронізації і перемикаються тільки за наявності на них синхронізуючого сигналу.

Вхід синхронізації може бути статичним і динамічним. Той і інший, у свою чергу, буває прямим й інверсним.

Статичним входом називають вхід, через який вплив сигналу на пристрій передається протягом усього часу його існування. Причому, якщо під час дії синхронізуючого сигналу будуть змінюватися сигнали на інформаційних входах, пристрій буде реагувати на ці зміни. У цьому випадку статичний вхід синхронізації еквівалентний входові дозволу в пристроях комбінаційного типу.

Статичними входами зазвичай є також входи, на які подаються інформаційні сигнали.

Динамічним входом називають вхід, через який вплив на пристрій передається тільки у момент перепаду вхідного сигналу з 0 у 1 або, навпаки, з 1 у 0 . Властивість такого входу подібна властивості диференціюючого кола (рис. 5.1) виділяти перепади напруги.

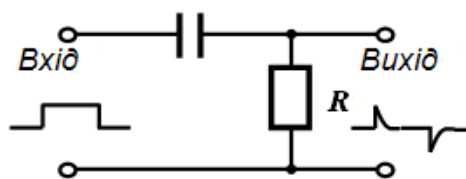


Рис. 5.1. Диференціююче коло (до поняття динамічного входу)

На рис. 5.2 показані варіанти умовних графічних позначень (УГП) різних типів входів, а також статичних виходів. При необхідності мітки виводів доповнюють літерами і символами, які показують функціональне призначення виводів. Помітимо, що такими ж літерами та символами позначають виводи і на структурних схемах, якщо виводи прямі, і доповнюють позначення інверсною рисою (наприклад, $\overline{R}$, $\overline{CR}$), якщо вони інверсні [11].

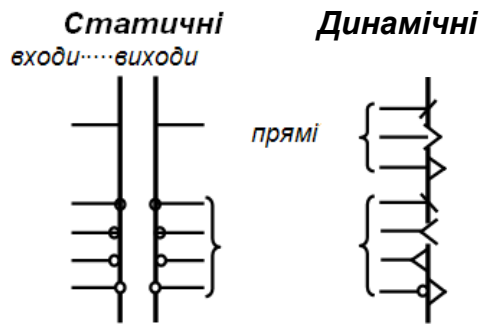


Рис. 5.2. Варіанти УГП входів і виходів цифрових пристроїв

Щоб уникнути плутанини в таблицях станів накопичувальних пристроїв, умовимося приводити не двійкові змінні, як це роблять деякі автори, а відповідні їм значення сигналів у позитивній логіці, тобто під цифрою **1** будемо мати на увазі напругу високого рівня, а під **0** – низького. Крім того, оскільки входи і сигнали, що надходять на них, зазвичай позначають однаковими літерами, для розрізнення будемо вводити в позначеннях сигналів або тактові індекси (наприклад, C^{n+1} , Q^n), або штрихову мітку (C' , J').

Синтез накопичувальних пристроїв ґрунтується на складанні спеціальних таблиць і карт переходів та їхньої обробки за визначеними правилами. Цей процес досить складний. Необхідність у синтезі виникає тільки у розробників ІС, тому тут він не описується.

Основні параметри накопичувальних пристроїв, які характеризують споживчі властивості, аналогічні параметрам комбінаційних пристроїв.

5.2. Тригери на логічних елементах

Тригер – це електронний пристрій, здатний довгостроково залишатися в одному з двох стійких станів і змінювати стан під впливом зовнішніх сигналів. Він може залишатися в заданому стані і після припинення дії перемикаючих сигналів. Приймавши один зі станів за одиничний, а інший за нульовий, можна вважати, що тригер зберігає (пам'ятає) один розряд двійкового числа (один біт інформації).

Тригери, як вироби мікроелектроніки, за характером функціонування бувають статичними та динамічними. Основу динамічних тригерів складає **МОН**-структура, яка виконує роль конденсатора. Той або інший стан характеризується наявністю або відсутністю заряду на конденсаторі. Оскільки заряд поступово розсмоктується, динамічні тригери мають потребу в періодичному відновленні інформації.

Статичні тригери, про які мова йде в даному підрозділі, будуються на логічних елементах. Вони не вимагають періодичної регенерації інформації.

Тригер, як правило, має два виходи. Один з них називається прямим і позначається буквою Q , а інший – інверсним і позначається $\bar{Q}$. Стан тригера ототожнюють із сигналом на прямому виході, тобто говорять, що тригер знаходиться в одиничному стані, коли $Q = 1$ а $\bar{Q} = 0$, і в нульовому, коли $Q = 0$, $\bar{Q} = 1$.

Зміна станів має багато синонімів: перемикання, перекидання, запис інформації. Узагальнена структурна схема статичних тригерів показана на рис. 5.3. Тригер містить чарунку пам'яті і вузол керування.

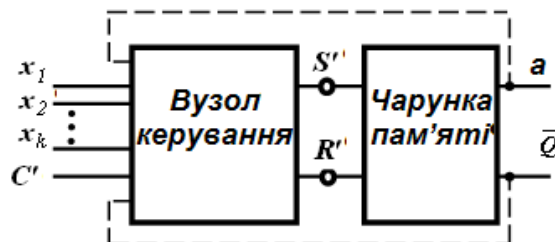


Рис. 5.3. Узагальнена структурна схема тригера

На входи вузла керування подаються зовнішні інформаційні сигнали x_i та синхронізуючий сигнал C' . Крім того, можуть надходити сигнали Q і $\bar{Q}$, які інформують про стан чарунки пам'яті. Вузол керування генерує сигнали S' і R' , які перемикають тригер відповідно до функції переходів:

$$Q^{n+1} = f(x_1, x_2, \dots, x_k, C', Q^n),$$

де k – кількість зовнішніх інформаційних сигналів.

Тригери на логічних елементах досить різноманітні, тому їх підрозділяють за рядом ознак: за способом запису інформації розрізняють тригери асинхронні і синхронні; за структурою побудови – одноступінчаті і двоступінчаті; за функціональним призначенням – тригери RS -, T -, D -, JK -типів та ін.

5.2.1. RS -тригери

Асинхронні RS -тригери – це найпростіші тригери, які використовуються як чарунки пам'яті, зокрема, у тригерах інших типів. Свою назву вони отримали за позначенням входів: S (*set* – установка), за допомогою якого встановлю-

ється одиничний стан, і **R** (*reset*– скидання), за допомогою якого встановлюється нульовий стан.

Для побудови асинхронного **RS**-тригера досить охопити петлею взаємного зворотного зв'язку два **ЛЕ НІ-ЧИ (НІ-І)**. Функціональна схема такого тригера в базисі **НІ-ЧИ** та його умовне графічне позначення наведене на рис. 5.4, а опис роботи – табл. 5.1.

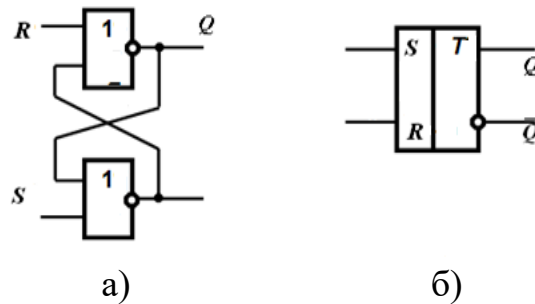


Рис. 5.4. Схема (а) та умовне графічне позначення (б) RS-тригера з прямими входами

Таблиця 5.1

Опис роботи RS-тригера

n	$n+1$			
Q^n	S^{n+1}	R^{n+1}	Q^{n+1}	$\overline{Q^{n+1}}$
0	0	0	0	1
1	0	0	1	0
0	1	0	1	0
1	1	0	1	0
1	0	1	0	1
0	0	1	0	1
0	1	1	н/в	н/в
1	1	1	н/в	н/в

Неважко переконатися, що при сигналах $S^{n+1} = R^{n+1} = 0$ стан тригера не змінюється ($Q^{n+1} = Q^n$) – режим збереження. При подачі на входи комбінацій $S^{n+1} = 1, R^{n+1} = 0, S^{n+1} = 0, R^{n+1} = 1$ встановлюються відповідно одиничний і нульовий стани. Якщо подані $S^{n+1} = R^{n+1} = 1$, тобто одночасно діють дві протилежні команди, стан невизначений (н/в). Невизначений у тому плані, що якщо невідома функціональна схема тригера, про вихідні сигнали нічого визначеного сказати не можна. У даному випадку (рис. 5.4 а)) на обох ви-

ходах нулі, і тригер, як пристрій із двома нерівнозначними виходами Q та $\overline{Q}$ втрачає сенс. Крім того, якщо після комбінації $S^n = R^n = 1$ одночасно подати $S^{n+1} = R^{n+1} = 0$, тригер установиться в один із двох станів непередбачено. Тому комбінація $S^{n+1} = R^{n+1} = 1$ тут неприпустима. УГП RS тригера наведено на рис. 5.4 б).

Скорочена форма задання роботи тригера наведена в табл. 5.2.

У ній виключений стовпчик для Q^n і, оскільки всі інші сигнали відносяться до $(n+1)$ -го такту, спрощена індексація вхідних сигналів (S', R') .

Таблиця 5.2

Скорочена таблиця функціонування RS -тригера з прямими входами

S'	R'	Q^{n+1}	$\overline{Q}^{n+1}$
0	0	Q^n	$\overline{Q}^n$
1	0	1	0
0	1	0	1
1	1	н/в	н/в

Часові діаграми, які показують роботу тригера, наведені на рис. 5.5.

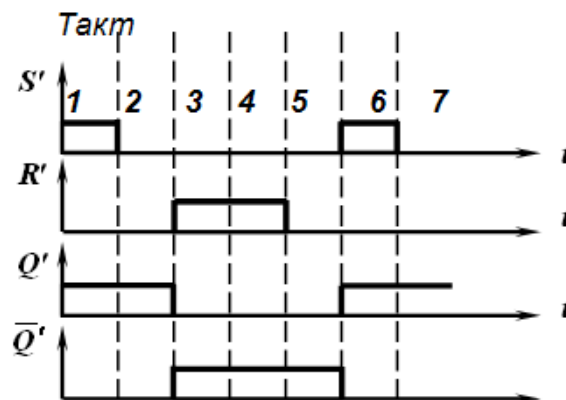


Рис. 5.5. Часові діаграми RS -тригера з прямими входами

Нагадаємо, що перепади вхідних сигналів зазвичай суміщають з початком умовних тактових інтервалів. Тому надалі без особливої потреби ці інтервали в явному вигляді виділятися не будуть.

При використанні ЛЕ І-НІ (рис. 5.6) входи S та R стають інверсними. Це означає, що вплив по них здійснюється низьким рівнем напруги, і тригер функціонує відповідно з табл. 5.3.

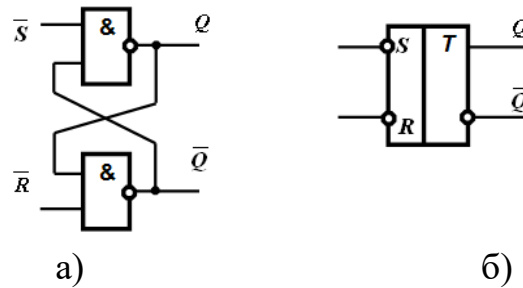


Рис. 5.6. Схема (а) та умовне графічне позначення (б) RS-тригера з інверсними входами

Таблиця 5.3

Скорочена таблиця функціонування RS-тригера з інверсними входами

S'	R'	Q^{n+1}	$\overline{Q^{n+1}}$
1	1	Q^n	$\overline{Q^n}$
0	1	1	0
1	0	0	1
0	0	н/в	н/в

Сигнали $S' = R' = 1$ переводять тригер у режим збереження. При $S' = 0$ і $R' = 1$ на виході $Q^{n+1} = 1$, а при $S' = 1$ та $R' = 0$ на виході $Q^{n+1} = 0$. При комбінації $S' = R' = 0$ стан невизначений.

Швидкодія асинхронного RS-тригера – це максимальна частота проходження сигналів S' та R' , при якій він функціонує без збоїв. Очевидно, вона визначається часом перемикавання обох ЛЕ: $F_{max} = \frac{1}{2t_{3m_{сер}}}$.

Синхронні RS-тригери. На рис. 5.7 показана функціональна схема синхронного RS-тригера на елементах **НІ-І** (рис.5.7 а) та його умовне графічне позначення (рис. 5.7 б). Він містить синхронний RS-тригер на ЛЕ3 і ЛЕ4 та вузол керування на ЛЕ1 і ЛЕ2, об'єднаних за входом синхронізації C . Таблиця станів його при $C' = 1$ така ж, як для асинхронного тригера з прямими входами (табл. 5.2).

Дійсно, нехай $C' = 0$. Незалежно від значень зовнішніх сигналів S' і R' на входах асинхронного тригера $S'_1 = R'_1 = 1$. Він має інверсні входи і відповідно до табл. 5.3 працює у режимі збереження. Нехай $C' = 1$. Якщо $S'_1 = R'_1 = 0$, то $S'_1 = R'_1 = 1$ – режим збереження. Якщо $S' = 1$ та $R' = 0$, то

$S'_1 = 0$ і $R'_1 = 1$, тому $Q' = 1$. При $S' = 0$ і $R' = 1$ навпаки – $Q' = 0$. Комбінація $S' = R' = 1$, як і в асинхронних тригерах, неприпустима.

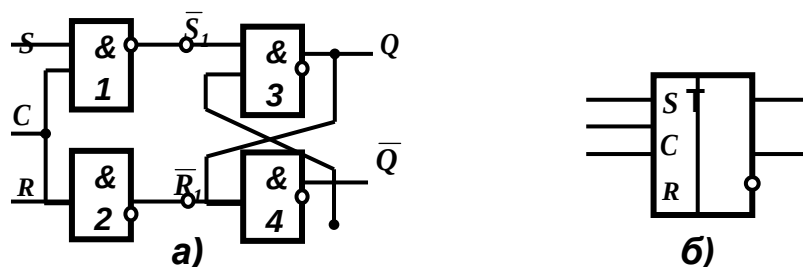


Рис. 5.7. Схема (а) та УГП (б) синхронного RS-тригера

На підставі принципу подвійності можна стверджувати, що якщо тригер виконується на елементах **НІ-ЧИ**, усі його входи стають інверсними. Він працює відповідно до табл. 5.3 при $C' = 0$.

У цифровій апаратурі знаходять застосування також двоступінчаті тригери. Структура й умовне позначення двоступінчатого синхронного RS-тригера зображені на рис. 5.8.

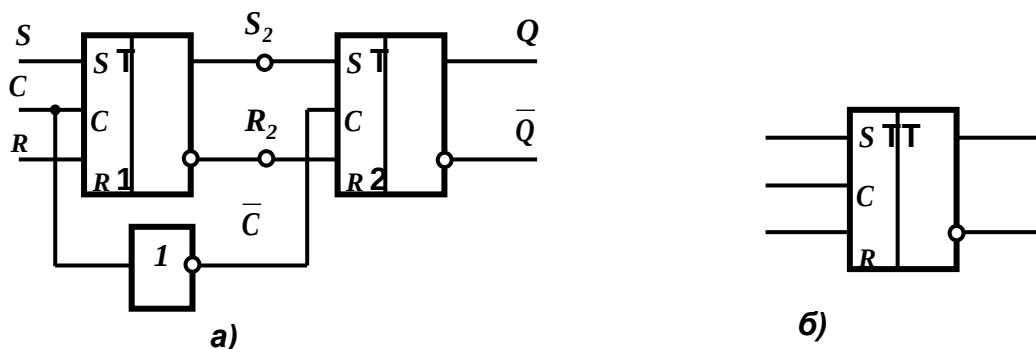


Рис. 5.8. Схема (а) та УГП (б) двоступінчатого **RC** -тригера

В основному полі графічного позначення розташовані дві букви “Т”. Ступені схеми, які становлять одноступінчаті синхронні RS-тригери, включені послідовної завдяки інверторам керуються протифазними синхронізуючими сигналами. Тому якщо перший ступінь працює у режимі прийому інформації ($C' = 1$), то другий – у режимі збереження ($\bar{C}' = 0$) і навпаки: при $C' = 0$ для першого ступеня – це режим збереження, а для другого – режим перепису інформації з першого ступеня.

На рис. 5.9 показані часові діаграми роботи тригера. Видно, що стан другого ступеня повторює стан першого із затримкою t_c , яка дорівнює тривалості імпульсу синхронізації C' . Тригер нібито має властивість внутрішньої затрим-

ки, яка іноді буває необхідна. Зокрема, вона потрібна для побудови регістрів зсуву.

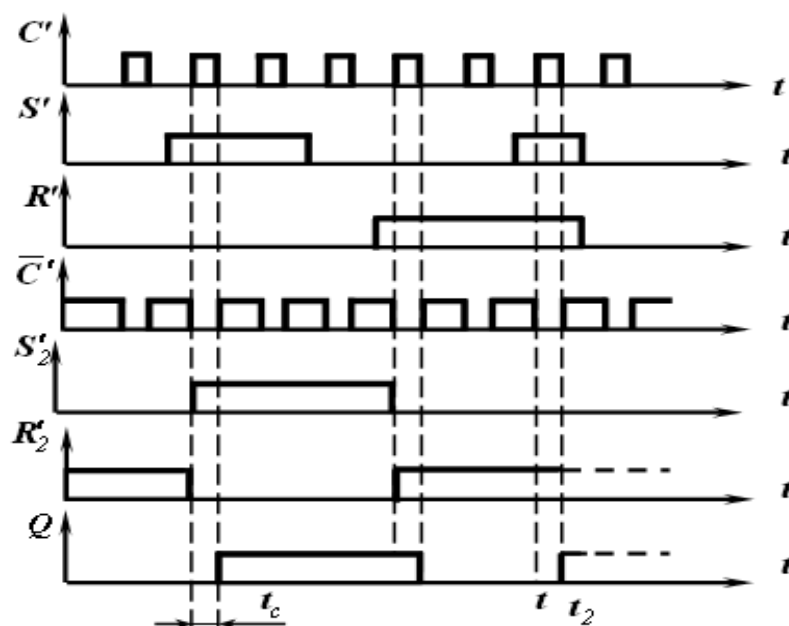


Рис. 5.9. Часові діаграми двоступінчатого *RS*-тригера (схема якого зображена на рис. 5.8)

Той факт, що перемикання другого ступеня відбувається за спадом імпульсу C' (хоча самі тригери ступенів мають прямі входи), дозволяє констатувати наступне: *двоступінчатість структури тригера еквівалентна інверсії типу входу синхронізації* – прямого на інверсний і навпаки.

Помітимо, що при двоступінчатій структурі не існує комбінації вхідних сигналів, при якій тригер втрачає сенс. Проте комбінація $S' = R' = 1$ неприпустима. Пояснюється це тим, що при $S' = R' = 1$ та $C' = 1$ на входах другого ступеня $S'_2 = R'_2 = 1$ (рис. 5.9, $t_1 < t < t_2$), і після того як сигнал C' стає нульовим ($t = t_2$), тригер першого ступеня переходить у непередбачений (невизначений) стан. Отже, в такий же стан перемикається тригер другого ступеня.

5.2.2. Універсальні тригери (*JK*-тригери)

JK-тригер має два інформаційні входи (J , K) і вхід синхронізації (рис. 5.10).

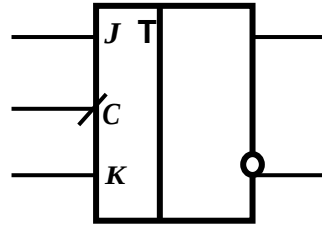
Працює він відповідно до функції переходів $Q^{n+1} = \overline{K'} Q^n \vee J' \overline{Q}^n$ або відповідної їй таблиці станів (табл. 5.4).

JK

Таблиця 5.4

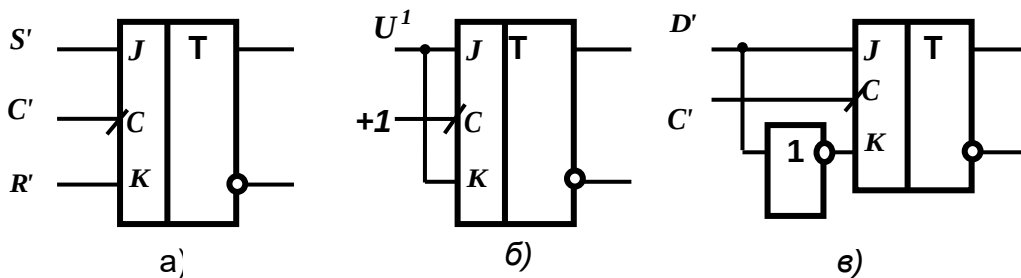
Таблиця станів JK-тригера

J'	K'	Q^{n+1}
0	0	Q^n
1	0	1
0	1	0
1	1	$\overline{Q^n}$

Рис. 5.10. Умовне графічне позначення JK -тригера

З надходженням синхронізуючих імпульсів C' і при наявності на інформаційних входах сигналів $J' = K' = 1$ тригер перемикається в протилежний стан ($Q^{n+1} = \overline{Q^n}$), тобто працює в рахунковому режимі. В інших випадках він функціонує як синхронний RS -тригер із прямими входами (табл. 5.2), якщо вхід J вважати входом S , а вхід K — входом R . Тригер JK D -типу може працювати і як D -тригер, якщо виключити випадки рівнозначності сигналів J' і K' . Іншими словами, він є універсальним. На рис. 5.11 показані схеми використання його в режимах:

- а) синхронного RS -тригера (за умови виключення комбінації $J' = K' = 1$);
- б) T -тригера;
- в) D -тригера.

Рис. 5.11. JK -тригери

На практиці найбільш широке застосування одержали двоступінчаті JK -тригери. Їх структурна схема й умовне графічне позначення зображені на рис. 5.12.

За складом і зв'язками схема аналогічна двоступінчатому RS -тригеру (рис. 5.8). Додані лише зворотні зв'язки з виходів другого ступеня на входи першого, які виключають невизначений стан при комбінації $J' = K' = 1$. Для

організації зворотних зв'язків тригер першого ступеня містить по два входи S та R , об'єднаних кон'юнктором.

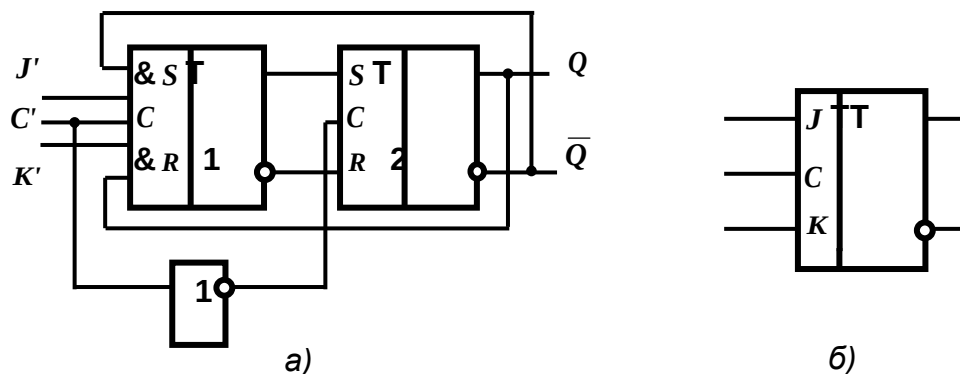


Рис. 5.12. Схема (а) та умовне графічне позначення (б) двоступінчатого JK -тригера

У цілому організація роботи двоступінчатого JK -тригера така ж, як і в усіх раніше розглянутих двоступінчатих структурах – почерговий перехід ступенів з режиму прийому інформації в режим збереження обумовлений сигналом синхронізації. Однак перехід тригера в один або інший стан залежить не тільки від інформаційних сигналів J' і K' , але й від стану його в попередньому такті. Покажемо це на прикладі роботи, коли $J' = K' = 1$. Якщо до надходження чергового сигналу $C' = 1$ на виході $Q^n = 0$, то на обох входах S першого тригера буде одиничний рівень, а на одному з входів R – нульовий. Тому у момент надходження сигналу $C' = 1$ перший тригер установиться в одиничний стан, а після закінчення його ($C' = 0$) – також другий тригер ($Q^{n+1} = 1$). У силу осьової симетрії схеми очевидно, якщо $Q^n = 1$, то $Q^{n+1} = 0$. Отже, має місце рахунковий режим: $Q^{n+1} = \overline{Q^n}$.

Часові діаграми, які ілюструють роботу JK -тригера, наведені на рис. 5.13, а варіант реалізації (ІС 133ТВ1) – на рис. 5.14. Він має по три входи J і K , зв'язані операцією I , а також асинхронні входи $\overline{S}$ і $\overline{R}$ для установки тригера в одиничні і нульові стани незалежно від сигналів на інформаційних і синхронізуючих входах. Тригер першого ступеня (ведучий тригер) утворений елементами **D1** і **D2**, а другого ступеня (ведений) – елементами **D3...D8**. Вузол керування веденого RS -тригера (**D3...D6**) крім звичайної ролі виконує також роль інвертора сигналу C' для другого ступеня (**D5, D6**).

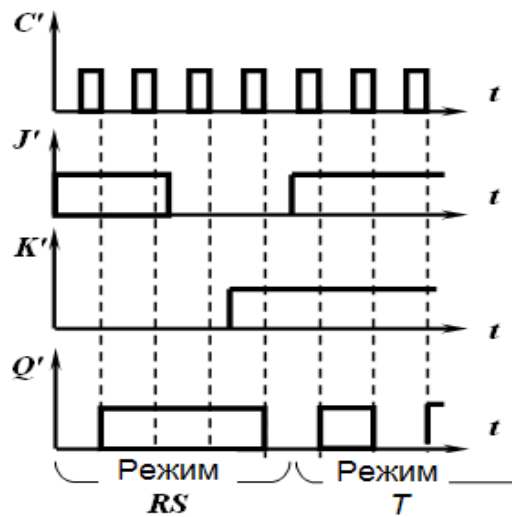


Рис. 5.13. Часові діаграми JK-тригера, (наведеного на рис. 5.12)

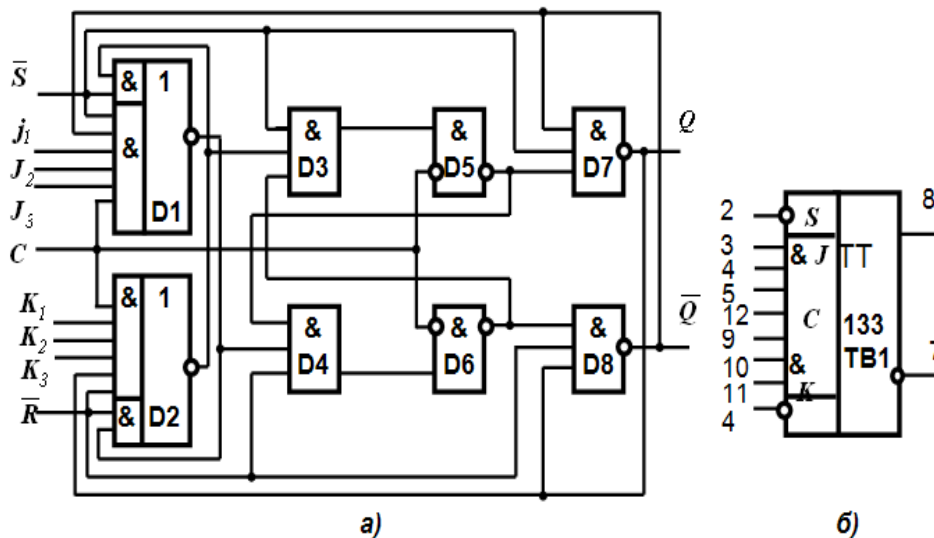


Рис. 5.14. Функціональна схема (а) та умовне графічне позначення (б) JK-тригера 133TB1

Багато серій цифрових ІС поряд із двоступінчастими тригерами містять одноступінчасті тригери з динамічною синхронізацією. З погляду функціонування між ними відмінностей немає. Але цим самим забезпечується різноманітність деяких споживчих властивостей тригерів. Наприклад, тригер, який входить до складу ІС **K155TB15** (рис. 5.15), має один інформаційний вхід прямий (J), а інший – інверсний ($\bar{K}$), вхід синхронізації динамічний. Така особливість дозволяє перетворювати його в D -тригер із прямими входами шляхом простого з'єднання виводів J і $\bar{K}$.

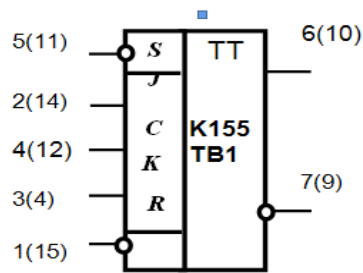


Рис. 5.15. Тригер K155TB1

Швидкодія *JK*-тригерів (у рахунковому режимі) залежить від конкретної схеми і звичайно трохи уступає швидкодії *D*-тригерів.

Якщо ввести у вихідні кола тригера двонаправлені ключі, то можна отримати третій (високоімпедансний) стан по виходах. Введення ж ключів у внутрішні інформаційні кола дозволяє спростити реалізацію складних тригерів.

На рис. 5.16 показана структура, яка має властивості *D*-тригера зі статичним входом синхронізації, що знайшла застосування в двоступінчатих тригерах різних типів.

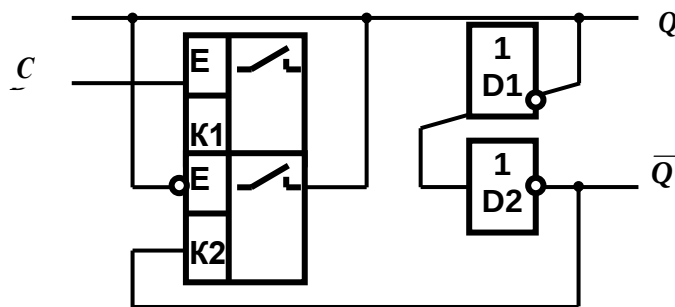


Рис. 5.15. *D*-тригер на основі двонаправлених ключів

Чарунка пам'яті в ній утворена за допомогою двох охоплених взаємними зворотними зв'язками інверторів (**D1**, **D2**). Причому одне коло зворотного зв'язку (з виходу **D2** на вхід **D1**) комутується двонаправленим ключем **K2**. Замикання кола відбувається при подачі сигналу $C' = 0$. Ключ **K1** має прямий вхід дозволу **E**, тому при $C' = 0$ він розімкнутий, і чарунка пам'яті від інформаційного входу **D** відключена. Тригер працює у режимі збереження.

Якщо на вхід **C** подати $C' = 1$, ключі перемикаються, й інформація з входу **D** надходить на інвертори, а потім на виходи **Q** і $\bar{Q}$. Наступний сигнал $C' = 0$ забезпечує фіксацію стану інверторів.

Якщо на вхід **C** подати $C' = 1$, ключі перемикають

На рис. 5.16 наведений приклад схеми *JK*-тригера, у ступенях якого використовується описаний вище *D*-тригер.

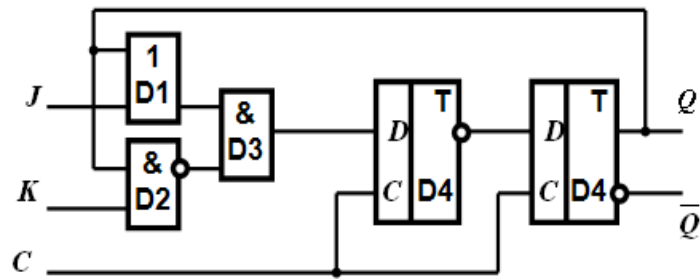


Рис. 5.16. *JK*-тригер, (схема якого зображена на рис. 5.15), на основі двох *D*-тригерів

До характерних особливостей тригерів із двонаправленими ключами варто віднести критичність їх роботи до тривалості фронту (спаду) імпульсу C' . Дуже повільна зміна сигналу C' може стати причиною того, що ключі **K1** і **K2** (рис. 5.15) будуть замкнені протягом часу, порівняного з часом затримки перемикавання тригера.

У результаті після завершення перехідних процесів тригер опиниться в невизначеному стані.

Зазначену особливість необхідно враховувати, однак практичні умови виникнення аномально поганих синхронізуючих імпульсів досить рідкісні. Наприклад, для тригера **K561ТВ1** припустима тривалість фронту і спаду імпульсу синхронізації складає 5 мкс, у той час як тривалість перепадів на виходах будь-якої ІС серії **K561**, що може служити джерелом сигналу C' , не перевищує декількох десятків наносекунд.

5.2.3. *D*-тригери

D-тригери мають інформаційний вхід D і вхід синхронізації C . Функціонують *D*-тригери відповідно до таблиці станів (табл. 5.5). З подачею синхронізуючого сигналу C' тригер встановлюється в стан $Q^{n+1} = D'$ і зберігає його до надходження наступного сигналу C' .

Таблиця 5.5

Таблиця станів *D*-тригера

D'	Q^{n+1}
0	0
1	1

Іншими словами, тригер здійснює затримку двійкової інформації, яка надходить на вхід D , на період проходження синхронізуючого сигналу C' . Тому його називають ще тригером затримки. Свою назву він отримав від слова **Delay** – затримувати [8].

У мікроелектронній схемотехніці знаходять застосування D -тригери як зі статичним, так і динамічним входом синхронізації.

На рис. 5.17 наведена найпростіша схема D -тригера з прямим статичним входом C .

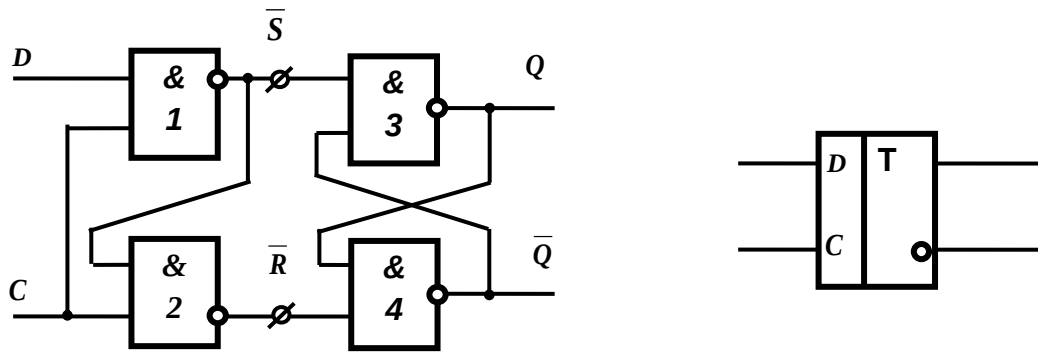


Рис. 5.17. D -тригер із статичним входом синхронізації

При $C' = 0$ логічні елементи ЛЕ1 та ЛЕ2 вимкнені ($S' = R' = 1$). Тому асинхронний RS -тригер (ЛЕ3 та ЛЕ4), який є чарункою пам'яті з інверсними входами, працює в режимі збереження. Сигнал $C' = 1$ дозволяє передачу інформації з входу D у чарунку пам'яті. Відбувається це так: якщо $D' = 0$, то $S' = 1, R' = 0$ і $Q^{n+1} = 0$; якщо $D' = 1$, то $S' = 0, R' = 1$ і $Q^{n+1} = 1$.

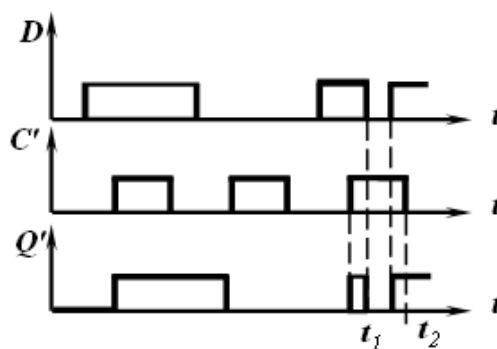


Рис. 5.18. Часові діаграми D -тригера, (зображеного на рис. 5.17)

Роботу тригера ілюструє рис. 5.18. Помітимо, що зміни інформації на вході D під час дії синхронізуючого сигналу передаються на вихід тригера ($t_1 < t < t_2$). Звичайно це небажано. Тому, коли необхідно фіксувати інформа-

цію тільки у визначені моменти часу, застосовують або дуже короткі імпульси C' , або D-тригер з динамічним входом C .

Найбільш розповсюдженою схемою інтегрального D-тригера з динамічною синхронізацією є схема на трьох асинхронних RS-тригерах (рис. 5.19).

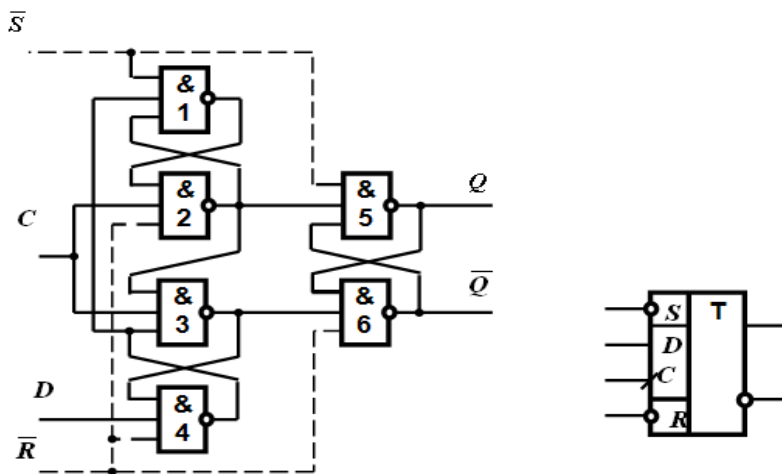


Рис. 5.19. D-тригер з динамічним входом

Тригер на ЛЕ5 та ЛЕ6 служить чарункою пам'яті, а два інших тригери (ЛЕ1 та ЛЕ2; ЛЕ3 та ЛЕ4) є допоміжними, які утворюють вузол керування. Поряд із входами D та C можуть бути також входи асинхронної установки в одиничний і нульовий стани (S та R). Асинхронність цих входів на умовному графічному позначенні підкреслюється відділенням їх від області мітки C за допомогою перегородок. Принцип дії тригера пояснюється часовими діаграмами напруг на виходах логічних елементів (рис. 5.20). Діаграми побудовані з урахуванням часу затримки перемикавання ЛЕ $t_{zm\_сер}$, що дозволяє підвищити наочність причинно-наслідкових зв'язків у тригері.

Нехай у вихідному стані ($t < t_1$) $D' = 1$, $C' = 0$. Сигнал $C' = 0$ подається на входи ЛЕ2 та ЛЕ3, тому вони вимкнені ($y_2 = y_3 = 1$), чарунка пам'яті (ЛЕ5, ЛЕ6) працює в режимі збереження. Для однозначності прийнято $Q' = 0$. На входах ЛЕ4 дві одиниці, виходить $y_4 = 0$, а пов'язаний з його виходом ЛЕ1 вимкнений ($y_1 = 1$).

З надходженням синхронізуючого сигналу ($t = t_1$) протягом найближчого інтервалу $t_{zm\_сер}$, вмикається ЛЕ2, тому що на його входах діють одиничні рівні. Елемент ЛЕ3 свій стан не змінює.

Протягом другого інтервалу дискретизації можливе перемикавання ЛЕ1, ЛЕ3, ЛЕ5, на входи яких надійшов сигнал $y_2 = 0$. Оскільки ЛЕ1 та ЛЕ3 були вимкнені, їхній стан не змінюється. Вимкнеться лише ЛЕ5 ($Q' = 1$).

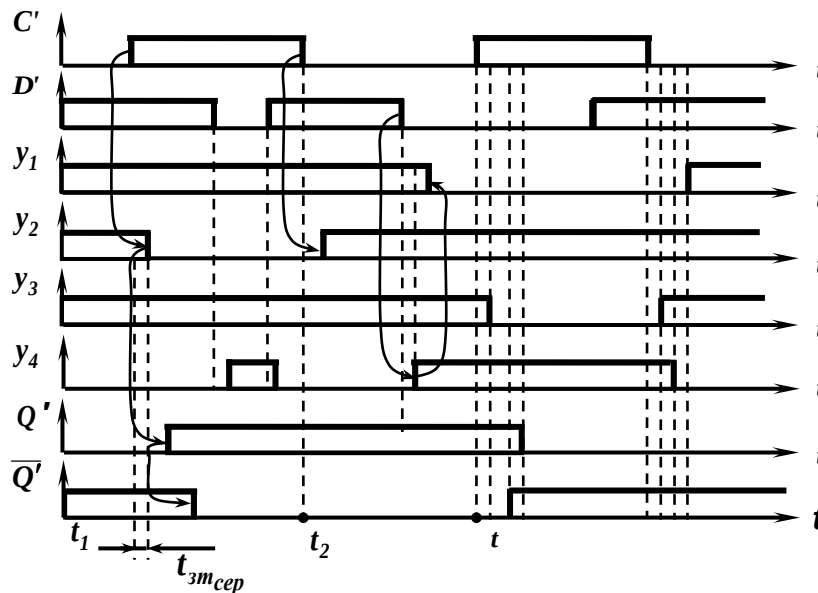


Рис. 5.20. Часові діаграми D-тригера, (зображеного на рис. 5.19), з урахуванням затримки перемикання логічних елементів

Протягом третього інтервалу дискретизації вмикається лише ЛЕ6 ($\overline{Q}' = 0$). На цьому перехідні процеси в тригері закінчуються.

Аналогічним чином будуються діаграми при інших комбінаціях зовнішніх сигналів C' та D' .

Опускаючи опис такої побудови, проведемо аналіз отриманих результатів. З рис. 5.20 видно, що, по-перше, тригер дійсно працює відповідно до функції переходу $Q^{n+1} = D'$, по-друге, він перемикається лише в моменти надходження логічного перепаду $0 \rightarrow 1$ на виході C . Причому, зміна інформації на вході D як під час дії синхронізуючого сигналу ($C' = 1$), так і під час паузи ($C' = 0$) не приводить до зміни стану тригера.

Отже, вхід синхронізації прямий динамічний. Завдяки цьому тригер фіксує вхідну інформацію тільки по фронту імпульсу C' (рис.5.21).

Динамічність синхронізації дозволяє використовувати D-тригер як T-тригер. Рахунковий режим можна одержати шляхом підключення входу D до виходу Q (рис. 5.22). Дійсно, до моменту надходження чергового синхронізуючого сигналу ($C' = 1$) на інформаційному вході $D' = Q^n$, отже в результаті впливу сигналу C' стан тригера стане $Q^{n+1} = D' = \overline{Q^n}$. Вираз $Q^{n+1} = \overline{Q^n}$ є функцією переходів T -тригера.

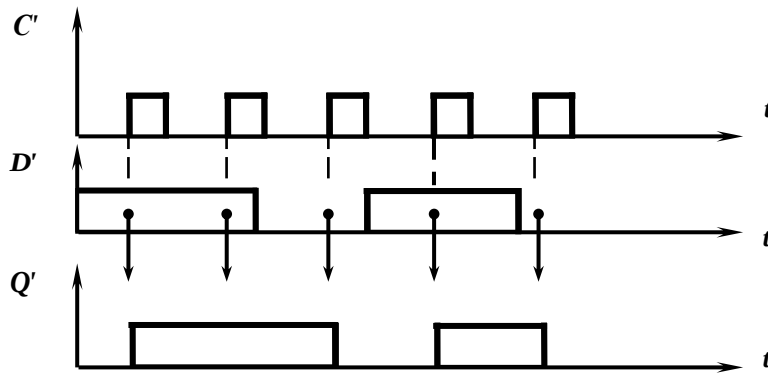


Рис. 5.21. Часові діаграми D-тригера з динамічним входом синхронізації

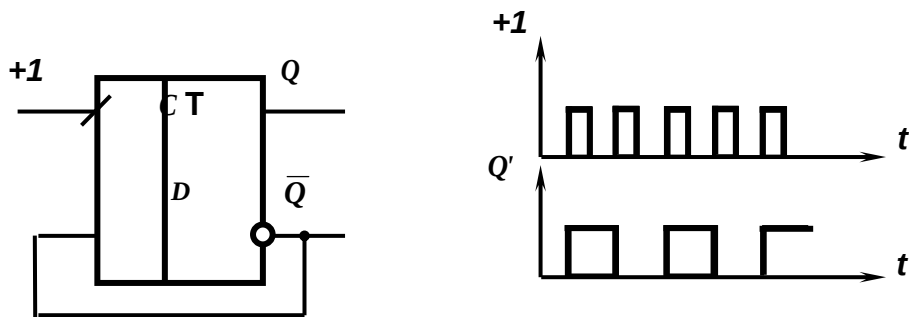


Рис. 5.22. Робота D-тригера у рахунковому режимі

Швидкодія тригера в рахунковому режимі визначається мінімальним періодом проходження рахункових імпульсів, що у свою чергу відповідає логічній глибині, тобто, найбільш довгому шляху проходження сигналу.

З рис. 5.21 і рис. 5.22 випливає, що логічна глибина (з урахуванням зв'язку D з $\bar{Q}$) дорівнює п'яти: ЛЕ2 – ЛЕ5 – ЛЕ6 – ЛЕ4 – ЛЕ1 і

$$F_{max} = \frac{1}{5 t_{zm_{cep}}}.$$

Мінімальна тривалість імпульсів, які рахуються, дорівнює мінімальній тривалості керуючих сигналів на входах чарунки пам'яті (асинхронного **RS**-тригера) і складає $t_{i_{min}} = 2 t_{zm_{cep}}$.

Тригери типу **D** можуть виконуватися також двоступінчастими. Для цього використовуються **D**-тригери зі статичною синхронізацією (рис. 5.23).

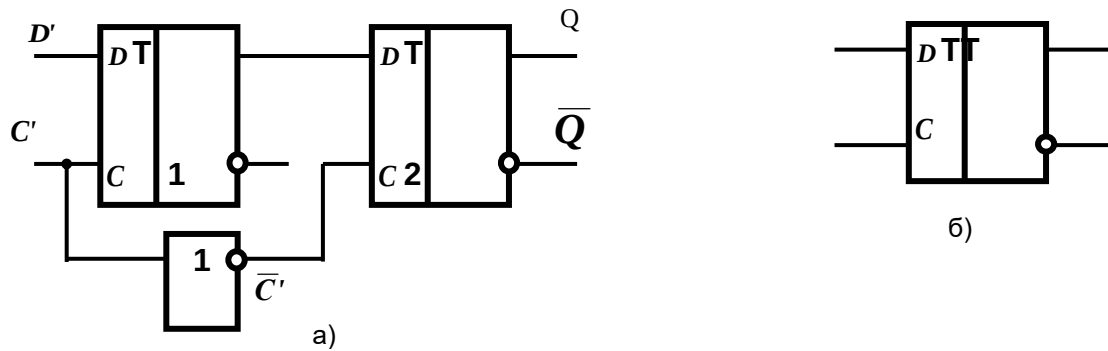


Рис. 5.23. Схема (а) та умовне графічне позначення (б) двоступінчатого D-тригера

Порядок роботи двоступінчатого **D**-тригера такий же, як двоступінчатого **RS**-тригера. Під час дії сигналу $C' = 1$ інформація з входу **D** записується в тригер першого ступеня, а при $C' = 0$ – другого ступеня.

При цьому другий ступінь фіксує лише той стан першого ступеня, який мав місце в момент зрізу імпульсу синхронізації. Інакше кажучи, двоступінчатість структури додає їй властивість динамічної синхронізації.

Помітимо, це правило є загальним для двоступінчатих тригерів інших типів.

5.2.4. T-тригери

T-тригер (рахунковий тригер) має всього один інформаційний вхід, який позначається +1. Зміна станів відбувається при кожному надходженні на нього одиничної інформації (1 або 0), тобто має місце логіка $Q^{n+1} = \overline{Q^n}$. Якщо вхід тригера прямий, зміна стану відбувається при надходженні перепаду $0 \rightarrow 1$, якщо інверсний – то за перепадом $1 \rightarrow 0$.

Умовне графічне позначення T-тригера з прямим рахунковим входом і часові діаграми, які ілюструють його роботу, показані на рис. 5.24.

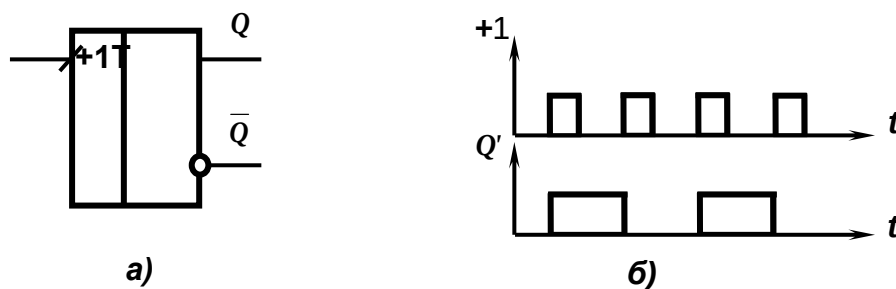


Рис. 5.24. Умовне графічне позначення (а) та часові діаграми (б) T-тригера

Перемикання за кожним вхідним імпульсом можна трактувати як підрахунок їх "сума за модулем 2" (після кожного непарного імпульсу на виході $Q' = 1$, а після парного – $Q' = 0$). Така властивість тригера використовується в лічильниках і накопичувальних суматорах.

В інтегральному виконанні у вигляді самостійних виробів T -тригери не виготовляються, тому що їх функції можуть виконувати тригери D - та JK -типів. Рахунковий режим у них виходить шляхом комутації виводів або подачі визначених сигналів.

5.3. Синтез тригерів із заданими умовами функціонування

Кінцевою метою синтезу тригера є отримання його принципової схеми, яка реалізує задані умови функціонування. При цьому необхідно намагатися отримати максимальну простоту схеми, що синтезується.

Припустимо, наприклад, що на основі RS -тригера з інверсними входами необхідно побудувати DV тригер на логічних елементах **НІ-ЧИ**. Вхід V є таким, що дозволяє функціонування тригера, а саме: при $V = 1$ тригер реагує на сигнали по входу D , а при $V = 0$ – ні.

Спочатку необхідно формалізувати умови функціонування тригера, синтезується у вигляді повної таблиці переходів (табл.5.6). Приймаємо до уваги той факт, що вхід V належить до управляючих входів і сигнал на цьому вході дозволяє тригеру здійснювати прийом інформації.

Таблиця 5.6

Таблиця переходів DV -тригера

t			t+1
V	D	Q	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

З повної таблиці переходів (табл. 5.6) визначаються всі припустимі комбінації сигналів на входах D та V , які забезпечують всі чотири типи переходів $Q(t) \rightarrow Q(t+1)$ (рис. 5.25).

"0 → 0"				"0 → 1"			
V	0	0	1	V	1		
D	0	1	0	D	1		

"1 → 0"				"1 → 1"			
V	1			V	0	0	1
D	0			D	0	1	1

Рис. 5.25. Забезпечення переходів тригера $Q(t) \rightarrow Q(t+1)$

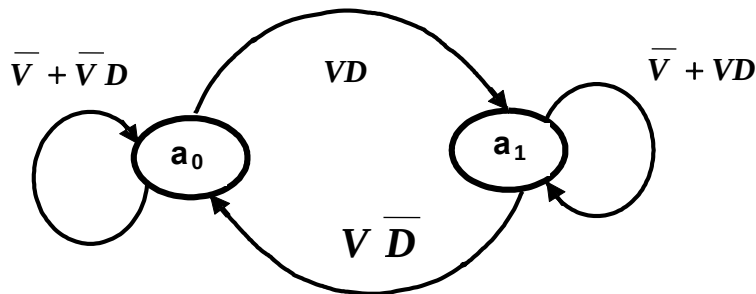


Рис. 5.26. Граф переходів DV -тригера

Складається граф переходів DV -тригера (рис. 5.26). Для цього позначається нульовий стан тригера через a_0 , а одиничний стан тригера через a_1 . Для дуги, що виходить із a_0 і входить в a_0 шляхом врахування диз'юнкції всіх кон'юнкцій сигналів на входах D та V , які забезпечують даний перехід (рис. 5.26.) отримується вираз $\bar{V} \bar{D} \vee \bar{V} D \vee V \bar{D}$, який за допомогою аксіом алгебри логіки спрощується до $\bar{V} \vee V \bar{D}$. Аналогічним чином для дуги із a_0 в a_1 отримується VD , для дуги із a_1 в a_0 отримується $V \bar{D}$, та для дуги з a_1 до a_1 - $\bar{V} \bar{D} \vee \bar{V} D \vee VD = \bar{V} \vee VD$.

Складається повна таблиця переходів DV -тригера (табл. 5.7) з врахуванням того, що як комірка запам'ятовування використовується RS -тригер з інверсними входами, звертаючи увагу на можливість невизначених станів входів RS -тригера при забезпеченні заданих переходів тригера для подальшої мінімізації функцій збудження цих входів.

За допомогою карт Карно визначаються функції збудження $R' S'$ - тригера з врахуванням вимог завдання щодо побудови схеми тригера на елементах ЧИ-НІ (рис. 5.27).

Повна таблиця переходів DV- тригера на базі RS-тригера
з інверсними входами

t			$t + 1$	$\bar{R}$	$\bar{S}$
V	D	Q	Q		
0	0	0	0	~	1
0	0	1	1	1	~
0	1	0	0	~	1
0	1	1	1	1	~
1	0	0	0	~	1
1	0	1	0	0	1
1	1	0	1	1	0
1	1	1	1	1	~

V, D	00	01	11	10
Q				
0	~	~	1	~
1	1	1	1	0

 R'

V, D	00	01	11	11
Q				
0	1	1	0	1
1	~	~	~	1

 S'

$$R' = \bar{V} \vee D$$

$$S' = \bar{V} \vee \bar{D}$$

Рис. 5.27. Визначення функцій збудження RS-тригера

За отриманими логічними функціями збудження будується функціональна схема DV- тригера. Для цього за допомогою аксіом алгебри логіки представляємо логічні функції R' та S' в заданому структурному базисі.

$$R' = \bar{V} \vee D = \overline{\bar{V} \vee \bar{D}}, \quad S' = \bar{V} \vee \bar{D} = \overline{\bar{V} \vee D}.$$

Будуємо функціональну схему DV- тригера (рис.5.28) в заданому базисі.

Сучасні програмні засоби схемотехнічного моделювання аналогових та цифрових пристроїв дозволяють імітувати роботу розробленого тригера.

Так, на рис. 5.29 за допомогою програмного пакета моделювання *Electronics Workbench* побудована синтезована схем DV-тригера для подальшої експериментальної перевірки умов його функціонування.

На підставі аналізу функціональної схеми синтезованого DV-тригера в заданому базисі можна оцінити деякі важливі часові характеристики його роботи, наприклад, швидкодію тригера.

Час затримки перемикання тригера $t_{затр.триг.}$ складається з затримки розповсюдження сигналів у схемі управління та затримки перемикання безпосередньо комірки пам'яті.

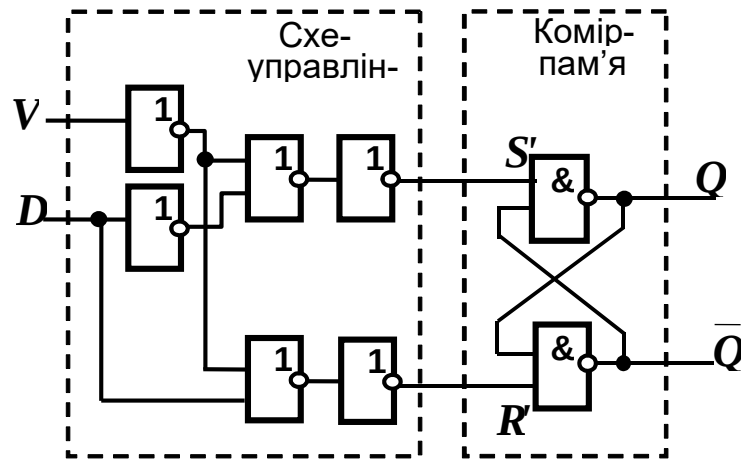


Рис. 5.28. Функціональна схема DV -тригера

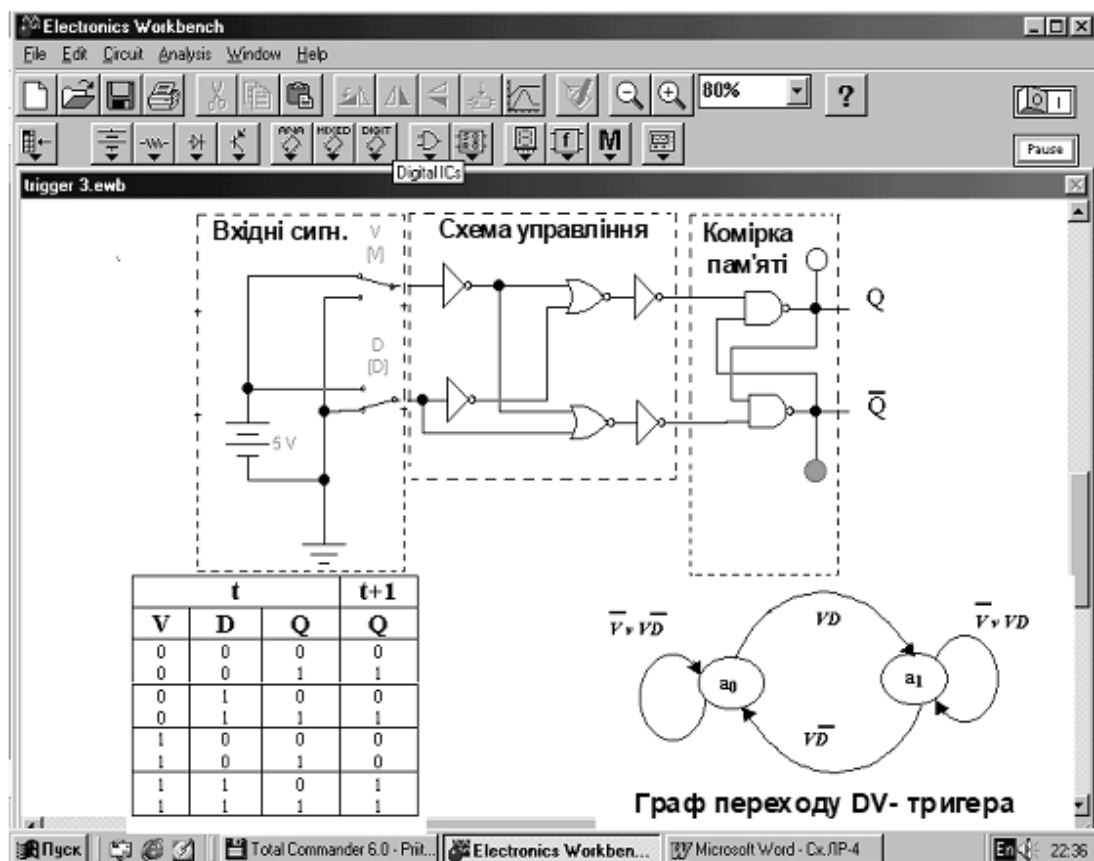


Рис. 5.29. Схема дослідження функціонування DV -тригера

Оскільки вхідні сигнали послідовно проходять через три логічних елемента схеми управління та два логічних елемента комірки пам'яті, то правомірно вважа-

ти, що загальний час затримки перевимикання **DV**-тригера буде дорівнювати часу затримки сигналу в п'яти логічних елементах. $t_{затр.триг.} = 5t_{затр.л.е.}$. Якщо прийняти до уваги, що затримка розповсюдження сигналу в одному логічному елементі приблизно рівняється 20 нс , то $t_{затр.триг.} \approx 100 \text{ нс}$. Звідси максимальну частоту роботи частоту DV-тригера можна оцінити наступним чином:

$$f = \frac{1}{t_{затр.триг.}} = \frac{1}{100 \cdot 10^{-9}} = 10^7 \text{ Гц} = 10 \text{ мГц}.$$

КОНТРОЛЬНІ ПИТАННЯ

1. Дати визначення тригеру.
2. За якими ознаками прийнято класифікувати тригери?
3. Навести умовне графічне зображення та функціональну схему **RS**-тригера з прямим (інверсним) входом.
4. Знайти три помилки в таблиці опису роботи **RS**-тригера з прямими входами.

n	$n+1$			
Q^n	S^{n+1}	R^{n+1}	Q^{n+1}	$\overline{Q}^{n+1}$
0	0	0	0	1
1	0	0	1	0
0	1	0	1	0
1	1	0	1	0
1	0	1	0	1
0	0	1	0	1
0	1	1	н/в	н/в
1	1	1	н/в	н/в

5. Пояснити принцип роботи 2-ступінчатого тригера.
6. Навести таблицю станів **D**- та **T**-тригерів.
7. Дати визначення **JK**-тригерів.
8. У чому полягає універсальний характер **JK**-тригерів?
9. Пояснити принцип роботи тригера Шмітта.
10. Синтезувати **TV**-тригер на базі **RS**-тригера з прямими входами на елементах **I** - **III**. Визначити швидкість його роботи.

6. СХЕМОТЕХНІКА ЦИФРОВИХ ВУЗЛІВ

6.1. Регістри

Призначення регістрів – прийом, збереження і видача двійкової кодової інформації (двійкових чисел, слів, операндів). Вони використовуються як безадресні запам'ятовуючі пристрої, перетворювачі і генератори кодів, пристрої часової затримки цифрової інформації, дільники частоти тощо.

Для реєстрації n -розрядного числа необхідно мати в складі регістра n -тригерів. Кожний з них запам'ятовує один біт інформації. Тригери між собою можуть бути пов'язані колами керування і перенесення.

Поняття “вага розряду” до регістра, на відміну від лічильника, не використовується, оскільки вагова залежність між змінними записаного слова може бути відсутня. З цієї причини на умовних графічних зображеннях регістрів мітки інформаційних входів і виходів нумеруються в природному порядку.

Усі регістри залежно від способу прийому і видачі інформації поділяються на такі типи:

- з паралельним прийомом і видачею;
- з послідовним прийомом і видачею;
- з комбінованим прийомом і видачею.

Регістри з паралельним прийомом і видачею називають регістрами пам'яті, а всі інші – регістрами зсуву.

За напрямком зсуву (передачі) інформації розрізняють односпрямовані та реверсивні регістри.

Регістри пам'яті становлять набір синхронних тригерів з незалежними інформаційними входами. Тригери поєднуються лише керуючими сигналами (синхронізації, дозволу запису, видачі, високого імпедансу тощо). Введення (запис) і виведення (зчитування) інформації здійснюється одночасно у всіх розрядах паралельним кодом. Введення забезпечується синхронізуючим імпульсом. З приходом чергового такого імпульсу відбувається відновлення записаної інформації.

Якщо вхідна інформація парафазна, тобто кожна змінна надходить одночасно в прямому й інверсному вигляді, використовуються **RS** - або **JK**-тригери (рис. 6.1 а)), якщо однофазна – **D**-тригери (рис. 6.1 б)).

Зчитування може відбуватися в прямому і зворотному кодах, тобто з прямих й інверсних виходів тригерів.

У регістрах, призначених для роботи на інформаційну магістраль, як вихідні (буферні) каскади використовуються елементи з трьома станами, керовані

сигналом дозволу зчитування ERD . Типовим прикладом може служити мікросхема **133ИР15** (рис. 6.2).

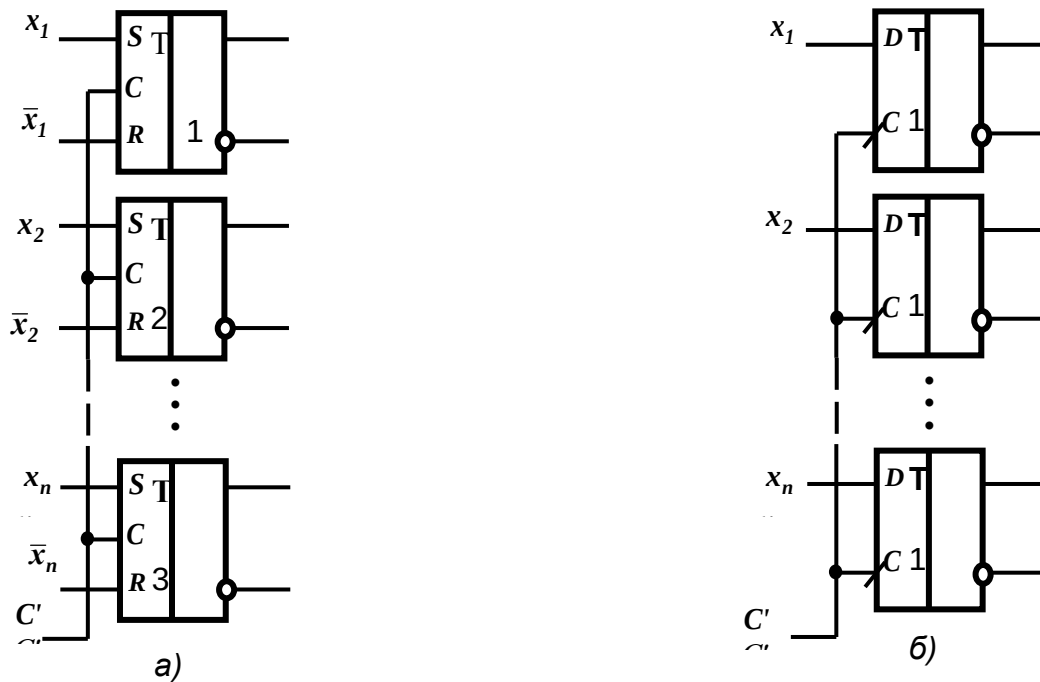


Рис. 6.1. Регістри пам'яті на основі синхронних RS -(а) та D -тригерів (б)

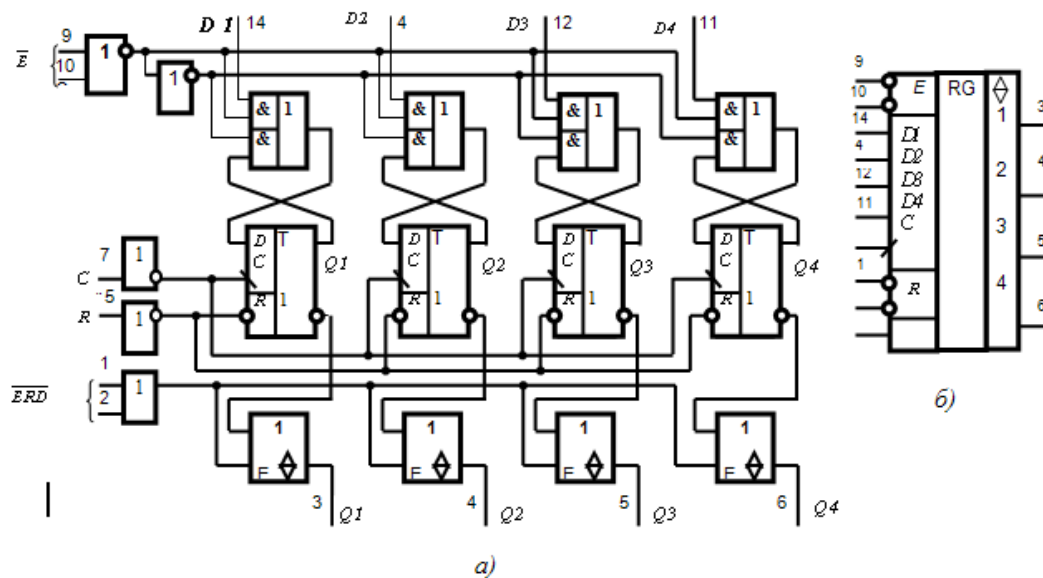


Рис. 6.2.Схема (а) та УГП (б) регістр пам'яті 133ИР15

Це чотирьохрозрядний регістр пам'яті з однофазними інформаційними входами $D1, D1, D1, D1$, запис через які здійснюється за перепадом $0 \rightarrow 1$ синхронізуючого сигналу C' , якщо на обох входах дозволу E низька напруга. Подача одиничного рівня хоча б на один вхід $\bar{E}$ переводить регістр у режим

збереження. Для зчитування інформації, яка зберігається в регістрі, на обидва входи $\overline{ERD}$ подається низька напруга, в іншому випадку виходи відключаються, що відповідає високоімпедансному стану.

Нарощування розрядності регістра досягається паралельним з'єднанням керуючих входів декількох мікросхем.

6.2. Регістри зсуву

Регістри зсуву мають властивість зсувати (переміщувати) записану в них інформацію вліво (убік старших розрядів) або вправо (убік молодших розрядів) на задану кількість розрядів. Тому тригери в них з'єднуються послідовно. Суть зсуву полягає в тому, що з приходом синхронізуючого імпульсу відбувається перезапис вмісту тригера кожного розряду в сусідній розряд. Кількість зсувів визначається кількістю синхронізуючих імпульсів.

Реальний напрямок зсуву в регістрі визначається напрямком міжтригерних зв'язків. Однак з погляду зсуву щодо числової розрядної сітки той самий регістр, що має односпрямовані міжтригерні зв'язки, може бути регістром зі зсувом вправо і регістром зі зсувом вліво залежно від того, у якому порядку розміщена інформація.

Будуються регістри на тригерах з динамічним входом синхронізації або з внутрішньою затримкою. Ця умова є обов'язковою. В іншому випадку при надходженні синхронізуючого сигналу інформація може пересунутися не на один розряд, а на стільки, скільки разів встигнуть перемкнутись тригери за час дії цього сигналу.

На рис. 6.3 показана схема, умовне графічне позначення і часові діаграми роботи трирозрядного регістра зсуву на D -тригерах. Він реалізує спосіб послідовного прийому та послідовної видачі. Передбачається, що вхідна інформація надходить молодшими розрядами вперед, тобто вхідним тригером є тригер старшого розряду, а вміст регістра зчитується з виходу $Q1$ тригера молодшого розряду.

За кожним фронтом імпульсу C' двійкова змінна, яка зберігається в i -му тригері, переписується в $(i - 1)$ -й тригер. Оскільки входи C динамічні, наступні зміни рівнів напруги на входах D у межах даного сигналу $C' = 1$ не приводять до нових перемикань тригерів. Таким чином, при надходженні одного імпульсу C' інформація в регістрі зрушується на один розряд вправо: змінна "х" із входу $0 \rightarrow$ записується

в тригер $D3$, його попередній вміст переміщується в тригер $D2$, а попередній вміст $D2$ переміщується у $D1$.

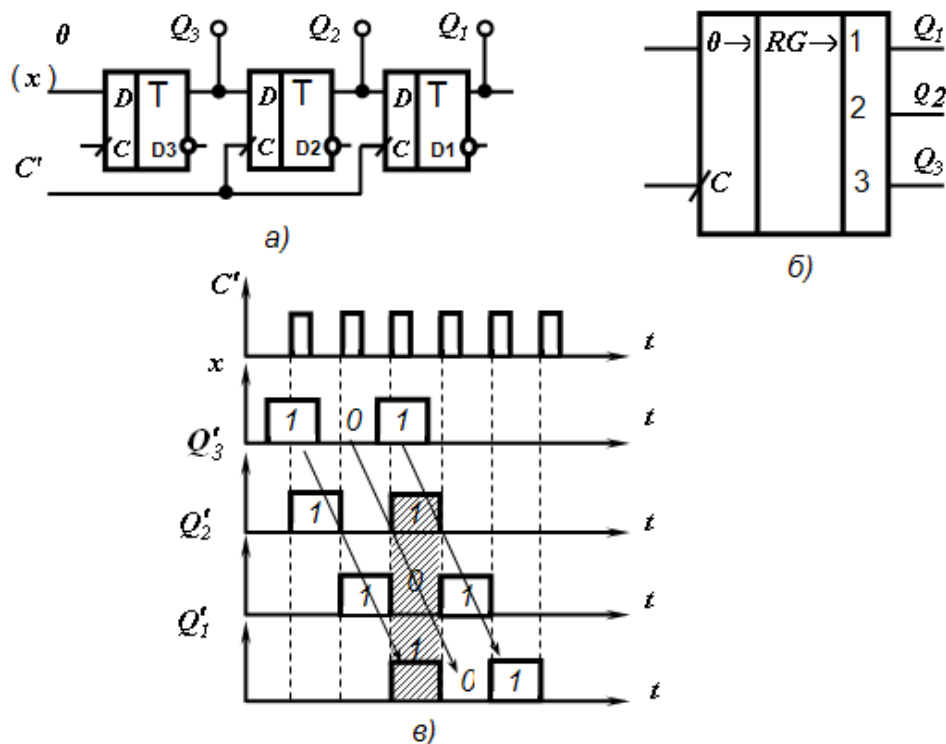


Рис. 6.3. Схема (а), умовне графічне позначення (б) та часові діаграми (в) трирозрядного регістра зсуву на D-тригерах

Для повного завантаження регістра новою інформацією необхідно подати три (за кількістю розрядів регістра) синхронізуючі імпульси. Наступні імпульси C' забезпечують видачу записаної інформації з виходу тригера $D1$.

Якщо імпульси синхронізації надходять безупинно з періодом T_C , то вихідний сигнал Q_1' повторює вхідний “х” із затримкою на nT_C (тут $n = 3$), і регістр виступає в ролі пристрою дискретної затримки цифрової інформації.

Виходи Q тригерів зазвичай мають зовнішні виводи (на рис. 6.3 а)), з яких можна знімати записану інформацію. Інакше кажучи, такий регістр може служити перетворювачем послідовного коду в паралельний.

Щоб змінювати напрямок зрушення, не змінюючи послідовності надходження вхідних інформаційних сигналів, використовують реверсивні регістри. Принцип їх побудови ґрунтується на комутації входу i -го тригера до виходу $(i - 1)$ -го або $(i + 1)$ -го тригера (рис. 6.4 а). Тут при сигналі керування $E' = 1$ підключається $(i - 1)$ -й тригер, а при $E' = 0$ – $(i + 1)$ -й тригер, відповідно встановлюється режим зсуву вліво або вправо.

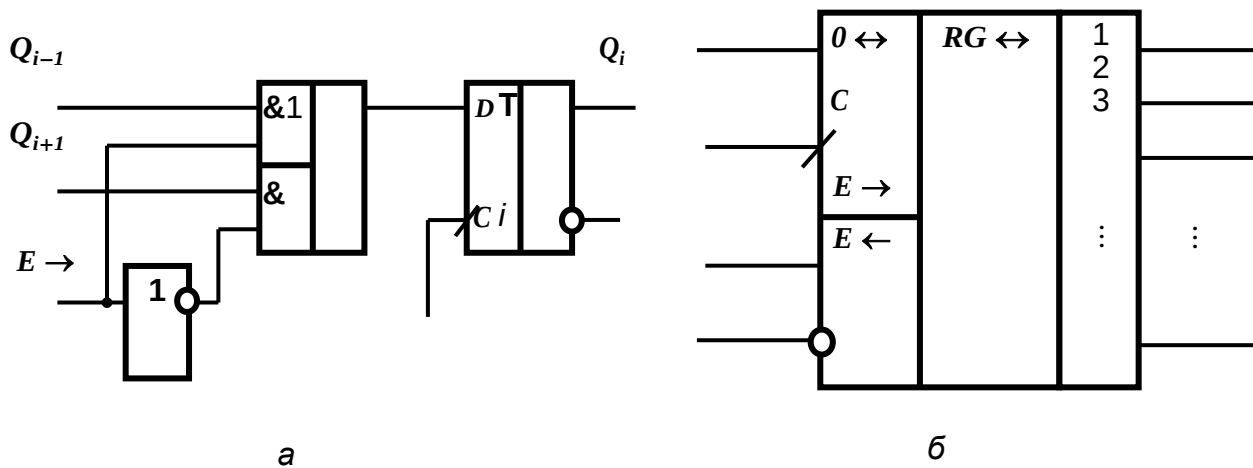


Рис. 6.4. Міжтригерні зв'язки (а) та УГП (б) реверсивного регістра

Функціональні можливості регістрів значно розширюються при введенні режиму паралельного запису інформації. Такі регістри додатково можуть виконувати функції регістрів пам'яті, а також перетворювачів паралельного коду в послідовний. На рис. 6.5 показана схема такого регістра. Він зібраний на **JK**-тригерах з асинхронними входами S та R . Паралельний запис двійкової інформації здійснюється за допомогою комутаторів, керованих вхідними змінними D'_i . Залежно від значення D'_i команда запису C'_2 проходить на вхід S або вхід R i -го тригера. У режимі зсуву виконавчим сигналом служить C'_1 .

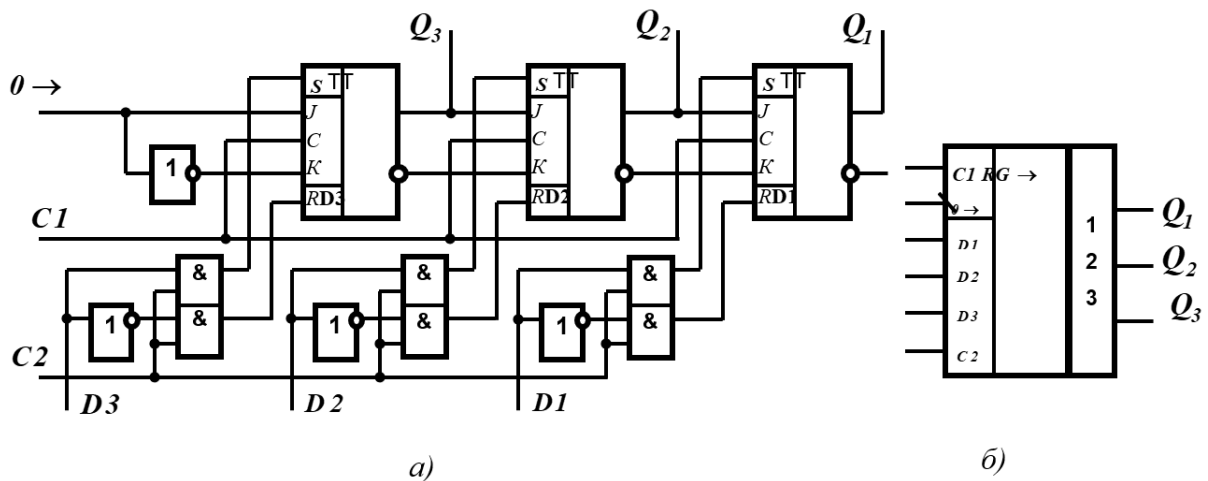


Рис. 6.5. Схема (а) та УГП (б) регістра зсуву з можливістю паралельного запису інформації

Одночасне надходження синхронізуючих сигналів C'_1 та C'_2 неприпустимо. Для того, щоб сигнали C'_1 та C'_2 не впливали на регістр при їхньому ви-

падковому збігу, тобто виключалося накладення різних режимів роботи, у багатьох практичних схемах вводиться керування режимами [2].

Наприклад, у регістрі 133ІР15, схема якого зображена на рис. 6.6 а), режими встановлює сигнал E' . При $E' = 1$ інформаційні входи тригерів, які працюють завдяки інверторам ЛЕ6...ЛЕ10 у режимі D -тригера, підключені до входніх шин паралельного запису $D1...D4$, а входи синхронізації тригерів – до шини сигналу C_2' . Якщо $E' = 0$, інформаційні входи тригерів підключені до виходів попередніх тригерів, а зовнішня інформація сприймається по входу $0 \rightarrow$ у вигляді послідовного коду. При цьому робота тригерів синхронізується імпульсами C_1' . Сигнали на інших входах регістра можуть бути будь-якими.

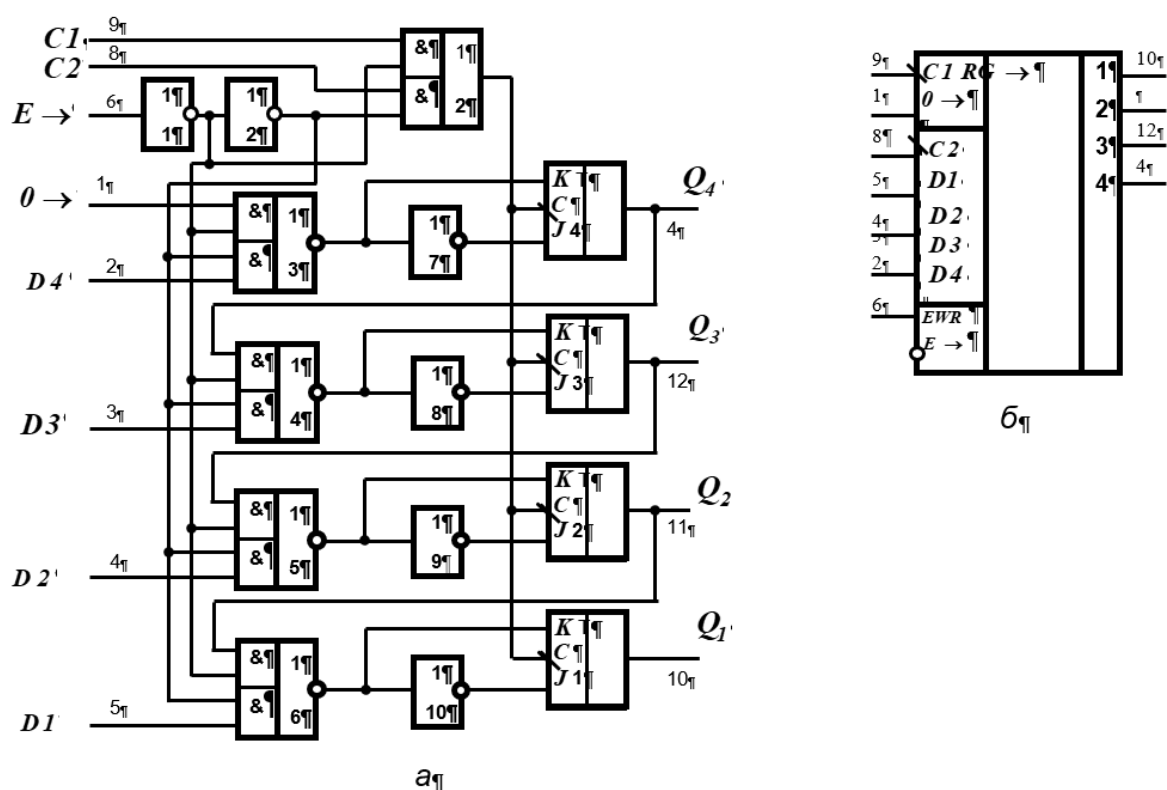


Рис. 6.6. Схема (а) та УГП (б) регістра 133ІР1

Багато сучасних регістрів, які виконані у вигляді ІС, містять вихідні ключі з високоімпедансним станом, що спрощує сполучення з інформаційними магістралями. Споживчі можливості регістрів ще вище, якщо ключі двонаправлені. Уведення ключів також у входні кола регістра дозволяє істотно спростити організацію двонаправленої передачі інформації між двома магістралями, причому з можливістю її попередньої обробки. Такою обробкою може бути зсув двійкового числа на m розрядів вліво або вправо. Якщо при цьому не відбувається втрата знаків числа (що можливо, коли розрядність регістра вище розряд-

ності числа), то зсув рівноцінний відповідно множенню або діленню числа на 2^m .

Область застосування регістрів зсуву не обмежується прикладами, наведеними вище. Вона значно ширше. Так, регістр може стати кільцевим лічильником при підключенні входу $0 \rightarrow$ до прямого виходу останнього тригера. Якщо в один з розрядів регістра ввести логічну одиницю або нуль, то ця одиниця або нуль з кожним імпульсом синхронізації C' буде переходити від тригера до тригера по замкнутому колу з циклом, який дорівнює кількості тригерів (n). У результаті на виходах тригерів формуються імпульсні послідовності, зміщені відносно один одного на період проходження імпульсів C' . Період проходження імпульсів у кожній послідовності в n разів більший, ніж період проходження імпульсів C' .

Якщо на вхід $0 \rightarrow$ подавати сигнал з виходу дешифратора, який виділяє певні стани регістра, виходить генератор кодових послідовностей.

Підключивши вхід $0 \rightarrow$ до інверсного виходу останнього тригера, можна збільшити коефіцієнт перерахування кільцевого лічильника в два рази ($K_n = 2^n$). У такому лічильнику всі тригери послідовно переходять в одиничний стан, а потім також послідовно – в нульовий і т.д. Тому для отримання зрушених імпульсних послідовностей необхідний спеціальний дешифратор. Особливість дешифратора полягає в тому, що на кожний стан лічильника з будь-якою розрядністю потрібен кон'юнктор усього лише на два входи.

6.3. Лічильники імпульсів

Лічильник призначений для підрахунку кількості одиниць інформації (рахункових імпульсів). Надходження одиниці інформації полягає у впливі на вхід лічильника перепаду напруги $0 \rightarrow 1$ при прямому вході або $1 \rightarrow 0$ при інверсному вході. Оскільки один імпульс містить той та інший перепади, то його й ототожнюють з одиницею інформації незалежно від типу входу лічильника.

Лічильник має K_n стійких станів, кожний з яких повторюється після підрахунку K_n рахункових імпульсів. Інакше кажучи, рахунок імпульсів здійснюється з коефіцієнтом (модулем) перерахування K_n .

За способом кодування числової інформації розрізняють лічильники з позиційним (одиничним, двійковим, десятковим й ін.) та непозиційним (наприклад, у кодах Грея) кодуванням. У лічильниках з позиційним кодуванням числовий вираз поточного стану визначається за формулою:

$$A = \sum_{i=0}^{n-1} Q_i M_i ,$$

де n – кількість розрядів; Q_i – логічне значення розряду ($Q_i = 0, 1$); M_i – вага i -го розряду.

У лічильниках з непозиційним кодуванням розряди не мають постійних ваг, і числовий вираз стану пропонується кожному наборові значень Q_i . Цей тип лічильників зустрічається на практиці істотно рідше, ніж лічильники з позиційним кодуванням.

Нижче розглядаються найбільш розповсюджені лічильники, які ведуть рахунок у двійковій системі числення, – двійкові лічильники. В основу їх побудови покладені n рахункових тригерів. Кожному з них ставиться у відповідність одна визначена вага з набору: $2^0, 2^1, \dots, 2^{n-1}$. Кількість імпульсів, які надійшли, подається у вигляді суми

$$A = Q_{n-1}2^{n-1} + Q_{n-2}2^{n-2} + \dots + Q_12^1 + Q_02^0,$$

або для стислості запису – у вигляді позиційного набору $Q_{n-1} Q_{n-2} \dots Q_1 Q_0$. Максимальний коефіцієнт перерахування двійкового лічильника $K_n = 2^n$.

Лічильники підрозділяються ще за іншими класифікаційними ознаками. За призначенням розрізняють лічильники підсумовуючі, віднімаючі і реверсивні; за способом запуску – асинхронні і синхронні; за способом організації перенесення – з послідовним, наскрізним і паралельним перенесеннями.

Основними параметрами лічильників є: роздільна здатність, час установа-лення і ємність.

Роздільна здатність – це мінімальний період проходження рахункових T імпульсів, при яких зберігається працездатність лічильника. Зворотна величина $F=1/T$ характеризує максимальну частоту рахунку.

Час установа-лення $T_{уст}$ – це інтервал між початком подачі на вхід рахункового імпульсу та моментом закінчення найтривалішого перехідного процесу в лічильнику.

Параметри T і $T_{уст}$ характеризують швидкодію лічильника.

Ємність лічильника визначається максимальною кількістю імпульсів, яку він може зареєструвати. Чисельно ємність дорівнює коефіцієнтові перерахування K_n .

Дільники – це ті ж лічильники, але мають, як правило, один вихід, на якому з'являється імпульсний сигнал після кожних K_n рахункових імпульсів. Часто передбачається можливість змінювати коефіцієнт ділення K_n за допомогою спеціального керуючого коду. У принципі дільники можна будувати й на основі

двійкових лічильників, підключивши до виходів дешифратор якого-небудь одного стану [4].

6.3.1. Лічильники з послідовним перенесенням

Підсумовуючі лічильники. Підсумовуючий лічильник повинен функціонувати так, щоб при надходженні на його вхід одного імпульсу записане в ньому число збільшилося на одиницю. Принцип побудови підсумовуючого лічильника впливає з правила додавання до двійкового числа одиниці. Відповідно до цього правила, наприклад, трирозрядний лічильник повинен послідовно приймати стани 000, 001, 010, 011, 100, ...111.

Видно, що тригер молодшого розряду перемикається кожним рахунковим імпульсом, тобто входом лічильника служить вхід цього тригера. Стани другого та третього тригерів змінюються відповідно кожним другим і четвертим імпульсом. Це забезпечується послідовним з'єднанням тригерів.

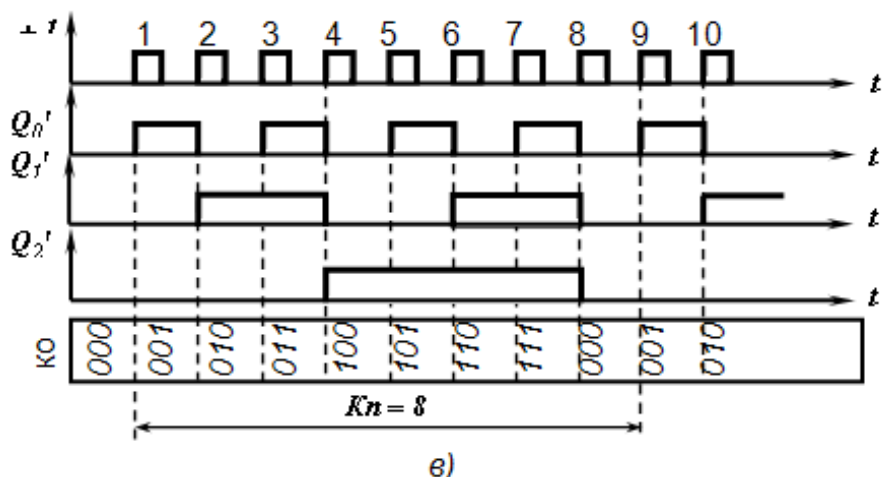
З правила додавання до двійкового числа одиниці також відомо, що зміна значення i -го розряду відбувається тоді, коли до додавання чергової одиниці всі попередні розряди були одиницями. Звідси впливає правило: якщо тригери мають прямий рахунковий вхід, то він підключається до інверсного виходу попереднього тригера; якщо має інверсний вхід, то він підключається до прямого виходу.

Приклад, який ілюструє роботу трирозрядного підсумовуючого лічильника з часовими діаграмами роботи й умовним зображенням, наведений на рис. 6.7. Лічильник може приймати **8** різних станів, які повторюються через кожні **8** вхідних імпульсів ($K^n = 8$). З найбільшою частотою перемикається тригер молодшого розряду, отже, розрізнявальна здатність лічильника визначається часом затримки перемикання тригера ($T = T_{m2}$). Це положення поширюється на всі типи двійкових лічильників.

Даний лічильник називають лічильником з послідовним перенесенням, тому що перемикання тригера i -го розряду відбувається в результаті послідовного перемикання всіх попередніх тригерів молодших розрядів, тобто інформація поширюється по колу тригерів послідовно. Час установа лічильника:

$$T_{уст} = nT_{m2}.$$

Це говорить про те, що якщо, наприклад, необхідно знімати інформацію після кожного вхідного імпульсу, то період їх проходження повинен бути $T > nT_{m2}$.



Помітимо, що даний лічильник можна розглядати як підсумовуючий, а підсумовуючий (рис. 6.7) – як віднімаючий лічильник при інвертуванні вихідних сигналів Q_{ji} або зніманні інформації з виходів $\overline{Q}_{ji}$.

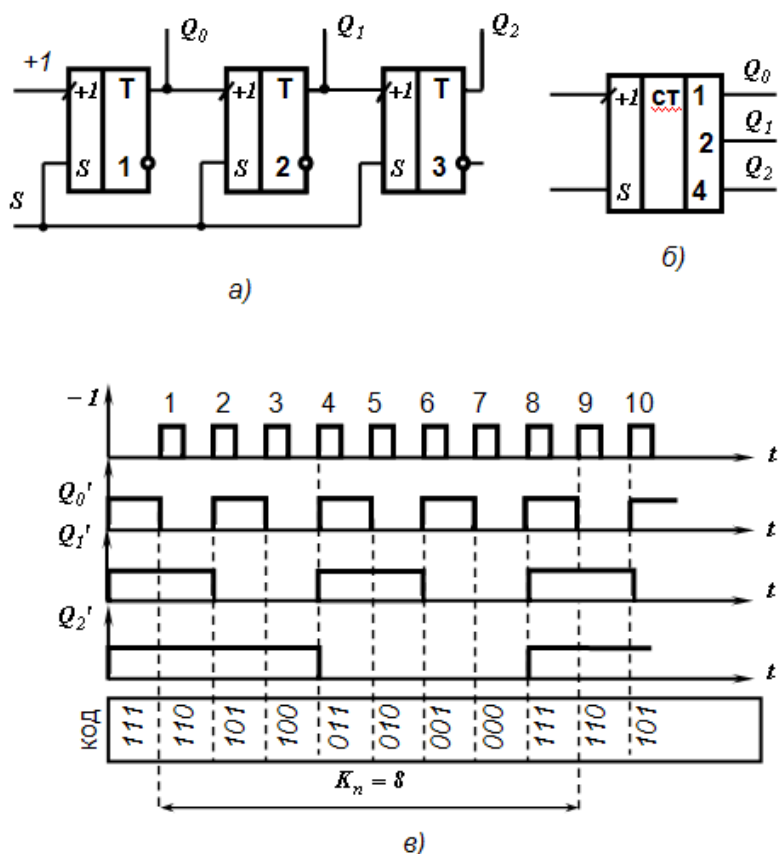


Рис. 6.8. Схема (а), умовне графічне позначення (б) та часові діаграми (в) трирозрядного віднімаючого лічильника з послідовним перенесенням

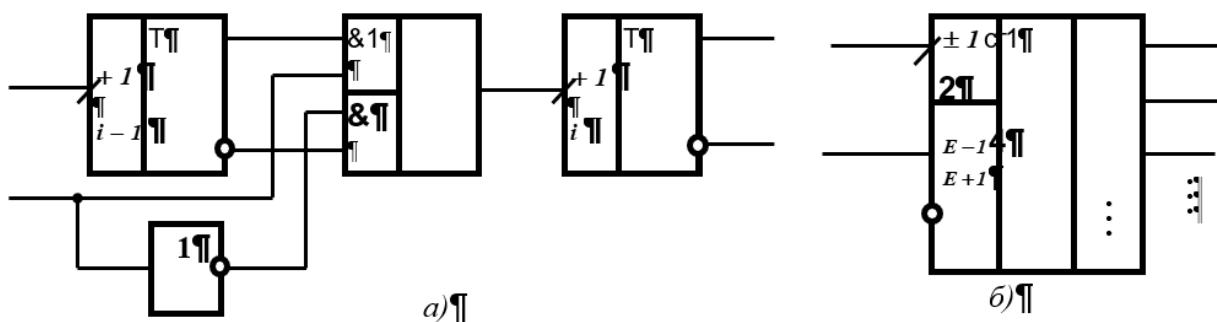


Рис. 6.9. Міжтригерні зв'язки (а) та умовне графічне позначення (б) реверсивного лічильника

Реверсивні лічильники. Вони можуть працювати як у режимі підсумовування, так і віднімання. Переключення режимів здійснюється комутацією рахункових входів усіх тригерів (крім тригера молодшого розряду) до інверсних або прямих виходів попередніх тригерів (рис. 6.9). Переключення режиму виконує сигнал дозволу E' . При $E' = 1$ прямих вхід i -го тригера комутується до пря-

мого виходу $(i-1)$ -го тригера, що відповідає режимові віднімання. При $E' = 0$ вхід підключається до інверсного виходу, і тригер стає підсумовуючим.

Універсальність реверсивного лічильника досягається ціною введення додатково $n-1$ логічних елементів і погіршення швидкодії:

$$T_{уст} = nT_{мг} + (n-1) t_{зм_{сер}},$$

де $t_{зм_{сер}}$ – середній час затримки перемикаання ЛЕ.

Помітимо, що правило з'єднання інформаційних входів тригерів для одержання підсумовуючих та віднімаючих лічильників поширюється і на лічильники з прискореним переносом. Тому з метою скорочення матеріалу нижче розглядаються лічильники тільки підсумовуючого типу.

6.3.2. Лічильники з наскрізним перенесенням

Як відомо, тригер i -го розряду перемикається, якщо $(i-1)$ -й тригер до моменту надходження на його вхід чергового рахункового сигналу T'_{i-1} знаходився в одиничному стані, тобто має місце $Q'_{i-1} \cdot T'_{i-1} = 1$. Тому з метою прискорення переносу можна вхідний сигнал T'_{i-1} пропустити на вхід i -го тригера за допомогою елемента І, мінаючи $(i-1)$ -й тригер (рис. 6.10).

Тригери можуть бути асинхронними та синхронними. Відповідно до цього лічильники виходять асинхронними або синхронними. Лічильник на рис. 6.10 буде синхронним, якщо як рахунковий вхід використовувати вхід синхронізації C , показаний пунктиром.

З часових діаграм для асинхронного лічильника, показаних на рис. 6.10 в видно, що завдяки схемі перенесення на входи другого, третього і т.д. тригерів транзитом передаються відповідно кожен другий, четвертий і т.д. вхідні імпульси. При цьому проходження імпульсів на вхід останнього тригера затримується на час перенесення:

$$T_{пер} = (n-1) t_{зм_{сер}}.$$

Час устанавлення лічильника

$$T_{уст} = T_{мг} + (n-1) t_{зм_{сер}}.$$

Приймаючи до уваги, що час затримки проходження сигналу через ЛЕ І менший, ніж через тригер, вираш в швидкодії у лічильників з наскрізним пе-

ренесенням порівняно з лічильниками з послідовним перенесенням очевидний. Однак тут більший обсяг устаткування.

У синхронного лічильника з наскрізним перенесенням рахунковим входом є об'єднаний вхід синхронізації C усіх тригерів, завдяки чому вони перемикаються одночасно. Інформаційний вхід першого тригера стає входом дозволу режиму рахування E (на рис. 6.10 а) позначення входів для даного варіанта показані в дужках, а умовне зображення наведене на рис. 6.10 з). При $E' = 0$ всі сигнали $T'_i = 0$, і лічильник знаходиться у режимі збереження. При $E' = 1$ встановлюється режим рахування.

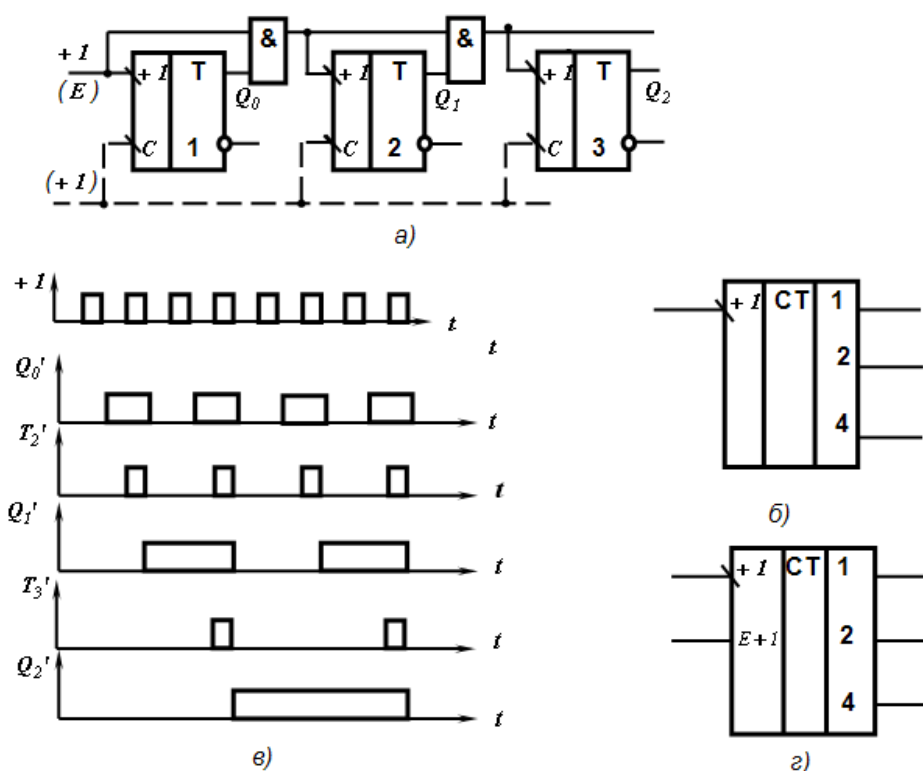


Рис. 6.10. Схема (а), умовне графічне позначення (б) і часові діаграми (в) асинхронного лічильника з наскрізним перенесенням; синхронний лічильник (з урахуванням пунктирної лінії на схемі) та його умовне графічне позначення (з)

Час перенесення і час установлення тут такі ж, як і в асинхронного лічильника. Однак, оскільки в асинхронному лічильнику протягом усього часу $T_{уст}$ йде безперервне (із затримкою $t_{зм\text{сер}}$) перемикання тригера, то для знімання інформації необхідний додатковий час, тобто збільшення періоду проходження вхідних імпульсів. У синхронному лічильнику тригери перемикаються одночасно (рахунковим імпульсом) і лише потім відбувається перенесення у колі логі-

чних елементів. Виходить, для знімання інформації додатковий час не потрібний – можна використовувати час $T_{пер}$. У цьому сенсі швидкодія синхронного лічильника вища, ніж асинхронного.

6.3.3. Лічильники з паралельним перенесенням

Для отримання паралельного (одночасного) перенесення використовується та обставина, що перемикання тригера i -го розряду відбувається тоді, коли перед надходженням чергового імпульсу на вхід лічильника всі тригери молодших розрядів знаходяться в одиничному стані, тобто $Q'_0 Q'_1 \dots Q'_{i-1} = 1$. Тому сигнал на інформаційний вхід кожного тригера можна генерувати шляхом кон'юнкції вихідних сигналів попередніх тригерів. Принцип побудови і роботи лічильника з паралельним перенесенням ілюструє рис. 6.11. Тут показаний найбільш розповсюджений варіант синхронного типу, який використовує двоступінчаті JK -тригери.

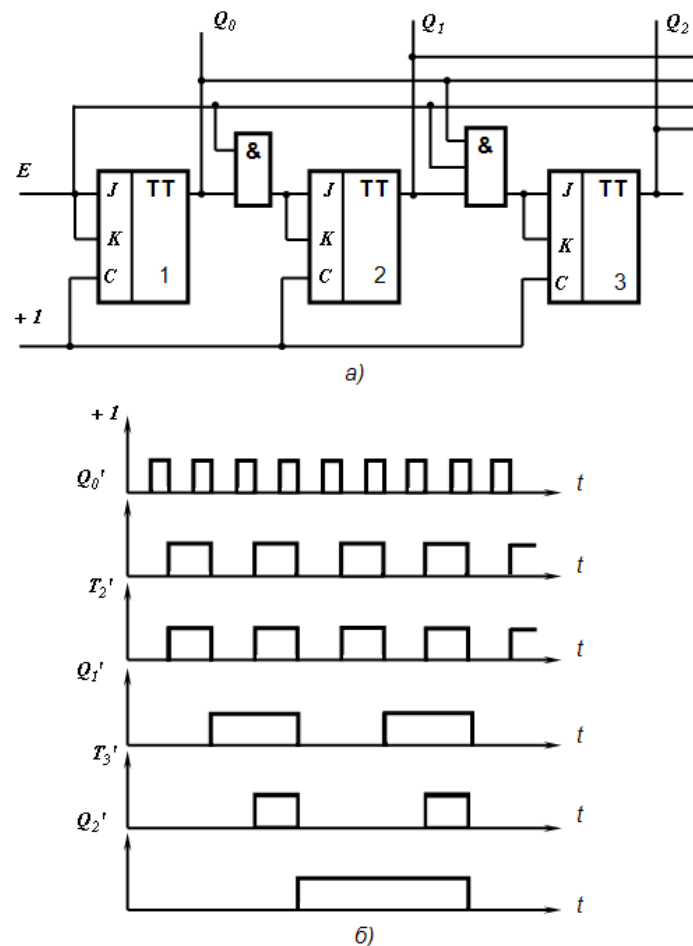


Рис. 6.11. Схема (а) та часові діаграми (б) лічильника з паралельним перенесенням

Час установлення лічильника:

$$T_{уст} = T_{mг} + t_{зм_{сер}},$$

тобто практично визначається його роздільна здатність. Однак маючи найвищу швидкодію, вона має один істотний недолік – нерегулярність схем перенесення. Зі зростанням ваги розряду кількість входів ЛЕ збільшується. Тому при великій розрядності вдаються до комбінованого перенесення: паралельного усередині груп тригерів і послідовного або наскрізного між групами.

У випадку побудови лічильника на **JK**-тригерах, які мають кілька інформаційних входів, об'єднаних кон'юнкцією, структура його спрощується – виключаються зовнішні елементи *I* (рис. 6.12).

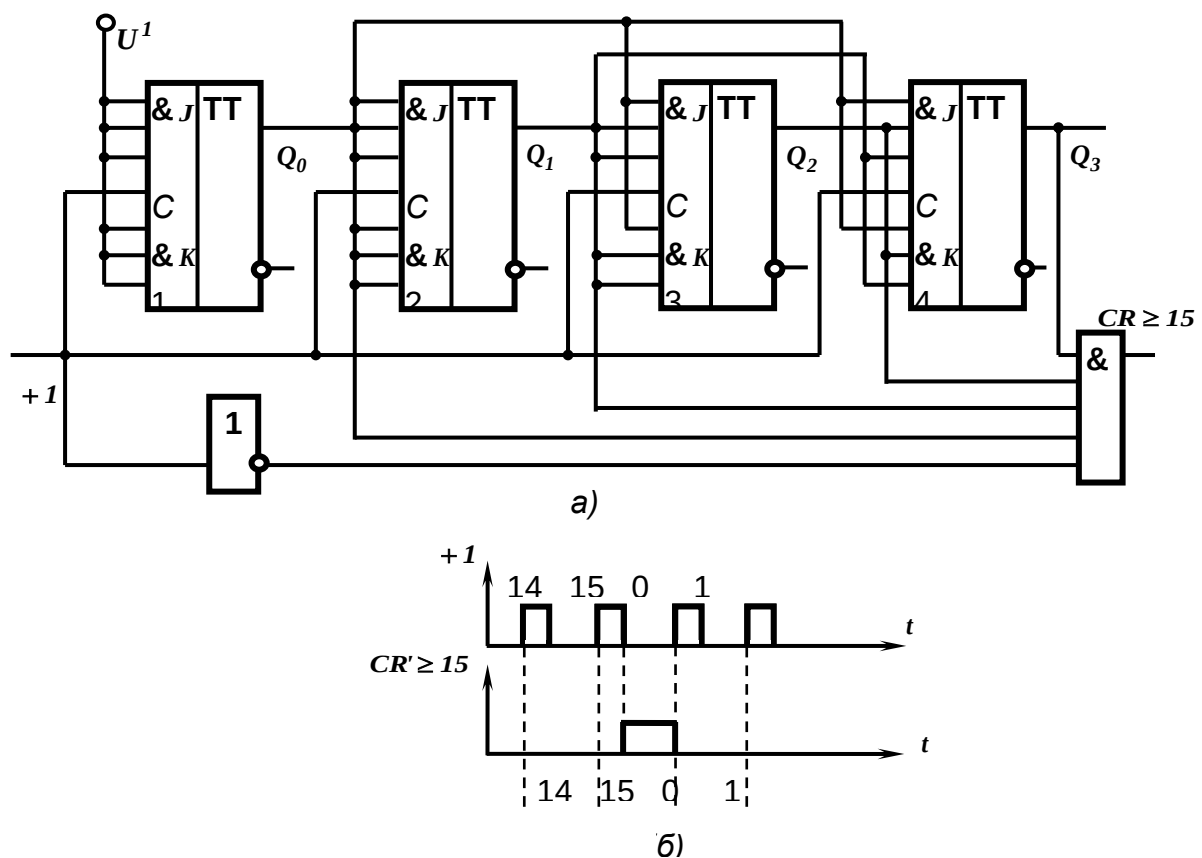


Рис. 6.12. Чотирихрозрядний лічильник з формувачем сигналу перенесення

Крім того, відмовлення від входу дозволу *E* дозволяє збільшити розрядність лічильника на одиницю без збільшення кількості інформаційних входів тригерів.

З метою збільшення розрядності лічильники можна з'єднувати послідовно. Для цього використовується сигнал перенесення **CR'** (≥ 15), який для наступного лічильника є рахунковим. Генерується він за допомогою імпульсу, який

стробується вхідним імпульсом дешифратора старшого (п'ятнадцятого) стану. З позначення (≥ 15) видно, що сигнал перенесення з'являється в проміжку після закінчення 15-го рахункового імпульсу до надходження чергового – нульового (рис. 6.12 б).

Лічильники з паралельним перенесенням, виконані у вигляді окремих ІС, зазвичай мають додаткові входи для попереднього запису інформації.

Наприклад, у чотирьохрозрядному реверсивному лічильнику 133ИЕ7 (рис. 6.13) такий запис здійснюється за допомогою входів D_i при подачі сигналу $C' = 0$. Реверсивний режим досягається відомим способом (рис. 6.9): кожен тригер для своїх сигналів Q' та $\bar{Q}'$ має мультиплексор, який керується сигналами з входів $+1$ та -1 , комутуючи вихід Q або $\bar{Q}$ з рахунковим входом наступного тригера. Для нарощування розрядності у режимі віднімання є вихід займу (≤ 0), на якому сигнал з'являється в інтервалі після закінчення останнього (нульового) рахункового імпульсу до надходження чергового (п'ятнадцятого). Формується він дешифратором нульового стану лічильника.

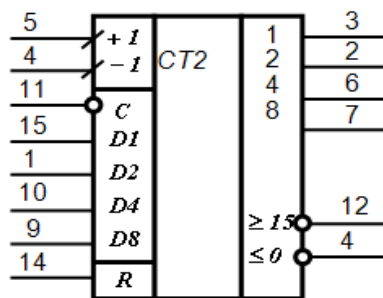


Рис. 6.13. Реверсивний лічильник 133ИЕ7

6.4. Дільники імпульсів

Велике поширення одержали лічильники та дільники з $K_n \neq 2^n$. Так, у цифрових індикаторних пристроях домінують двійково-десяткові лічильники ($K_n = 10$).

Принцип побудови лічильників з $K_n \neq 2^n$ зводиться до такого. Беруть таку кількість n тригерів, щоб виконувалася умова $2^{n-1} < K_n < 2^n$.

Потім схемним шляхом виключають $2^n - K_n$ надлишкових станів. Найчастіше виключають старші стани, рідше – молодші або проміжні. Робиться це або за допомогою дешифратора визначеного стану, який своїм вихідним сигналом примусово встановлює лічильник у вихідний стан, або за допомогою зворотних зв'язків між тригерами.

На рис. 6.14 наведений приклад декадного (двійково-десятькового) лічильника з виключенням старших надлишкових станів. Початковий стан у нього нульовий: $A_0 \{0000\}$. При надходженні входних імпульсів рахунок йде як у звичайному двійковому лічильнику. Як тільки встановлюється стан $A_{10} \{1010\}$, на виході елемента І, який відіграє роль дешифратора, генерується сигнал $y = Q_1 Q_3 = 1$, і лічильник примусово переводиться у початковий стан A_0 .

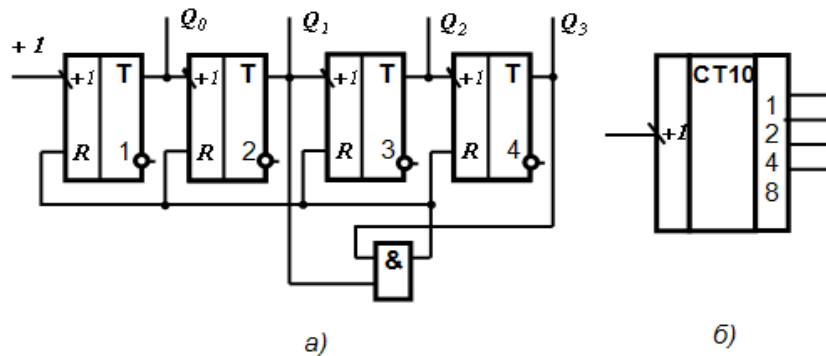


Рис. 6.14. Двійково-десятьковий лічильник з виключенням старших надлишкових станів

Прикладом використання зворотних зв'язків для виключення надлишкових станів може служити ІС 133ИЕ2 (рис. 6.15 а).

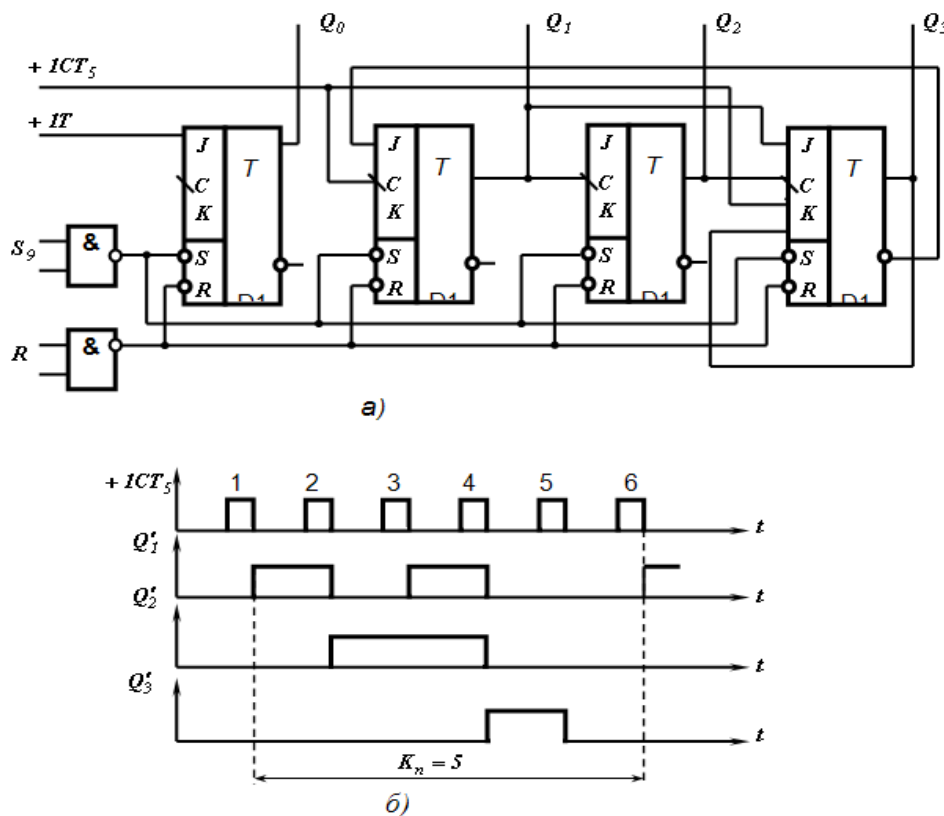


Рис. 6.15. Схема (а) та часові діаграми (б) у режимі $Kn = 5$ лічильника 133ИЕ2

Вона містить T -тригер **D1** та двійково-п'ятірковий лічильник на тригерах **D2**, **D3**, **D4**. Завдяки зворотному зв'язкові з виходу $\overline{Q_3}$ тригера **D4** на вхід J тригера **D2**, а також зворотного зв'язку в самому тригері **D4** (з виходу $\overline{Q_3}$ на вхід **K**), забезпечується відповідно блокування дії п'ятого рахункового імпульсу на тригер **D2** і установка тригера **D4** п'ятим імпульсом у нульовий стан (рис. 6.15 б)). Таким чином, після п'ятого імпульсу виходить $A_0 \{000\}$. Якщо вихід Q_0 тригера **D1** з'єднати з входом $+1$ лічильника, а рахункові імпульси подавати на вхід $+1T$, то лічильник стає двійково-десятковим з $K_n = 10$.

Як відзначалося раніше, принцип побудови дільників багато в чому аналогічний принципів побудови лічильників. Вони мають, як правило, один вихід, на якому за інтервал перерахування з'являється імпульс з частотою, меншою у коефіцієнт ділення, ніж надходять на вхід ($K_{dil} = \frac{N_{Bx}}{N_{Bux}}$). Виділяються ці імпульси за допомогою дешифратора станів.

Змінювати коефіцієнт K_{dil} у дільниках можна так само, як і в лічильниках з $K_n \neq 2^n$, тобто шляхом виключення різної кількості надлишкових станів, або програмно – за допомогою зовнішніх керуючих сигналів. Такий спосіб реалізований, наприклад, у **IC564IE15**. Вона дозволяє отримувати $K_{dil} = 3...21327$ із кроком одиниця.

Знайшов поширення також спосіб, за якого коефіцієнт перерахування лічильника, який складає основу дільника, не міняється, а здійснюється вибір дешифраторів, які настроєні на виділення різних станів лічильника. Даний спосіб побудови дільників ілюструється на прикладі **IC133IE8** (рис. 6.16).

Дільник складається із шестирозрядного двійкового лічильника з паралельним перенесенням, дешифраторів станів на **ЛЕ1...ЛЕ6**, виходи яких об'єднані в один вихід $\overline{Y}$ елементом **НЕ-ЧИ** (**ЛЕ7**), а також дешифратора-формувача сигналу перенесення $\overline{CR}$ і допоміжних логічних елементів. Кожен дешифратор включається в роботу при подачі на нього керуючого сигналу $K'_i = 1$.

Дешифратори настроєні на виділення неоднакових станів. Наприклад, дешифратор на **ЛЕ1** виділяє стани **0,2,4** і т.д. через **2**, на **ЛЕ2** – **1,5,9** і т.д. через **4**, на **ЛЕ3** – **3,11,19** і т.д. через **8**. Тому сигнали (імпульси) на виходах дешифраторів у часі не збігаються, і на загальному виході $\overline{Y}$ виходить їх сума.

Таким чином, за один цикл роботи, що містить $2^6 = 64$ вхідних імпульсів, кількість імпульсів на виході $\overline{Y}$ буде меншою і визначається числом

$N_{\text{вих}} = 2^5 K'_5 + 2^4 K'_4 + \dots + 2^0 K'_0$ відповідно до кодової комбінації на входах $K_0 \dots K_5$. Коефіцієнт ділення $K_{\text{дл}} = \frac{N_{\text{вх}}}{N_{\text{вих}}} = \frac{64}{N_{\text{вих}}}$.

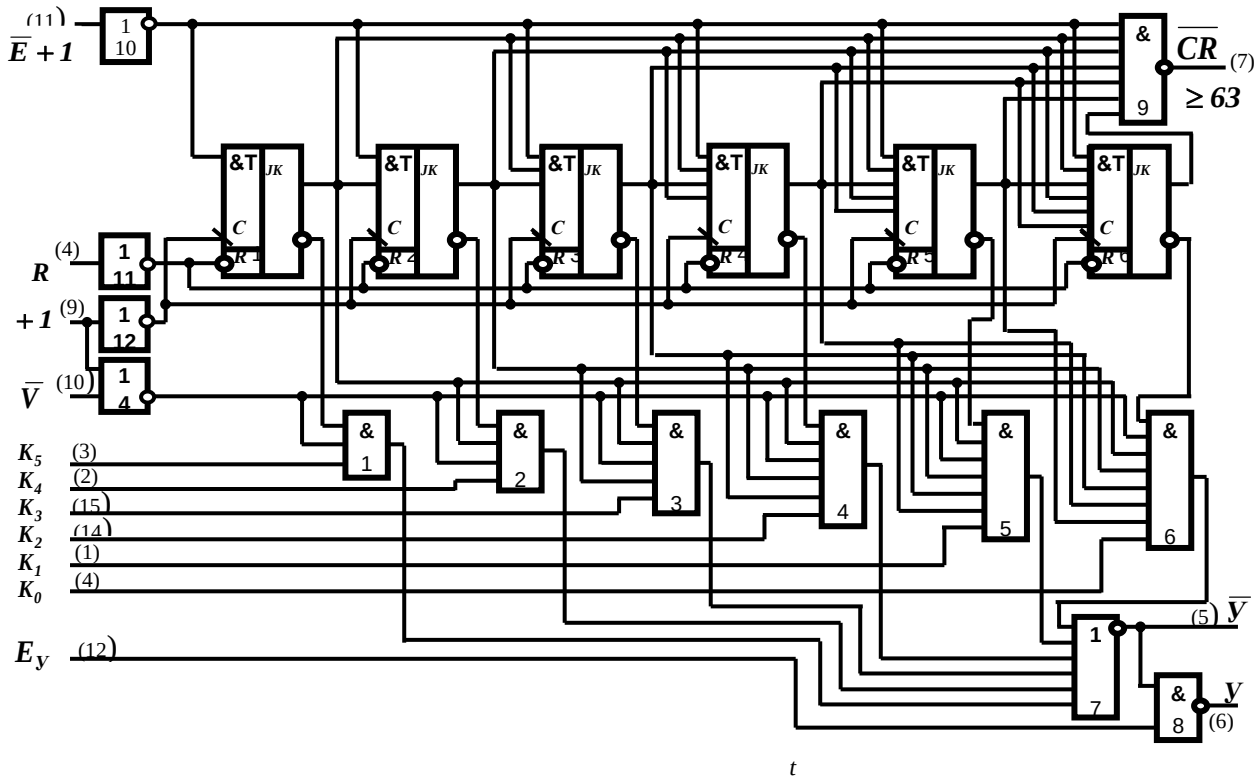


Рис. 6.16. Дільник з можливістю керування коефіцієнтом ділення 133ИЕ8

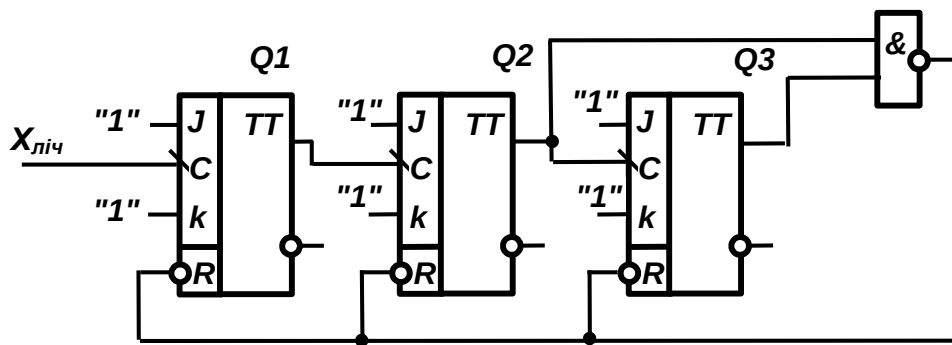
Якщо тільки один із сигналів $K'_i = 1$, то $N_{\text{вих}} = 2^i$, тобто коефіцієнт ділення – ціле число. В інших випадках коефіцієнт ділення – число дробове. Варто звернути увагу на те, що якщо вхідна послідовність імпульсів періодична, то вихідна послідовність буде періодичною тільки, коли коефіцієнт розподілу – ціле число, в інших випадках вихідні імпульси розподілені в часі нерівномірно.

З метою збільшення діапазону перебудови коефіцієнта розподілу мікросхеми з'єднують послідовно. Сигнали перенесення відіграють роль вхідних стосовно наступних мікросхем.

КОНТРОЛЬНІ ПИТАННЯ

1. Дати визначення регістру.
2. Пояснити принцип роботи регістра зсуву.
3. Дати визначення реверсивного регістра та пояснити принцип роботи міжтригерних зв'язків.

4. Яким чином на базі регістру **133ИР1** можна організувати перетворювач паралельного коду в послідовний?
5. Дати визначення лічильнику.
6. Якими часовими параметрами характеризується робота лічильника?
7. Надати порівняльну характеристику лічильника з різною організацією переносів в між розрядних ланцюгах.
8. Побудувати лічильник, що працює в режимі додавання (віднімання) з коефіцієнтом перерахування $K = 11$.
9. Зобразити схему найбільш економного (за кількістю елементів що використовуються) чотирьохрозрядного лічильника.
10. Який коефіцієнт лічби реалізує приведений на схемі лічильник?



11. Навести часові діаграми сигналів на вході лічильника та виходах тригерів.
12. Яка кількість інтегральних мікросхем **133ИЕ7** знадобиться для того, щоб побудувати дільник з коефіцієнтом ділення **1953**?

7. ПЕРЕТВОРЕННЯ АНАЛОГОВИХ СИГНАЛІВ У ЦИФРОВІ

7.1. Загальні відомості про перетворювачі

Успішний розвиток методів і засобів цифрової обробки інформації сприяє їх широкому впровадженню в системи сучасних комунікацій. Інформація в цифрових пристроях подається в дискретній цифровій формі, а вхідні величини, які надходять від датчиків, і величини, які надходять на виконавчі пристрої, є, як правило, аналоговими. У зв'язку з цим виникає необхідність узгодження аналогових датчиків і виконавчих пристроїв із засобами цифрової обробки інформації. Цю задачу вирішують перетворювачі інформації [7].

До першої групи перетворюючих пристроїв відносяться *аналого-цифрові перетворювачі (АЦП)*. Вони служать для перетворення вихідних аналогових величин у відповідний цифровий еквівалент – код.

До другої групи відносяться *цифроаналогові перетворювачі (ЦАП)*.

Вони призначені для одержання вихідної аналогової величини, яка відповідає цифровому коду, який надходить на вхід перетворювача.

Призначення та місце перетворень інформації можна пояснити на прикладі спрощеної структурної схеми автоматизованої системи керування, виконаної на основі ЕОМ (рис. 7.1).

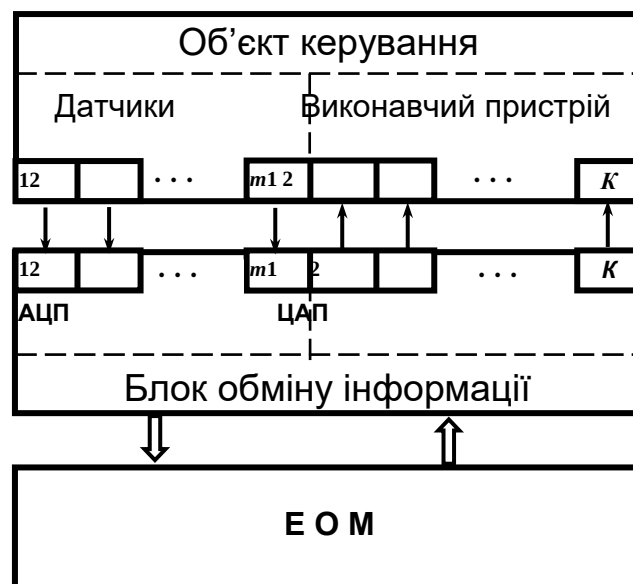


Рис. 7.1. Структурна схема автоматизованої системи керування

Інформація про стан об'єкта керування в аналоговому вигляді надходить з датчиків на АЦП. За допомогою АЦП аналогові сигнали перетворюються в

цифровий код, який через блок обміну інформацією надходить в **ЕОМ**. Цифровий код, який характеризує результат рішення задачі керування, з виходу **ЕОМ** через блок обміну передається на цифроаналогові перетворювачі. **ЦАП** перетворює цифровий код в аналогові сигнали (напруга, струм, часовий інтервал тощо) і видає їх на виконавчі пристрої та пристрої відображення. Перетворювачі можуть входити до складу блока обміну інформацією або безпосередньо до складу датчиків, виконавчих пристроїв і пристроїв відображення. **АЦП** і **ЦАП** є досить складними пристроями й значною мірою визначають параметри всієї системи.

Функція, яку виконує **АЦП**, може бути визначена як заміна аналогової безперервної величини $x(t)$ цифровим кодованим сигналом $N(i\Delta t)$, пропорційним її миттєвим значенням у визначенні моменту часу t_i , розділені інтервалом дискретизації Δt .

У загальному випадку процес аналого-цифрового перетворення містить у собі дискретизацію за часом, квантування за рівнем і кодування результату. Відповідно до цього узагальнена структурна схема **АЦП** може бути подана у вигляді, показаному на рис. 7.2.



Рис. 7.2. Узагальнена структурна схема **АЦП**

У результаті дискретизації безперервна в часі аналогова величина $x(t)$ перетворюється у вибірки її дискретних значень $x'(t_i)$ у моменти дискретизації t_i . Часовий інтервал Δt між сусідніми вибірками називається *кроком дискретизації*. Він може бути постійним і змінним. Постійний крок Δt називається *періодом дискретизації*. Вихідна аналогова величина $x(t)$ і відповідна їй дискретизована величина $x'(t_i)$ наведені на рис. 7.3 а) і б).

Операцію дискретизації аналогової величини можна виконати за допомогою аналогового ключа, що короткочасно замикається у моменти дискретизації t_i .

Для подальшого аналого-цифрового перетворення іноді виникає необхідність запам'ятовування і збереження значень вибірок на період дискретизації.

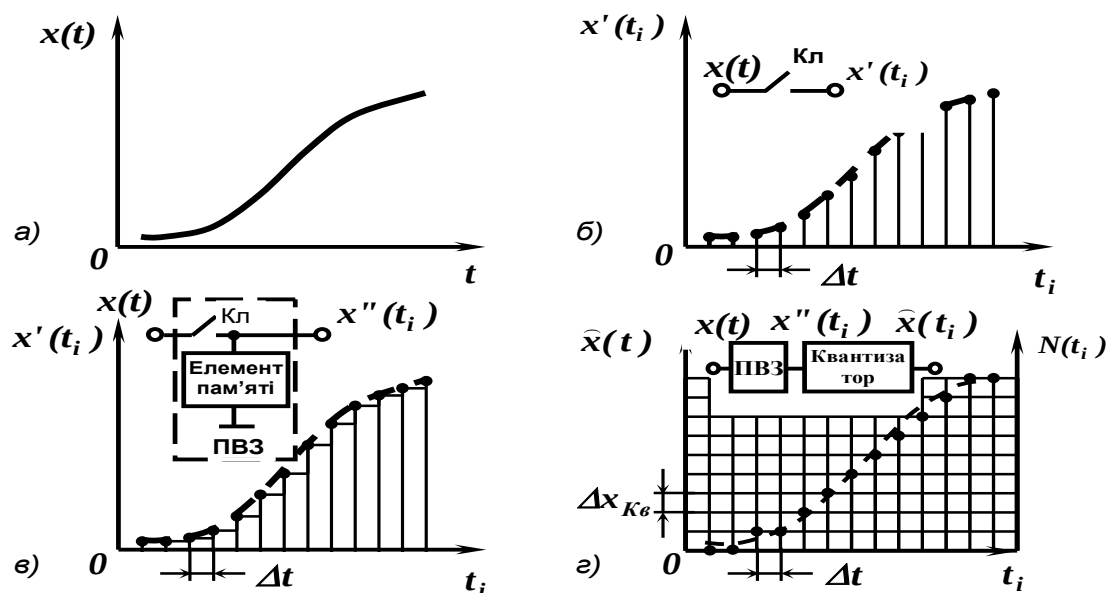


Рис. 7.3. Форми сигналу на вході та виході основних вузлів АЦП:

- а) – вхідний аналоговий сигнал;
- б) – дискретний сигнал на виході дискретизатора;
- в) – сигнал на виході ПВЗ;
- г) – сигнал на виході квантизатора та на вихідний цифровий сигнал АЦП

Для цього після ключа включається елемент запам'ятовування. Ключ з елементом запам'ятовування називають пристроєм вибірки та збереження (ПВЗ). Такий пристрій дозволяє фіксувати значення вибірок аналогового сигналу, що змінюється, на час його перетворення. Сигнал на виході ПВЗ $x''(t_i)$ має вигляд, показаний на рис. 7.3 в).

Операція квантування безперервної за значенням величини $x(t)$ здійснюється шляхом заміни значень вибірок цієї величини $x'(t_i)$ найближчими фіксованими значеннями, які називаються рівнями квантування. Рівні квантування мають значення від мінімального Δx_{K6} , який ще називають кроком квантування, до максимального $N \Delta x_{K6}$, де N – кількість рівнів квантування однієї полярності. Квантування виконується за допомогою квантизатора. Квантизатор становить датчик рівнів квантування й один або кілька пристроїв порівняння (компараторів), які порівнюють вибірки

перетворюваної величини з відповідними (найближчими) рівнями квантування. Типові статичні передаточні характеристики квантизатора наведені на рис. 7.4. Сигнал на виході квантизатора $\hat{x}(t)$ має вигляд, показаний на рис. 7.3 з. За результатом квантування аналогової величини $x(t)$ виникає методична похибка, яка називається похибкою квантування (шумом квантування). Максимальна похибка квантування залежить від виду характеристики квантизатора і складає $\pm \frac{\Delta x_{kv}}{2}$ (рис. 7.4 а) або Δx_{kv} (рис. 7.4 б).

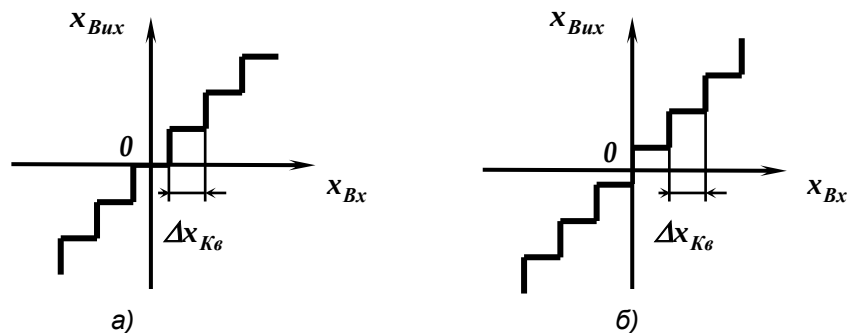


Рис. 7.4. Типові статичні характеристики квантизаторів:
а) незміщена; б) зміщена

У загальному випадку до складу АЦП входить кодувальний пристрій, який перетворює номер рівня квантування, що врівноважує величину $x(t_i)$, у цифровий $N(t_i)$ код у необхідній системі числення. Пристрій, який кодує, може виконуватися на елементах цифрової логіки, а в окремих перетворювачах – на ПЗП. На рис. 7.3 з наведений приклад кодування рівнів квантування в цифровому двійковому коді.

У більшості типів АЦП послідовність виконання зазначених вище операцій (рис. 7.2) може бути іншою, окремі операції можуть не виконуватися або кілька операцій виконуються одночасно одним типовим вузлом.

ЦАП є декодувальним пристроєм і виконує операцію, протилежну аналого-цифровому перетворенню, тобто ставить у відповідність вхідному цифровому кодові визначене значення аналогової величини. Узагальнена структурна схема ЦАП наведена на рис. 7.5. На його вхід надходить цифровий код $N(t_i)$ та аналогова еталонна величина x_{Em} , з якої формується набір еталонів. Набір аналогових ключів, керованих вхідним кодом, підключає відповідні еталони до вихідного суматора. При цьому вихідний аналоговий сигнал ЦАП буде пропорційний вхідному цифровому кодові.

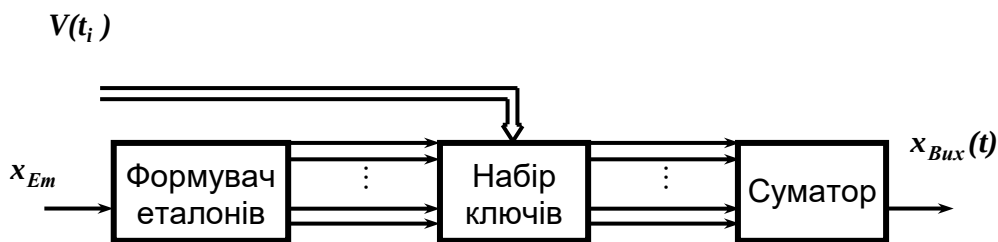


Рис. 7.5. Узагальнена структурна схема ЦАП

Необхідно зазначити, що перетворювачі інформації є досить складними аналого-цифровими пристроями. До їх складу входять аналогові ключі та перемикачі, компаратори аналогових сигналів, джерела опорних і еталонних сигналів, пристрої вибірки та збереження, операційні підсилювачі, цифрові пристрої тощо. Названі елементи і вузли випускаються серійно у вигляді окремих ІС або входять до складу інтегральних перетворювачів. Розглянемо докладніше аналогові ключі струму і пристрої вибірки та збереження. Параметри цих вузлів значною мірою визначають швидкодію і точність перетворювачів.

7.1.1. Пристрої вибірки та збереження

Пристрої вибірки та збереження (ПВЗ) аналогових сигналів служать для фіксації миттєвих значень (які відповідають моментам дискретизації) сигналів на час, необхідний для їх обробки. Сигнал на виході **ПВЗ** відповідає формі подання, наведений на рис. 7.3 в. **ПВЗ** застосовуються разом з **АЦП** для зменшення динамічних похибок, що виникають у процесі перетворення безперервних сигналів, які змінюються в часі. Такі пристрої дозволяють здійснити точну часову прив'язку вихідного коду **АЦП** до реального моменту зняття відліку.

Найбільш точними і компактними пристроями короткочасного збереження аналогових сигналів є схеми, засновані на принципі збереження заряду на накопичувальному елементі, наприклад, конденсаторі (рис. 7.6). На цьому принципі побудована більшість **ПВЗ** у дискретному й інтегральному виконанні.

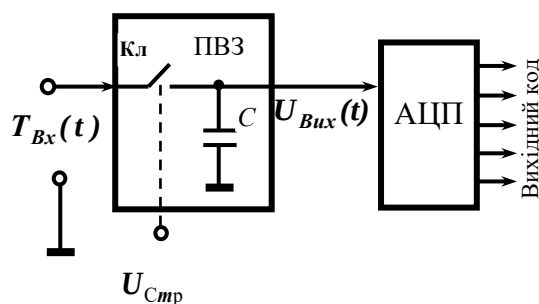


Рис. 7.6. Спрощена схема ПВЗ

Коли ключ **Кл**, керований напругою стробування $U_{стр}$, замкнути (режим вибірки), конденсатор C експонентно зі сталою часу τ_3 заряджається (або розряджається) до значення $U_{вх}(t)$. Значення сталої часу кола зарядження накопичувального конденсатора $\tau_3 = C(R_{Вих} + R_{Пр})$, де $R_{Вих}$ – вихідний опір джерела сигналу, а $R_{Пр}$ – опір замкнутого ключа, вибирається таким, щоб за час дії стробувального імпульсу напруга на конденсаторі установилася із заданою точністю. Як правило, $\tau_3 = 3 t_c$. Це досягається відповідним підбором ємності конденсатора і включенням на вході **ПВЗ** буферного підсилювача з малим вихідним опором, наприклад, повторювача напруги на **ОП**.

При розмиканні ключа (режим збереження) заряд на конденсаторі повільно змінюється через наявність струму витоку ключа і струму навантаження. Для зменшення впливу струму навантаження на спад напруги на конденсаторі у режимі збереження сигнал на навантаження передається через буферний підсилювач, який має високий вхідний і низький вихідний опір. Як буферний підсилювач можна також використовувати повторювач напруги на операційному підсилювачі (рис. 7.7).

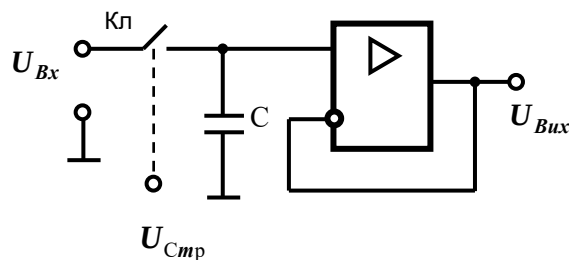


Рис. 7.7. ПВЗ із буферним каскадом

Під час дії стробуючого імпульсу генерується вибірка миттєвого значення вхідної напруги (у вигляді напруги на конденсаторі) і віднесення її до моменту закінчення імпульсу. На інтервалі між стробуючими імпульсами здійснюється збереження значення вибірки.

7.2. Цифроаналогові перетворювачі

7.2.1. Основні параметри та класифікація

Цифроаналоговий перетворювач є декодувальним пристроєм, на вхід якого надходить цифровий сигнал N , а з виходу знімається аналоговий сигнал Y , пов'язаний із вхідним співвідношенням:

$$Y = N \times q, \quad (7.1)$$

де q – чутливість (ціна молодшого розряду) перетворювача.

В інтегральних **ЦАП** вхідним сигналом, як правило, є двійковий позиційний код або побудований на його основі десятковий код. Вихідним сигналом зазвичай є напруга постійного струму. Такі **ЦАП** називаються перетворювачами коду в напругу (**ПКН**). Вихідна напруга **ПКН** відповідно до (7.1):

$$U_{Вих} = N \times U_0, \quad (7.2)$$

де U_0 – ціна молодшого розряду **ПКН**.

Число N у двійковій системі числення подається таким виразом:

$$N = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \dots + a_12^1 + a_02^0 = \sum_{s=0}^{n-1} a_i 2^i,$$

де a_i – значення розрядів вхідного коду (0 або 1);

n – кількість розрядів вхідного коду.

Тому вихідна напруга **ПКН** відповідно до (7.2):

$$U_{Вих} = \sum_{i=0}^{n-1} a_i 2^i U_0. \quad (7.3)$$

Таким чином, перетворення коду в напругу полягає в підсумуванні еталонів напруги $2^i U_0$, пропорційних вагам розрядів вхідного коду. У підсумуванні беруть участь тільки ті еталони, для яких $a_i = 1$.

При послідовному зростанні значень вхідного коду N вихідний сигнал $U_{Вих}$ утворює східчасту функцію (рис. 7.8 а). Така залежність називається статичною характеристикою перетворення $U_{Вих} = f(N)$. Вона визначає основні статичні параметри **ПКН**.

В ідеального **ПКН** усі сходинки характеристики перетворення мають однакові значення U_0 і середні точки сходинок розташовані на ідеальній прямій 1 (рис. 7.8 а). Основною характеристикою ідеального **ПКН** є роздільна здатність, яка обумовлена кількістю розрядів вхідного коду. Абсолютне значення роздільної здатності такого **ПКН** дорівнює ціні молодшого розряду:

$$U_0 = \frac{U_{Вих_{max}}}{2^n - 1}, \quad (7.4)$$

де $U_{Вих_{max}}$ – максимальна вихідна напруга перетворювача.

Зазначена роздільна здатність є потенційно можливою і характеризує тільки ідеальні **ПКН**.

Наявність інструментальних похибок вузлів реальних **ПКН** призводить до того, що їх статичні характеристики перетворення істотно відрізняються від ідеальних розмірами та формою сходинок, а також орієнтацією на координат-

ному полі (рис. 7.8 а). Це викликає похибки перетворювача і погіршення його роздільної здатності.

Для кількісного опису сумарної інструментальної похибки вводиться ряд параметрів. Абсолютна похибка перетворення в кінцевій точці шкали (похибка повної шкали) $\Delta U_{nш}$ визначає відхилення значення вихідної напруги $U_{Вих_{max}}$ від номінального (розрахункового) при максимальному вхідному коді N_{max} (рис. 7.8 а). Похибка повної шкали приводиться в довідкових даних в одиницях молодшого розряду $\delta_{nш} = \frac{\Delta U_{nш}}{U_0}$.

Похибка зміщення нуля на виході $U_{зм}$ характеризує значення $U_{Вих}$ при нульовому значенні вхідного коду (рис. 7.8 а).

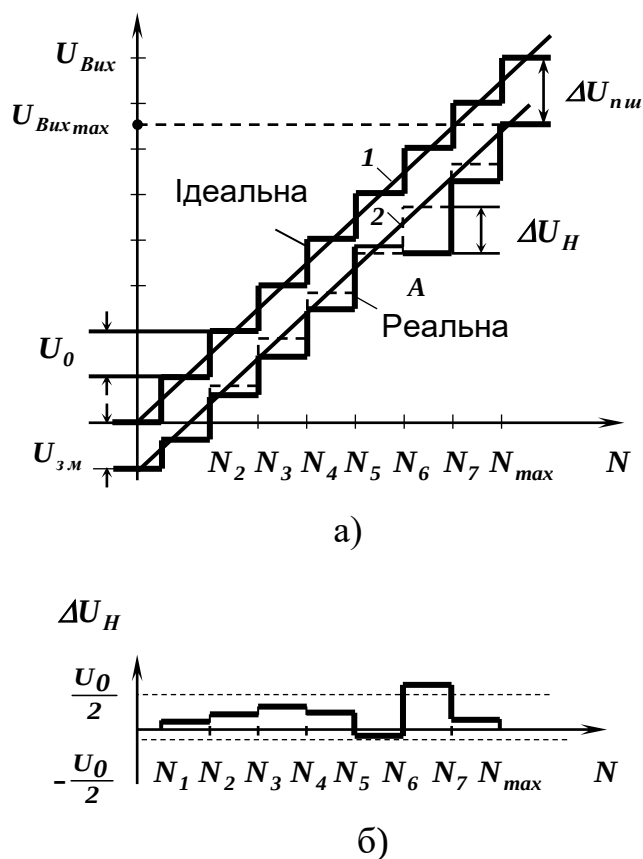


Рис. 7.8. Визначення основних статичних параметрів ПКН:

а) статичні характеристики перетворення ідеального та реального ПКН і основні похибки; б) похибка нелінійності реального ПКН

Диференціальна нелінійність δ_H характеризує відхилення різниці значень двох вихідних напруг, які відповідають двом сусіднім кодам, від ціни молодшого розряду.

Похибка нелінійності ΔU_H характеризує ідентичність збільшень (сходжень) вихідного сигналу у всьому діапазоні перетворення. Вона визначається як максимальне відхилення реальної характеристики перетворення від лінії абсолютної точності 2. Лінія проводиться через точки на характеристиці, які відповідають мініимальному та максимальному значенням вхідного коду (рис. 7.8 а)). Абсолютне значення похибки нелінійності вимірюється в одиницях молодшого розряду $\delta_H = \frac{\Delta U_H}{U_0}$.

Похибка нелінійності (рис. 7.8 б) є найбільш важливою характеристикою ПКН і, як правило, не повинна виходити за межі $\pm \frac{U_0}{2}$. В іншому випадку характеристика перетворення може бути немонотонною. Це означає, що при подачі вхідного коду, який вимагає збільшення вихідної напруги, вона (напруга) може зменшитися (ділянка А характеристики реального ПКН на рис. 7.8).

З динамічних параметрів ПКН найбільш істотними є час установлення вихідної напруги і максимальна частота перетворення.

Час установлення вихідної напруги – інтервал часу від моменту надходження вхідного коду до моменту установлення вихідної напруги з заданою точністю.

Максимальна частота перетворення – найбільша частота дискретизації (надходження вхідних кодів), за якої параметри ПКН відповідають заданим значенням.

Класифікація ПКН. Залежно від набору еталонів розрізняють два методи перетворення коду в напругу: метод підсумовування одиничних еталонів і метод підсумовування з урахуванням ваги розрядів. У першому випадку при формуванні вихідної напруги використовується еталон тільки однієї ваги U_0 , а на вхід ПКН подається число-імпульсний код. У другому випадку використовуються еталони, ваги яких пропорційні 2^i , де $i = 0, 1, 2, \dots, n - 1$, а на вхід ПКН надходить двійковий позиційний код.

За видом вхідного коду ПКН поділяються на послідовні й паралельні, на які надходять, відповідно, послідовний і паралельний двійковий коди.

За принципом функціонування розрізняють ПКН прямого перетворення і ПКН із проміжним перетворенням. У першому випадку вхідний код безпосередньо перетворюється в напругу, у другому – спочатку в проміжний сигнал (кі-

лькість імпульсів, струм, часовий інтервал), який потім перетворюється в напругу.

В інтегральній мікросхемотехніці найбільшою поширення одержали паралельні **ПН** із прямим перетворенням і підсумовуванням еталонів з урахуванням ваги розрядів. Головною перевагою таких **ПН** є висока швидкодія. Залежно від виду використовуваного джерела електричного сигналу вони поділяються на дві групи: **ПН** зі стабілізацією напруги і **ПН** зі стабілізацією струмів. Ті й інші можуть виконуватися на матриці резисторів з ваговими опорами і на матриці резисторів $R - 2R$. Як ключові елементи в **ПН** можуть використовуватися як ключі напруги, так і струму, але в сучасних інтегральних перетворювачах застосовуються більш швидкодіючі ключі та перемикачі струму.

7.2.2. ПН на матриці резисторів з ваговими опорами

Функціональна схема **ПН** наведена на рис. 7.9. Як джерело електричного сигналу в ньому використовується джерело стабілізованої напруги U_E . Перетворювач складається з n перемикачів струму ($\Pi_0 \dots \Pi_{n-1}$), матриці резисторів (опори яких “зважені” за двійковим законом від $2^0 R$ у старшому $n - 1$ розряді до $2^{n-1} R$ в молодшому (нульовому) розряді) і суматора струмів на ОП, який виконує також перетворення вихідного струму в напругу.

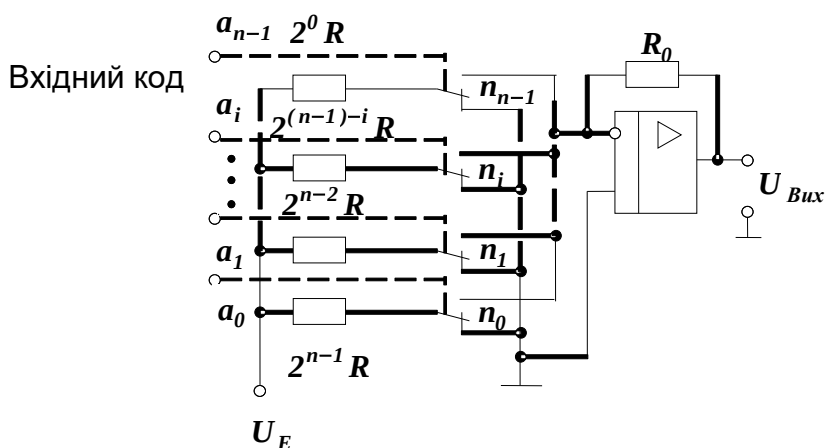


Рис. 7.9. ПН на матриці резисторів з ваговими опорами

На керуючі входи перемикачів $\Pi_0 \dots \Pi_{n-1}$ подається вхідний код, а на входи, які комутуються, – стабілізована напруга U_E (через відповідні вагові резистори матриці).

За принципом дії **ПКН** аналогічний аналоговому n -входовому суматору на **ОП**. Коефіцієнт передачі по кожному входу визначається значенням вагового резистора матриці, який підключається до інверсного входу **ОП** відповідним перемикачем.

При нульовому значенні вхідного коду ($a_i = 0$) перемикачі замикають усі вагові резистори на загальну шину й $U_{Bux} = 0$. Якщо у вхідному коді тільки $a_i = 1$, то i -й перемикач підключить до входу суматора ваговий резистор зі значенням $2^{(n-1)-i} R$, по якому буде протікати струм:

$$I_i = \frac{a_i U_E}{2^{(n-1)-i} R} \text{ при цьому } U_{Bux\ i} = I_i R_0 = \frac{a_i U_E R_0}{2^{(n-1)-i} R}.$$

У загальному випадку при довільному значенні вхідного коду:

$$U_{Bux\ i} = \sum_{i=0}^{n-1} U_{Bux\ i} = \frac{U_E R_0}{2^{n-1} R} \sum_{i=0}^{n-1} a_i 2^i = \frac{U_E R_0}{2^{n-1} R} N.$$

Як впливає з отриманого виразу, вихідна напруга **ПКН** пропорційна значенню вхідного коду N .

Коефіцієнт пропорційності визначає ціну молодшого розряду даного перетворювача $U_0 = \frac{U_E R_0}{2^{n-1} R}$, яка залежить від співвідношення опорів резистора у колі зворотного зв'язку **ОП** і максимального вагового резистора матриці. Якщо на вхід **ПКН** надходить максимальний вхідний код ($a_i = 1$ у всіх розрядах), то перемикачі $P_0 \dots P_{n-1}$ підключать усі вагові резистори до входу суматора. При цьому максимальне значення вихідної напруги

$$U_{Bux\ max} = \frac{U_E R_0}{2^{n-1} R} (2^n - 1).$$

Основним недоліком **ПКН** із ваговими опорами є необхідність використання прецизійних опорів різних номіналів, максимальний перепад яких дорівнює 2^{n-1} (від R до $2^{n-1} R$). Відношення розсіюваних потужностей на граничних вагових опорах також складає $2^{n-1} R$. При такому широкому діапазоні розсіюваних потужностей важко витримати суворо двійкові співвідношення між опорами. Тому більш широке поширення одержали **ПКН** із резистивною матрицею $R - 2R$.

7.2.3. ПКН на матриці резисторів $R - 2R$

Функціональна схема ПКН на матриці резисторів $R - 2R$ наведена на рис. 7.10. Як джерела еталонного сигналу в перетворювачі застосовані n генераторів стабільного струму I_0 .

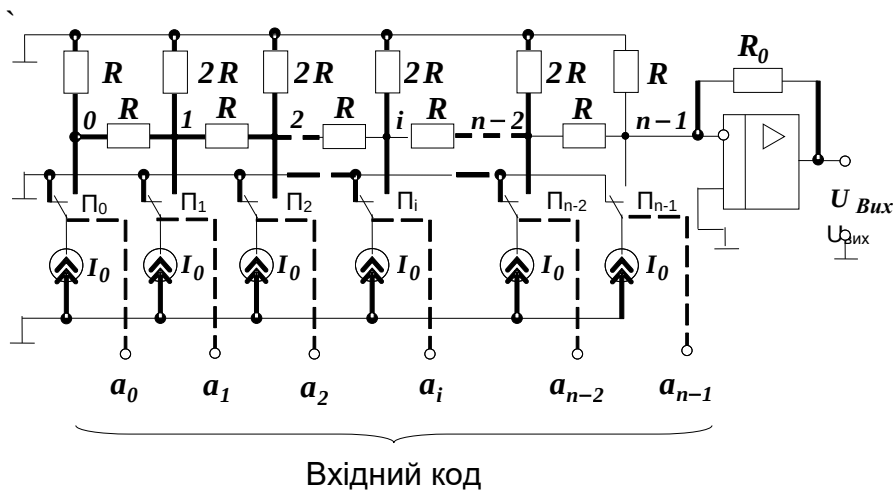


Рис. 7.10. ПКН на матриці резисторів $R-2R$

Крім них у ПКН входять n перемикачів струму, матриця резисторів $R - 2R$ і буферний підсилювач (ОП, охоплений негативним зворотним зв'язком), призначений для виключення впливу опору навантаження R_H на параметри перетворювача. Матриця резисторів $R-2R$ складається тільки з двох номіналів R та $2R$. Її особливістю є те, що спільний опір між будь-якою i -ю точкою матриці і спільною шиною дорівнює $\frac{2}{3}R$, а потенціал, прикладений (або створений) до i -ї точки, у кожній наступній праворуч (або ліворуч) точці буде зменшуватися у два рази (рис. 7.11).

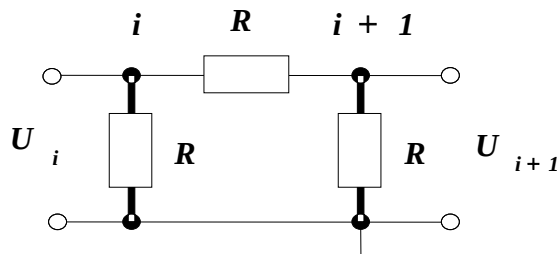


Рис. 7.11. Еквівалентна схема матриці $R - 2R$ в i -ї точці

Вхідний двійковий код надходить на перемикачі $\Pi_0 \dots \Pi_{n-1}$. Якщо у всіх розрядах $a_i = 0$, то виходи всіх стабілізаторів струму I_0 підключені до спільної шини, і вихідна напруга ПКН дорівнює нулю [8].

При наявності одиниці в i -му розряді вхідного коду перемикач Π_i підключає генератор струму I_0 до i -ї точки матриці. Оскільки внутрішній опір стабілізаторів струму значно більше опору R резисторів матриці, то, незалежно від положення інших перемикачів, опір у i -ій точці матриці дорівнює $\frac{2}{3}R$. Тому, коли замкнуті тільки i -й перемикач, у точці з номером “ i ” створюється спадання напруги $U_i = a_i \frac{2}{3}RI_0$.

Ця напруга в кожній наступній $(i+1)$ -й точці поділяється навпіл (див. рис. 7.11). При цьому на виході буферного підсилювача ПКН створюється напруга:

$$U_{Buxi} = \frac{U_i K}{2^{(n-1)-i}} = \frac{a_i 2RI_0 K}{3 \cdot 2^{(n-1)-i}},$$

де K – коефіцієнт передачі буферного підсилювача.

Оскільки матриця резисторів є лінійним колом, то у випадку, коли замкнута довільна кількість перемикачів, вихідну напругу ПКН можна обчислити як суму:

$$U_{Bux} = \sum_{i=0}^{n-1} U_{Buxi} = \frac{2}{3}RI_0 K \frac{1}{2^{n-1}} \sum_{i=0}^{n-1} a_i 2^i.$$

При використанні операційного підсилювача як буферного підсилювача (рис. 7.10), коефіцієнт передачі якого визначається відношенням опору зворотного зв'язку R_0 і вихідного опору матриці $R_M = \frac{2}{3}R$, вихідна напруга ПКН

$$U_{Bux} = \frac{I_0 R_0}{2^{n-1}} \sum_{i=0}^{n-1} a_i 2^i.$$

Ціна молодшого розряду перетворювача $U_0 = \frac{I_0 R_0}{2^{n-1}}$ практично не залежить від параметрів резисторів матриці. Опір резисторів матриці можна вибрати низькоомним. Малі опори резисторів матриці $R - 2R$ у сполученні зі швид-

кодуючими перемикачами струму дозволяють створювати **ПКН** із часом встановлення одиниці-десятки наносекунд. Матриця резисторів $R-2R$ практично легко узгоджується з інтегральною напівпровідниковою технологією й у більшості випадків є основою багаторозрядних інтегральних перетворювачів.

Точний аналіз *похибок* **ПКН** пов'язаний із громіздкими математичними викладеннями. Тому розглянемо тільки джерела сумарної статичної похибки, яка визначає основні параметри перетворювача.

Статична похибка паралельних **ПКН** обумовлюється тільки інструментальною похибкою елементів, які входять до їх складу. Відносну інструментальну похибку перетворювача можна визначити як:

$$\delta U_{Bux} = \frac{U_{Bux} - U_{Buxi}}{U_{Buxi_{max}}},$$

де U_{Bux} – фактичне значення вихідної напруги при заданому вхідному коді;

U_{Buxi} та $U_{Buxi_{max}}$ – теоретичне значення вихідної напруги ідеального **ПКН** при заданому та максимальному значеннях вхідного коду.

Джерелами інструментальної похибки є усі вузли, що входять до складу розглянутих перетворювачів. Загальна похибка:

$$\delta U_{Bux} = \delta U_E + \delta U_{Kл} + \delta U_{PM} + \delta U_{ПП},$$

де δU_E – похибка джерела еталонної напруги (струму);

$\delta U_{Kл}$ – похибка розрядних ключів;

δU_{PM} – похибка резистивної матриці;

$\delta U_{ПП}$ – похибка підсумовуючого підсилювача.

Вираз для окремих складових можна одержати шляхом аналізу конкретної схеми **ПКН**. У загальному випадку всі складові є випадковими. Тому для знаходження закону розподілу загальної похибки необхідно знати закони розподілу ймовірностей окремих похибок основних вузлів перетворювача. У більшості **ПКН**, а в інтегральних, як правило, максимальна припустима загальна похибка $\Delta U_{Bux_{max}}$ n -розрядного перетворювача має значення:

$$\Delta U_{Bux_{max}} = \pm \frac{U_0}{2} = \pm \frac{U_{Bux_{max}}}{2^{n+1}}. \quad (7.5)$$

Слід зазначити, що в деяких випадках джерело еталонного сигналу або підсумовуючий підсилювач, або обидва ці пристрої можуть не входити до складу інтегральної схеми перетворювача. Однак сумарна похибка базової напівпровідникової схеми перетворювача і зовнішніх вузлів не повинна перевищувати значення $\Delta U_{Вих_{max}}$.

7.2.4. Особливості застосування ЦАП

ЦАП широко застосовуються в цифроаналогових керуючих системах, системах контролю параметрів, системах збору й обробки даних, вимірювальній апаратурі, телевізійній техніці тощо. Впровадженню ЦАП сприяє розробка та серійний випуск інтегральних перетворювачів на біполярних транзисторах і МОН-елементах із роздільною здатністю 8...12 розрядів і часом установалення десятки наносекунд. Наприклад, швидкодіючі інтегральні ЦАП К1118ПА1, 2 і 5 мають, відповідно, кількість розрядів 8, 10, 12, а час установалення не більше 20, 30 і 100 наносекунд. Мікросхеми за входними рівняннями і напругою живлення сумісні зі схемами типу ЕЗЛ (ІС К1118ПА2 узгоджується і зі схемами ТТЛ).

Особливості застосування швидкодіючих ЦАП розглянемо на прикладі К1118ПА5. Основна схема ввімкнення ІС на узгоджений тракт із хвильовим опором 50 Ом наведена на рис. 7.12.

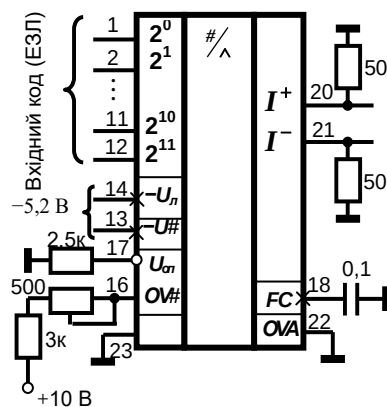


Рис. 7.12. Схема підключення швидкодіючого інтегрального ЦАП К1118ПА5

ЦАП має два струмових виходи: основний I^+ і доповняльний I^- . Вихідний струм перетворюється в напругу безпосередньо на резисторах, включених між виходами та спільною шиною. При використанні

тільки одного виходу другий вихід обов'язково підключається до спільного аналогового виводу. Установка номінального значення вихідного струму в кінцевій точці шкали здійснюється зміною опорної напруги. Опір навантаження вибирається з урахуванням того, щоб вихідна напруга ЦАП знаходилася в межах $-1... 0 \text{ В}$. При конструктивному виконанні й експлуатації вузлів на основі швидкодіючих ЦАП необхідно враховувати особливості спільної роботи ІС ЕЗЛ і прецизійних аналогових ІС.

7.3. Аналого-цифрові перетворювачі

7.3.1. Основні параметри та класифікація

АЦП є кодувальним перетворювачем, на вхід якого надходить аналоговий x_{Bx} – сигнал, а з виходу знімається цифровий код N , пов'язаний із вхідним сигналом співвідношенням:

$$N = \frac{x_{Bx}}{\Delta x},$$

де Δx – крок квантування перетворювача.

Серед АЦП найбільшого поширення одержали перетворювачі, в яких вхідним аналоговим сигналом є напруга, а вихідним цифровим сигналом – двійковий код. Такі перетворювачі називають перетворювачами напруги в код (ПНК). Відповідно до узагальненої схеми АЦП (рис. 7.2) у ПНК виконується дискретизація та квантування вхідної напруги й кодування результату перетворення. При цьому вхідній напрузі U_{Bx} ставиться у відповідність цифровий двійковий код:

$$N = \frac{U_{Bx}}{\Delta U_{K\theta}}, \quad (7.6)$$

де величина $\Delta U_{K\theta} = \frac{U_{Bx \max}}{2^n}$ є кроком квантування (квантом) n -розрядного перетворювача.

Процес перетворення напруги в код, у загальному випадку, полягає в зрівноважуванні вхідної напруги U_{Bx} сумою еталонних квантів $\Delta U_{K\theta}$, підрахунку кількості квантів, необхідних для зрівноважування і формування вихідного коду N .

Статична характеристика квантування ПНК $N = f(U_{Bx})$ на рис. 7.13 аналогічна характеристиці перетворення ПКН $U_{Bux} = f(N)$, наведеної на

рис. 7.8 а). Характеристика квантування (рис. 7.13) також визначає основні статистичні параметри **ПНК**, але стосовно вхідної напруги U_{Bx} .

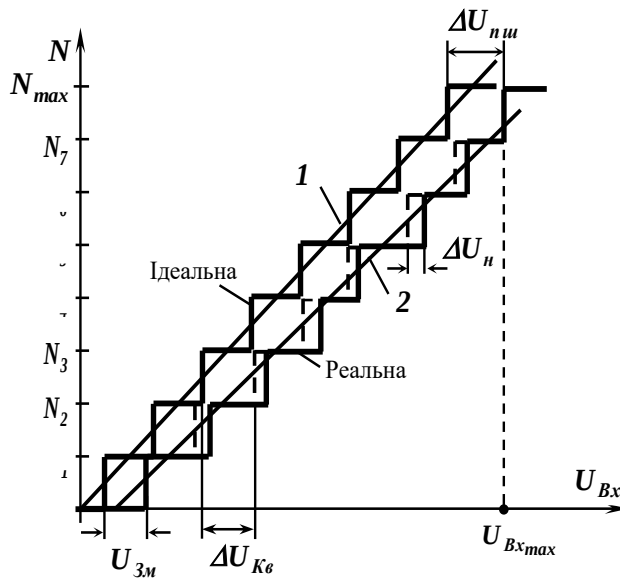


Рис. 7.13. Статистичні характеристики квантування ідеального та реального ПНК, визначення основних похибок

Основними характеристиками **ПНК** є: роздільна здатність, точність і швидкість.

Роздільна здатність ПНК визначається кількістю розрядів вихідного коду. Абсолютне значення роздільної здатності **ПНК** дорівнює кроку квантування, який жорстко пов'язаний з діапазоном зміни вхідної напруги і кількістю розрядів n -вихідного коду:

$$\Delta U_{Kв} = \frac{U_{Bx_{max}} - U_{Bx_{min}}}{2^n},$$

де $U_{Bx_{max}}$ та $U_{Bx_{min}}$ — максимальне та мінімальне значення вхідної напруги.

Крок квантування $\Delta U_{Kв}$ характеризує потенційну роздільну здатність перетворювача за рівнем, тобто найменше помітне збільшення вхідної величини. Крок квантування також визначає *методичну похибку* аналого-цифрового перетворювача, яка виникає в результаті заміни безперервної аналогової величини квантованими значеннями (найближчими рівнями квантування).

Статична точність ПНК окрім методичної похибки квантування характеризується також *загальною інструментальною похибкою* всіх елементів, які утворюють перетворювач. Основними складовими загальної інструментальної похибки є: абсолютна похибка перетворення в кінцевій точці шкали $\Delta U_{nш}$ (похибка повної шкали), похибка зміщення нуля на вході $U_{зм}$, диференціальна

похибка δU_{H_0} і похибка нелінійності δU_n . Зазначені складові інструментальної похибки визначаються за статичною характеристикою квантування $\text{ПНК } N = f(U_{Bx})$.

Абсолютна похибка перетворення в кінцевій точці шкали $\Delta U_{nш}$ – відхилення значення вхідної напруги від номінальної (розрахункової) при максимальному вихідному коді N_{max} (див. рис. 7.13). Абсолютна похибка повної шкали в довідкових даних наводиться в одиницях кроку квантування:

$$\delta_{nш} = \frac{\Delta U_{nш}}{\Delta U_{K\epsilon}}.$$

Похибка зміщення нуля на вході $U_{зм}$ – приведена до входу напруга, яка відповідає відхиленню початкової ділянки характеристики квантування від номінального значення (рис. 7.13).

Диференціальна нелінійність δ_{H_0} – відхилення різниці значень двох вхідних напруг, які відповідають середнім значенням двох сусідніх рівнів квантування, від величини кроку квантування.

Похибка нелінійності δ_n – максимальне відхилення реальної характеристики квантування від лінії абсолютної точності 2 (див. рис. 7.13). Абсолютне значення похибки нелінійності вимірюється в одиницях кроку квантування:

$$\delta_n = \frac{\Delta U_H}{\Delta U_{K\epsilon}}.$$

Похибка нелінійності інтегральних **ПНК** не повинна виходити за межі $\pm \frac{\Delta U_{K\epsilon}}{2}$. Сумарна статична похибка перетворювача не може визначатися у вигляді деякого заданого числа, тому що вона залежить від багатьох факторів, які знаходяться під впливом як часу, так і температури.

Швидкодія **ПНК** характеризується динамічними параметрами: часом перетворення $t_{пр}$ і максимальною частотою перетворення F_m .

Час перетворення $t_{пр}$ – інтервал часу від моменту заданої зміни сигналу на вході до появи на виході відповідного стійкого коду. Вибір часу перетворення пов'язаний зі швидкістю зміни вхідного сигналу. Якщо є умова, щоб за час $t_{пр}$ вхідна напруга змінювалася не більше, ніж на крок квантування $\Delta U_{K\epsilon}$, то:

$$t_{пр} \leq \frac{\Delta U_{K\epsilon}}{U'_{Bx}},$$

де U'_{Bx} – максимальна швидкість зміни вхідної напруги.

Якщо ця вимога не виконується, то на вході **ПНК** необхідно включити пристрій вибірки та збереження, який запам'ятовує вибірки вхідної напруги в моменти дискретизації на час перетворення.

Максимальна частота перетворення F_m – найбільша частота дискретизації, за якої параметри **ПНК** відповідають заданим значенням. Максимальна частота перетворення вибирається залежно від ширини спектра вхідної напруги і необхідних якісних показників системи, до складу якої входить **ПНК**.

Поряд з розглянутими характеристиками та параметрами існують параметри, які визначають зовнішні умови роботи перетворювача, такі, як: параметри вихідного коду; динамічний діапазон вхідної напруги; параметри джерел еталонної і живлячої напруги; граничні експлуатаційні параметри та ін.

*Класифікація **ПНК***. Основною ознакою класифікації **ПНК** є алгоритм його функціонування в процесі перетворення. Алгоритм відбиває комплекс операцій, за допомогою яких устанавлюється чисельна відповідність між вхідною аналоговою величиною і вихідним цифровим кодом.

За видом алгоритми перетворення **ПНК** поділяються на три основних типи:

- послідовного рахунку;
- порозрядного кодування (послідовного наближення або порозрядного зрівноважування);
- паралельні (зчитування).

За принципом перетворення **ПНК** поділяються на перетворювачі прямої дії та перетворювачі з проміжним перетворенням. У **ПНК** прямої дії вхідна напруга перетворюється безпосередньо в цифровий код. У **ПНК** із проміжним перетворенням напруга перетворюється спочатку в іншу аналогову величину (часовий інтервал, частоту тощо), яка потім перетворюється в двійковий код.

7.3.2. Перетворювачі напруги в код послідовного рахунку

У **ПНК** послідовного рахунку вхідна напруга врівноважується сумою однакових і мінімальних еталонів, які дорівнюють кроку квантування. Кількість еталонів, використовуваних для зрівноважування, підраховується лічильником, а момент рівності вхідної напруги й суми еталонів фіксується компаратором. Результатом перетворення є цифровий код лічильника.

Найбільшого поширення серед **ПНК** послідовного рахунку отримали перетворювачі, у яких еталонні напруги одержують за допомогою **ЦАП** ввімкненого в коло зворотного зв'язку. До них відносяться: **ПНК** зі східчастою напругою розгортання та слідкуючі **ПНК**. ***ПНК** зі східчастою напругою розгортан-*

ня. Функціональна схема **ПНК** і часові діаграми, які пояснюють принцип роботи, наведені на рис. 7.14.

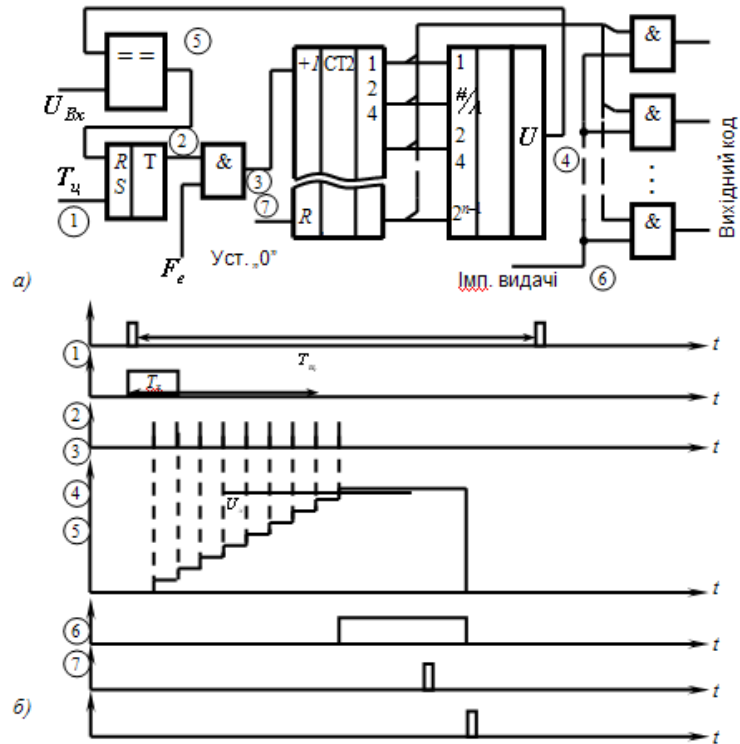


Рис. 7.14. ПНК зі східчною напругою розгортання:

а) – функціональна схема; б) – часові діаграми роботи ПНК

Імпульс запуску T_u , який визначає початок циклу перетворення, установлює **RS**-тригер в одиничний стан. Рівень логічної одиниці з виходу тригера дозволяє проходження через логічний елемент **I** імпульсів еталонної частоти на вхід лічильника. Код, що послідовно збільшується, з виходу лічильника за допомогою **ПКН** перетворюється в східчасту напругу, яка змінюється за лінійним законом. Величина сходинки дорівнює кроку квантування ΔU_{Ke} . При перевищенні вихідною напругою **ПКН** вхідної напруги U_{Bx} компаратор спрацьовує, установлює **RS**-тригер у нульовий стан і блокує проходження еталонних імпульсів на вхід лічильника. Кількість імпульсів N , записаних у лічильник, буде дорівнювати кількості квантів ΔU_{Ke} , необхідних для зрівноважування вхідної напруги:

$$N = \frac{U_{Bx}}{\Delta U_{K\theta}}.$$

Перед початком чергового циклу перетворення здійснюється видача коду споживачу й установка лічильника в нульовий стан.

Сумарна похибка перетворювача складається з методичної та інструментальної складових. Методична складова характеризується похибкою квантування і є функцією числа розрядів вихідного коду. Максимальна похибка квантування дорівнює кроку квантування $\Delta U_{K\delta}$.

Основними джерелами інструментальної похибки є компаратор і ПНК. У найгіршому випадку сумарна інструментальна похибка

$$\Delta U_{i_{\max}} = \Delta U_{Комп} + \Delta U_{Пнк},$$

а сумарна похибка перетворювача

$$\Delta U_{Пнк} = \Delta U_{K\delta} + \Delta U_{Комп} + \Delta U_{Пнк}.$$

Інструментальні похибки $\Delta U_{Комп}$ та $\Delta U_{Пнк}$ містять систематичні і випадкові складові. Систематичні складові можна компенсувати підстроюванням елементів перетворювача. Випадкові складові (дрейф, шуми тощо) необхідно враховувати, оцінюючи параметри ПНК. Інструментальні похибки ПНК характеризуються такими основними параметрами, як похибка нелінійності δU_n , похибка повної шкали $\Delta U_{nш}$ і похибка зміщення нуля $U_{зм}$. Час перетворення $t_{пр}$ ПНК послідовного рахунку мінливий і залежить від U_{Bx} . Він має максимальне значення при:

$$U_{Bx} = U_{Bx_{\max}} = (2^n - 1) \Delta U_{K\delta}.$$

$$t_{Пр_{\max}} = \frac{U_{Bx_{\max}}}{\Delta U_{K\delta}} T_e = (2^n - 1) T_e,$$

де T_e – період проходження еталонних імпульсів;

n – кількість розрядів вихідного коду.

Період $T_{ц}$ вибирається з умови: $T_{ц} \geq t_{Пр_{\max}} + t_{пнк}$, де $t_{пнк}$ – час, необхідний для видачі коду і встановлення елементів перетворювача у вихідний стан.

Перевагою ПНК послідовного рахунку є порівняльна простота і висока точність, основний недолік – низька швидкодія.

Основні сфери їхнього застосування – цифрові вольтметри та радіотехнічні системи, призначені для роботи з постійними напругами і напругами, які змінюються повільно.

ПНК послідовного рахунку, як правило, не виконуються у вигляді закінчених **ІС**. Вони легко реалізуються на основі інтегральних типових вузлів: **ЦАП**, компараторів, лічильників тощо.

ПНК послідовного рахунку зі східчастою напругою розгортання легко перетворюється в *слідкуючий ПНК*. Для цього необхідно замінити підсумовуючий лічильник на реверсивний і використовувати прямий та інверсний виходи компаратора для керування режимом рахунку. Функціональна схема слідкуючого **ПНК** наведена на рис. 7.15.

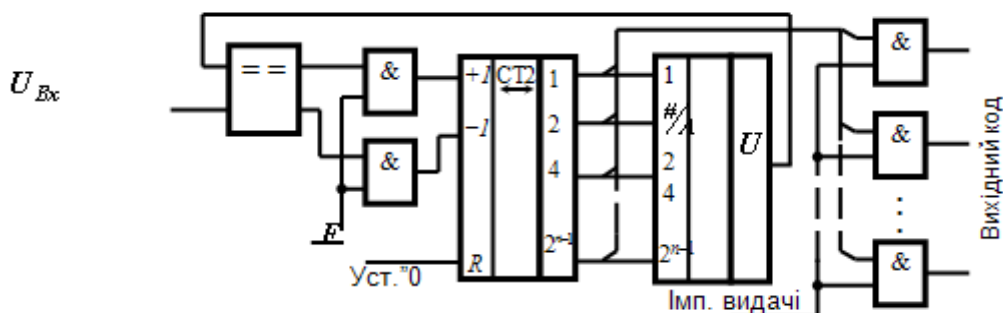


Рис. 7.15. Слідкуючий ПНК послідовного рахунку

На входи компаратора подається вхідна напруга U_{Bx} і напруга з виходу **ПКН** $U_{Пкн}$. Залежно від знака різниці між U_{Bx} і $U_{Пкн}$ імпульси еталонної частоти надходять на підсумовуючий або віднімаючий вхід лічильника. Якщо $U_{Bx} > U_{Пкн}$, то вміст лічильника буде збільшуватися, а при $U_{Bx} < U_{Пкн}$ – зменшуватися. При досягненні рівності вхідної напруги та напруги зворотного зв'язку з виходу **ПКН** у системі стеження встановлюється динамічна рівновага і вихідний код **ПНК** буде коливатися навколо відповідного вхідній напрузі середнього значення з точністю до одиниці молодшого розряду (рис. 7.16). Якщо в цьому стані вхідна напруга почне змінюватися, то вихідний код перетворювача буде відслідковувати його з похибкою, яка не перевищує одиниці молодшого розряду. Зазначене значення похибки забезпечується в тому випадку, якщо між швидкістю зміни вхідної напруги U'_{Bx} і кроком квантування виконується співвідношення:

$$U_{Bx} \cdot T_e \leq \Delta U_{Кв},$$

де T_e – період проходження еталонних імпульсів.

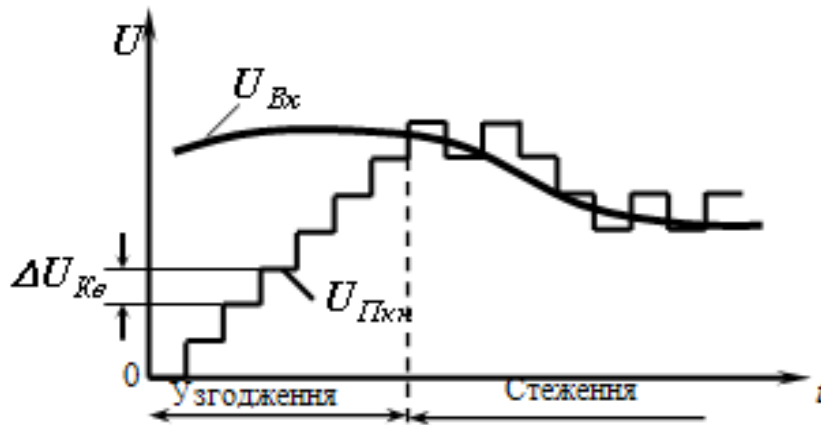


Рис. 7.16. Часові діаграми роботи слідкуючого ПНК у режимах узгодження та стеження

За умови виконання зазначеного співвідношення час перетворення слідкуючого **ПНК** дорівнює одному тактовому інтервалу T_e .

Недоліком найпростішого слідкуючого **ПНК** є великий час початкового узгодження значень $U_{Пкн}$ та $U_{Вх}$, тому що в цей час $U_{Пкн}$ змінюється від нуля до $U_{Вх}$, а перетворювач працює як **ПНК** послідовного рахунку зі східчасною напругою розгортання (див. рис. 7.16).

7.3.3. Перетворювачі напруги в код порозрядного кодування

ПНК порозрядного кодування є одним із перетворювачів, які найчастіше використовуються. Його робота заснована на виконанні послідовних порівнянь вхідної напруги з набором із n еталонів, ваги яких пропорційні степеню числа 2. Цей процес аналогічний процесу зважування, у якому невідома вхідна величина врівноважується опорною, яка сформована з набору еталонів.

Зрівноважування починається з еталона з максимальною вагою $\frac{U_E}{2^1}$, де

$U_E = U_{Вх \max}$ – максимально можлива вхідна напруга **ПНК**. Якщо $U_{Вх} < \frac{U_E}{2^1}$,

то в старшому розряді вихідного коду перетворювача фіксується нуль і далі здійснюється зрівноважування вхідної величини $U_{Вх}$ наступним за вагою еталон

лоном $\frac{U_E}{2^2}$. Якщо ж $U_{Вх} \geq \frac{U_E}{2^1}$, то в старшому розряді вихідного коду фіксується одиниця, і далі вхідна напруга врівноважується сумою еталонів $\frac{U_E}{2^1} + \frac{U_E}{2^2}$.

Аналогічні операції виконуються при визначенні значень усіх n розрядів вихідного коду.

Існує велика кількість різних схем **ПНК**, які реалізують алгоритм порозрядного кодування. Усі вони поділяються на дві групи:

виконані у вигляді замкнутих систем із **ПKN** у колі зворотного зв'язку;
побудовані у вигляді розімкнутих систем.

В інтегральному виконанні, як правило, реалізуються **ПНК** першої групи. Більш докладно розглянемо принцип роботи таких перетворювачів на прикладі 4-розрядного **ПНК**, схема якого наведена на рис. 7.17.

До складу **ПНК** входять: компаратор, **ПKN**, вихідний регістр і пристрій керування (**ПК**). До пристрою керування відносяться: регістр зсуву, регістр пам'яті на **D**-тригерах і логічні елементи **I** та **ЧИ**. В інтегральній схемотехніці пристрій керування, що містить зазначені елементи, часто виконується у вигляді окремої **ІС**, яка називається регістром послідовних наближень [9].

Для забезпечення роботи перетворювача на **ПК** надходять (або ж генеруються в ньому) такі імпульси: імпульс запуску перетворювача U_{zan} , тактові імпульси $U_{m1} \dots U_{m4}$, імпульс видачі коду U_{m5} й імпульс установки регістрів у нульовий стан U_{m6} (уст. "0"). Часові діаграми, що пояснюють функціонування **ПНК** і часове розташування тактових імпульсів, наведені на рис. 7.18.

Цикл перетворення починається з надходження імпульсу запуску U_{zan} і першого тактового імпульсу U_{m1} , що збігається з ним. За фронтом імпульсу U_{m1} в тригер старшого розряду регістра пам'яті **ПК** записується одиниця, яка надходить на вхід старшого розряду **ПKN**. На виході **ПKN** з'являється напруга $U_{Пкн} = \frac{U_E}{2}$. У компараторі вона порівнюється з вхідною напругою U_{Bx} і, якщо $U_{Bx} \geq \frac{U_E}{2}$, то вихідна напруга компаратора $U_K = U^0$. За зрізом тактового імпульсу U_{m1} в перший розряд регістра зсуву записується логічна одиниця. Вона надходить на вхід першого логічного елемента **I**, на другий вхід якого прикладена напруга логічного нуля з виходу компаратора ($U_K = U^0$). При цьому вихідна напруга логічного елемента **I** дорівнює U^0 , і в тригері *старшого значущого розряду* (**СЗР**) регістра пам'яті фіксується логічна одиниця (значення старшого розряду вихідного коду $a_3 = 1$). Якщо $U_{Bx} < \frac{U_E}{2}$, то $U_K = U^1$ і

після закінчення тактового імпульсу U_{m1} на виході першого логічного елемента І з'явиться логічна одиниця, яка встановить тригер старшого розряду регістра пам'яті в нульовий стан і зафіксує нульове значення старшого значущого розряду вихідного коду ($a_3 = 0$).

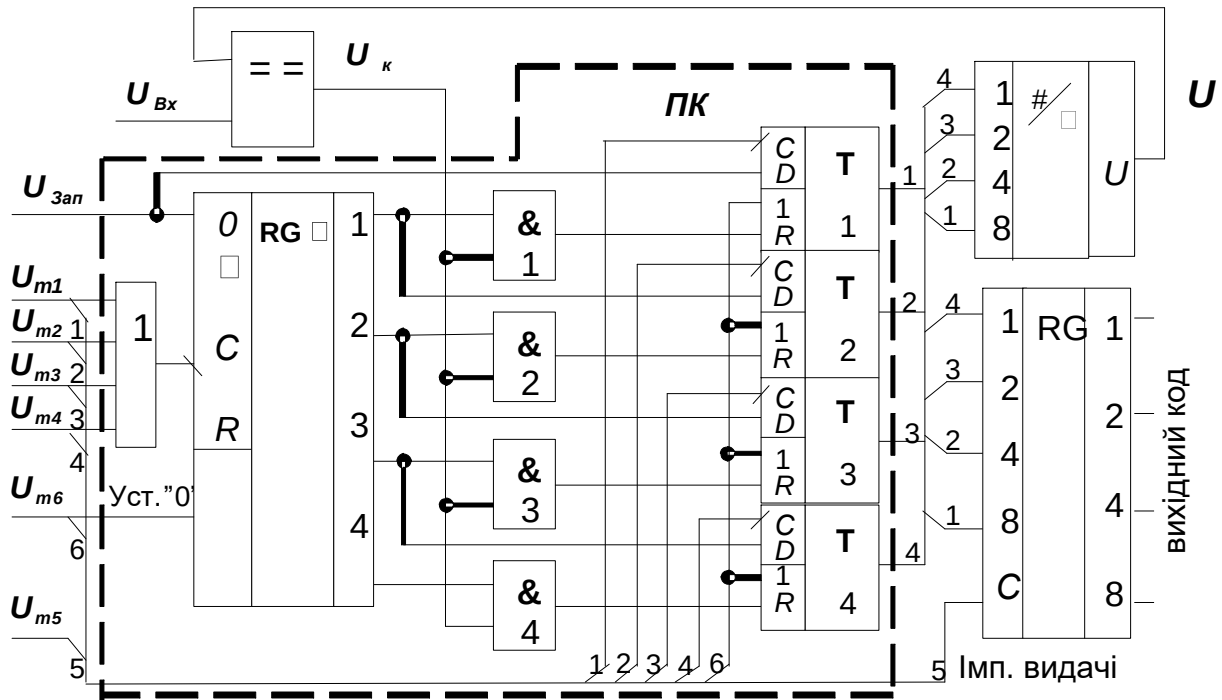


Рис. 7.17. Функціональна схема ПНК порозрядного кодування

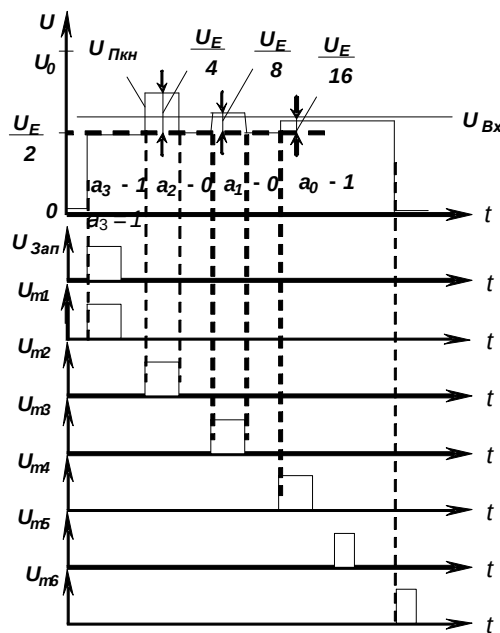


Рис.7.18. Часові діаграми функціонування ПНК порозрядного кодування

За фронтом наступного тактового імпульсу U_{m2} логічна одиниця записується в другий тригер регістра пам'яті. При цьому вихідна напруга ПКН:

$$U'_{Пкн} = a_3 \frac{U_e}{2} + \frac{U_e}{4}, \quad \text{де} \quad a_3 = \begin{cases} 1, & \text{при } U_{Bx} \geq \frac{U_E}{2} \\ 0, & \text{при } U_{Bx} < \frac{U_E}{2} \end{cases}.$$

Компаратор порівнює цю напругу з U_{Bx} .

Якщо $U_{Bx} \geq U'_{Пкн}$, то в другому тригері регістра пам'яті фіксується логічна одиниця, тобто $a_2 = 1$. При $U_{Bx} < U'_{Пкн}$ тригер встановлюється в нульовий стан ($a_2 = 0$).

У наступних двох тактах перетворення (за імпульсами U_{m3} та U_{m4}) аналогічно визначаються значення вихідного коду a_1 та a_0 . При цьому вхідна напруга послідовно врівноважується напругою з виходу ПКН із похибкою, яка не перевищує кроку квантування $\frac{U_E}{16}$. Цикл перетворення закінчується, і вихідний код $a_3 a_2 a_1 a_0$, відповідний вхідній напрузі U_{Bx} , за імпульсом видачі U_{m5} видається споживачу.

Перед початком нового циклу перетворення регістр зсуву і регістр пам'яті U_{m6} імпульсом (Уст."0") встановлюються в нульовий стан.

На рис. 7.18 показаний алгоритм функціонування розглянутого ПНК у процесі перетворення вхідної напруги, яка має значення:

$$\frac{U_E}{2} + \frac{U_E}{16} < U_{Bx} < \frac{U_E}{2} + \frac{U_E}{8}.$$

Вихідний код $a_3 a_2 a_1 a_0$, який відповідає цій напрузі, дорівнює **1001**.

Цикл перетворення n -розрядного ПНК виконується за n послідовних кроків наближення (n тактів) плюс $1...2$ службових тактів, необхідних для видачі результату перетворення й установки в нульовий стан елементів схеми. Час перетворення $t_{Пр} = nT_e + (1...2)T_e$, де T_e – період проходження тактових імпульсів U_m .

Статична інструментальна похибка перетворювачів цього типу визначається компаратором та ПКН і може бути дуже малою. Це дозволяє одержувати роздільну здатність більше ніж **16** двійкових розрядів.

Прикладом функціонально закінченого інтегрального ПНК порозрядного кодування може служити **10**-розрядний перетворювач **K1108ПВ1** (рис. 7.19).

Час перетворення – не більше **0,9мкс**, вихідні рівні відповідають **ТТЛС**. До складу ПНК входить ЦАП, джерело опорної напруги, тактовий генератор, регістр послідовних наближень і вихідний регістр, який забезпечує збереження інформації протягом наступного циклу перетворення. У ПНК передбачена можливість роботи в наступних режимах: **8**-розрядному режимі перетворення (укорочений цикл) і в **10**-розрядному режимі; із внутрішньою і зовнішньою опорною напругою; із внутрішнім тактуванням та із синхронізацією від системи перетворення й обробки інформації.

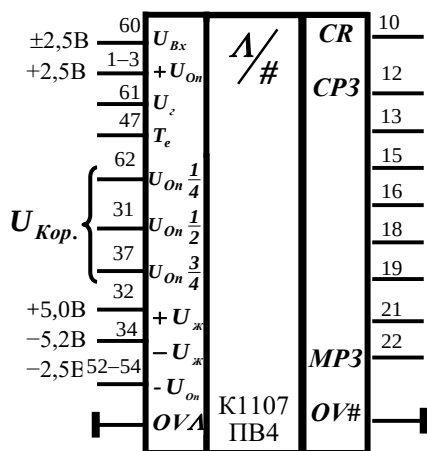


Рис. 7.19. 10-розрядний інтегральний ПНК порозрядного кодування 572ПВ1

В **ІСК 1108ПВ1** передбачена можливість коригування нелінійності характеристики квантування, для цього на виводи 31, 37 і 62 подаються рівні напруги, відповідно, $\frac{1}{4}$, $\frac{1}{2}$ і $\frac{3}{4}$ від U_{On} .

ПНК порозрядного кодування становить визначений технічний компроміс між точністю і швидкодією. Маючи високу точність і час перетворення порядку одиниць-десятих мікросекунд, вони знаходять широке застосування як при побудові цифрових вимірювальних приладів, так і в різних сферах цифрової обробки сигналів. Для цифрових вимірювальних приладів розроблені **АЦП** порозрядного кодування з роздільною здатністю до 22 бітів. Для перетворення в цифрову форму звукових сигналів потрібні 16-розрядні **АЦП**, для телевізійних сигналів достатні 8-розрядні, але високої швидкодії.

Для систем керування використовуються 10 – 14-розрядні перетворювачі, як правило, порозрядного кодування. Усі ці перетворювачі використовують ті

або інші послідовні методи перетворення, що обмежує їх швидкодію. Максимальну швидкодію забезпечує паралельний метод перетворення.

7.3.4. Паралельні перетворювачі напруги в код

Спрощена схема паралельного ПНК наведена на рис. 7.20.

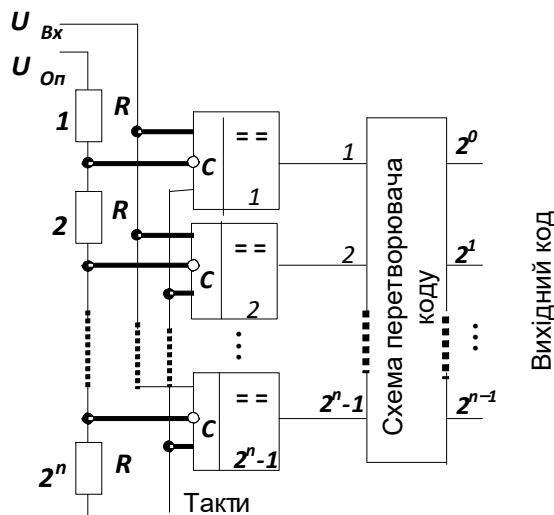


Рис.7.20. Паралельний ПНК

Основними елементами перетворювача є $2^n - 1$ компараторів напруги. На один із входів кожного компаратора подається своя опорна напруга. Індивідуальна опорна напруга для кожного компаратора формується резистивним дільником з еталонної напруги $U_E = U_{Bx_{max}}$. Мінімальна опорна напруга має значення $\frac{U_E}{2^n}$, максимальна напруга – $\frac{U_E}{2^n}(2^n - 1)$. Різниця між опорними напругами двох найближчих компараторів дорівнює кроку квантування $U_{Kv} = \frac{U_E}{2^n}$. Інші входи компараторів об'єднані і на них подається вхідний сигнал U_{Bx} . За функціонуванням і структурою компаратори напруги в паралельних ПНК аналогічні компараторам, які стробуються, з убудованим тригером-замком, наприклад, 597СА1, 2.

Вхідна напруга U_{Bx} одночасно за допомогою $2^n - 1$ компараторів порівнюється з набором опорних напруг (еталонів). На виходах компараторів установлюються логічні рівні, які відповідають співвідношенню сигналів на їхніх входах у момент приходу фронту тактового імпульсу (імпульсу стробу). Трива-

лість тактового імпульсу складає одиниці наносекунд. Після закінчення тактового імпульсу результат порівняння миттєвого значення U_{Bx} з набором еталонів зберігається на виходах компараторів до приходу наступного тактового імпульсу. Стан виходів компараторів становить $(2^n - 1)$ -розрядне слово унітарного коду. Для перетворення цього коду в двійковий n -розрядний використовуються логічні кодувальні схеми, наприклад, пріоритетні шифратори, постійні запам'ятовуючі пристрої тощо.

Оскільки кожна з $2^n - 1$ градацій вхідного сигналу оцифровується окремим компаратором, то час перетворення в такому ПНК визначається лише часом перемикавання компаратора та логічної кодувальної схеми, а отже, він мінімально можливий. Однак висока швидкодія паралельних ПНК досягається за рахунок значних апаратних витрат. Їх обсяг практично подвоюється з кожним новим розрядом. Звичайна кількість розрядів паралельних перетворювачів не перевищує **8...10**.

В інтегральному виконанні паралельні ПНК випускаються в складі серії **1107**. ІСК **1107ПВ1** і **К1107ПВ2** становить 6- та 8-розрядні ПНК, вони містять відповідно **63** та **255** компараторів, ділянки опорної напруги, шифратори, буферні і вихідні регістри, схеми перетворення і керування вихідним кодом. Вихідні рівні погоджені з ТТЛС. Час перетворення не більше **0,1** мкс. Результат перетворення вхідної напруги, яка змінюється в діапазоні від -2 до 0 В, може подаватися прямим, зворотним і доповняльним кодом. Тип вихідного коду задається сигналами керування вихідним кодом.

Для перетворення широкосмугових аналогових сигналів у двійковий код з рівнями ЕЗЛ розроблені ВІС **К1107ПВ3**, **К1107ПВ4** і **К1107ПВ5**, які становлять, відповідно, 6-, 8- і 10-розрядні ПНК із часом перетворення **20** нс і максимальною частотою перетворення **100 МГц**. В ІС ЕЗЛ призначені для перетворення сигналів, які швидко змінюються у часі у цифрову форму у вигляді двійкового прямого коду в телебаченні, радіолокації, телеметрії й інших системах обробки сигналів. У перетворювачах передбачена можливість нарощування розрядності шляхом паралельного з'єднання декількох ІС.

На рис. 7.21 наведена спрощена схема 12-розрядного паралельно-послідовного АЦП, побудованого на основі двох однакових 6-розрядних паралельних перетворювачів.

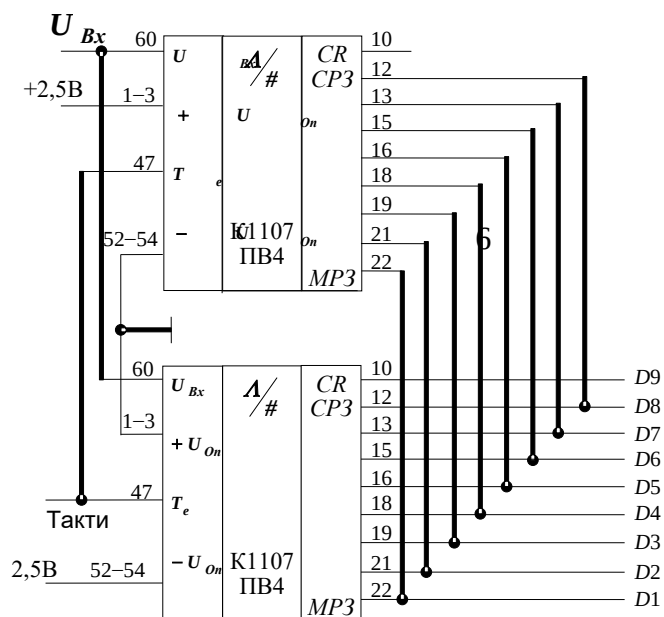


Рис. 7.21. Знання ІС Л1107ПВ4 в 9-розрядному ПНК

Вхідний аналоговий сигнал надходить на перший 6-розрядний перетворювач, за допомогою якого генерується код шести старших розрядів **7..12**. Цей код за допомогою перетворювача коду в напругу перетворюється в напругу U_1 , пропорційну вазі шести старших розрядів вихідного коду. Напруга U_1 віднімається від вхідної напруги і різниця $\Delta U = U_{Bx} - U_1$ збільшується на 2^{n_1} , де n_1 – кількість старших розрядів. У даному прикладі різниця ΔU збільшується на 64, тому що перший паралельний ПНК визначає 6 старших розрядів. Для віднімання та множення використовуються швидкодіючі операційні підсилювачі. Вихідна напруга підсилювача $\Delta U \cdot 2^{n_1}$ надходить на другий паралельний ПНК, на виході якого формуються значення шести молодших розрядів (**1...6**) вихідного коду.

Паралельно-послідовні ПНК становлять компроміс між обсягом устаткування та швидкодією. На даний час існує велика кількість варіантів схемної реалізації АЦП, у яких використовується паралельно-послідовний принцип перетворення (рис. 7.22). Швидкодія розглянутого ПНК буде визначатися сумою часів перетворення двох паралельних ПНК, ПКН і часу встановлення ОП.

На закінчення розгляду основних типів ПНК необхідно надати їм порівняльну оцінку за критеріями: час перетворення, точність і обсяг устаткування.

більшою мірою це відноситься до таких параметрів, як швидкодія і точність, між якими існує складний і суперечливий взаємозв'язок. Так, прецизійні АЦП одержали свою високу точність за рахунок відмовлення від швидкодії, а швидкодіючі – шляхом обміну точності на швидкодію, про що свідчить невелика кількість розрядів паралельних АЦП. Розв'язання протиріччя точність – швидкодія є основною задачею, яка стоїть перед розроблювачами перетворювачів форми інформації.

В останні роки зусилля розробників концентруються за напрямками:

1. Підвищення швидкодії за рахунок удосконалення існуючих і пошуку нових схемотехнічних рішень, модернізації технологічних процесів, зменшення розмірів окремих компонентів, оптимізації характеристик окремих вузлів перетворювачів, застосування вихідного матеріалу з високою рухливістю носіїв заряду. Підвищення швидкодії перетворювачів стало можливим завдяки успіхам біполярної напівпровідникової технології надвисокої інтеграції. У даний час серійно випускаються 10- і більше розрядні АЦП і ЦАП з часом перетворення до одиниць наносекунд, які містять кілька сотень тисяч еквівалентних інтегральних елементів, розміщених на одному кристалі. Варто думати, що підвищення ступеня інтеграції ІС перетворювачів забезпечить умови для подальшого підвищення точності при збереженні їх високої швидкодії.

2. Збільшення точності перетворювачів (збільшення розрядності ЦАП і АЦП) при одночасному зниженні потужності споживаної на один біт інформації. Воно досягається як упровадженням нових схемотехнічних рішень, так і за рахунок застосування перспективних матеріалів і технологій.

3. Розширення функціональних можливостей перетворювачів за рахунок збільшення схемотехнічної і конструктивної складності, включення в один кристал із самим перетворювачем схем з'єднання з мікропроцесорами, джерел опорної напруги й операційних підсилювачів. При широкому впровадженні мікропроцесорів у системи обробки інформації необхідно, щоб АЦП і ЦАП були не тільки сумісні з ними, але і, у перспективі, органічно ввійшли до складу мікропроцесорних комплектів або, ще краще, виконувалися на єдиному, кристалі. Це дозволяє використовувати мікропроцесори для виконання логічних функцій АЦП, що звичайно реалізуються його цифровою частиною. Одержали поширення з'єднані з мікропроцесором системи збору даних, виконані в одному корпусі, які по суті є багатоканальними програмно-керованими АЦП і ЦАП.

4. Перспективним напрямком у розвитку перетворювачів інформації є розробка функціональних АЦП і ЦАП, які дозволяють одночасно з перетворенням форми подання сигналу робити його функціональне перетворення відпо-

відно до заданої залежності. Прикладом функціональних ЦАП є перетворювач **K572ПА1**, що перемножує, у якому немає внутрішнього джерела опорної напруги. Використання таких ЦАП у складі АЦП дозволяє виконувати перетворення аналогового сигналу з визначеним функціональним зв'язком між вихідним кодом і еталонною напругою.

5. Важливим напрямком розвитку перетворювачів є поліпшення їх експлуатаційних параметрів: температурної стабільності, розширення робочого діапазону температур, надійності і довговічності.

6. Останнім часом намітилася тенденція розробки АЦП і ЦАП, призначених для виконання нетрадиційних для них функцій. Наприклад, генерування і формування сигналів різної форми, фільтрації, затримки інформації тощо.

КОНТРОЛЬНІ ПИТАННЯ

1. У чому полягає аналого-цифрове перетворення сигналів? Пояснити сутність операцій дискретизації, квантування та кодування.

2. Пояснити суть цифроаналогового перетворення сигналів.

3. Яку функцію виконують пристрої вибірки та збереження аналогових сигналів? Пояснити принцип роботи такого пристрою.

4. Навести узагальнену структурну схему АЦП та пояснити функціонування.

5. Пояснити природу виникнення шуму квантування.

6. Пояснити принцип роботи пристрою вибірки та збереження.

7. У чому полягають переваги та недоліки ЦАП на основі матриці з ваговими опорами?

8. Розрахувати максимальне значення вихідної напруги чотирьохрозрядного ЦАП на основі матриці з ваговими опорами, в якому використовується стабільна напруга 5 В , мінімальний номінал резистора складає 1 кОм , а опір резистора зворотного зв'язку ОП складає 5 кОм .

9. Розрахувати максимальне значення вихідної напруги чотирьохрозрядного ЦАП на основі матриці $R - 2R$, в якому використовується джерело струму 100 мА , а опір резистора зворотного зв'язку ОП складає 1 кОм .

10. Навести класифікацію АЦП.

11. Пояснити принцип роботи АЦП послідовного рахунку. Охарактеризувати переваги та недоліки АЦП такого типу.

12. Пояснити принцип роботи АЦП порозрядного кодування.

13. Пояснити принцип роботи паралельних АЦП.

14. Надати порівняльну характеристику АЦП різних видів.

8. СХЕМОТЕХНІКА КОМБІНАТОРНИХ ВУЗЛІВ

8.1. Комбінаторні вузли

8.1.1. Кодування чисел для виконання арифметичних операцій

Будь-яке число становить набір цифр (код) у визначеній системі числення. Найбільше поширення отримали позиційні системи числення, в яких кожна цифра виражає число не тільки своїм значенням, але і положенням (позицією) серед інших цифр. Основа системи – це число q , яке дорівнює кількості різних цифр, необхідних для вираження всіх чисел у межах одного розряду. Позитивне ціле число з n розрядів у позиційній системі з основою q може бути подане як:

$$A_q = \sum_{i=0}^{n-1} a_i q^i = a_{n-1} q^{n-1} + a_{n-2} q^{n-2} + \dots + a_1 q^1 + a_0 q^0, \quad (8.1)$$

де a – будь-яка цифра від 0 до $q-1$ включно;

n – кількість розрядів числа A (перший член суми становить старший розряд числа, а останній – молодший).

Кодоване в десятковій системі числення число, наприклад, 397 подається як $397_{10} = 3 \cdot 10^2 + 9 \cdot 10^1 + 7 \cdot 10^0$.

У цифровій техніці переважно застосовують елементи з двома робочими станами, один з яких ототожнюється з цифрою 1, а другий – з 0. Тому тут поширене так зване натуральне двійкове кодування, при якому n -розрядне число виражається як:

$$A_2 = \sum_{i=0}^{n-1} a_i 2^i, \quad (8.2)$$

де a_i може приймати значення 1 або 0. Наприклад, число 13_{10} при двійковому кодуванні має вигляд 1101_2 .

Хоча числа в двійковій системі й виглядають дуже громіздкими, арифметичні дії над ними дуже спрощуються. Додавання, наприклад, можна звести до чотирьох елементарних дій: $0+0=0$; $0+1=1$; $1+0=1$; $1+1=10$. Результат останньої дії означає перенесення одиниці у старший розряд. Двійкове множення складається також з чотирьох елементарних дій: $0 \cdot 0=0$; $1 \cdot 0=0$; $0 \cdot 1=0$; $1 \cdot 1=1$.

Поряд з цим знаходять застосування й інші коди, що дають змогу спростити арифметичні дії. До них відносяться, зокрема, обернений і доповняльний коди.

У загальному випадку існує необхідність записувати додатні та від'ємні числа. З цією метою вводиться додатковий *знаковий розряд* z , який зазвичай розташовується перед числовими розрядами (для зручності сприйняття відокремлюється крапкою). Для додатних чисел значення знакового розряду $z = 0$, для від'ємних чисел $z = 1$. Додатне число A , яке має модуль $|a_{n-1}...a_1a_0|$ в прямому, оберненому та доповняльному коді, має однаковий вигляд $A_{np} = A_{ob} = A_{don} = 0.a_{n-1}...a_1a_0$. Від'ємне число, яке має той же модуль, в оберненому коді має вигляд $A_{ob} = \bar{A} = 1.\overline{a_{n-1}}...\overline{a_1}\overline{a_0}$, а в доповняльному – вигляд $A_{don} = \bar{A} + 1 = 1.\overline{a_{n-1}}...\overline{a_1}\overline{a_0} + 1$ (додається 1 в молодший розряд).

Широко застосовується двійково-десятковий код, в якому цифри кожного розряду десятичного числа подаються чотирирозрядним двійковим числом (тетрадою). Так, число $A_{10} = 397_{10}$ у двійково-десятковому коді має вигляд $A_{2-10} = 0011\ 1001\ 0111$. Основна позитивна якість двійково-десятичного коду полягає в простоті взаємного переведення десятичних і двійкових чисел апаратними засобами. Головний недолік – громіздкість запису і надмірність, тому що шість двійкових комбінацій у кожній тетраді (від $1010_2 = 10_{10}$ до $1111_2 = 15_{10}$) не використовуються [5].

Останнього недоліку немає у двійково-шістнадцятковому кодуванні. Тут усі 16 двійкових комбінацій тетрад використовуються. Вони відповідають числам у шістнадцятковій системі числення, які позначаються від 0 до 9 десятковими цифрами, а від 10 до 15 – буквами A, B, C, D, E, F . Наприклад, число $A_{10} = 397_{10}$ у шістнадцятковому коді має вигляд $18D_{16} (1 \cdot 16^2 + 8 \cdot 16^1 + 13 \cdot 16^0)$, а в двійково-шістнадцятковому – $0001\ 1000\ 1101_{2-16}$. Як бачимо, тут тетради відображають числа 1, 8 і D шістнадцяткової системи числення, а в сукупності становлять звичайний двійковий код.

У табл. 8.1 наведені приклади запису чисел у двійковій, десятичній і шістнадцятковій системах числення.

Як бачимо з табл. 8.1, у двійковій системі числення кожне наступне число одержується додаванням одиниці до молодшого розряду, причому сума нуля й

одиниці дає одиницю, а сума двох одиниць дає нуль і перенесення одиниці в старший розряд. У старшому розряді знову додається цифра розряду та перенесена одиниця і т. д. Така операція називається “сума за модулем 2”

Таблиця 8.1

Приклади запису чисел у різних системах числення

2	10	16
<i>0000</i>	<i>0</i>	<i>0</i>
<i>0001</i>	<i>1</i>	<i>1</i>
<i>0010</i>	<i>2</i>	<i>2</i>
<i>0011</i>	<i>3</i>	<i>3</i>
<i>0100</i>	<i>4</i>	<i>4</i>
<i>0101</i>	<i>5</i>	<i>5</i>
<i>0110</i>	<i>6</i>	<i>6</i>
<i>0111</i>	<i>7</i>	<i>7</i>
<i>1000</i>	<i>8</i>	<i>8</i>
<i>1001</i>	<i>9</i>	<i>9</i>
<i>1010</i>	<i>10</i>	<i>A</i>
<i>1011</i>	<i>11</i>	<i>B</i>
<i>1100</i>	<i>12</i>	<i>C</i>
<i>1101</i>	<i>13</i>	<i>D</i>
<i>1110</i>	<i>14</i>	<i>E</i>
<i>1111</i>	<i>15</i>	<i>F</i>

Переведення чисел із системи числення з вільною основою q у десяткову систему числення ($q = 10$) виконується за наведеними вище формулами, для чого потрібно перевести в десяткову систему числення числа a_i та q . Дещо складніше перевести числа з десяткової системи числення в систему числення з основою $q \neq 10$. Найпростіше така операція виконується для $q = 2, 8, 16$.

Нехай потрібно перевести число 1993 з десяткової у шістнадцяткову систему числення. Переведення здійснюється послідовним діленням десяткового числа на основу системи числення $q = 16$.

Переведення з десяткової в двійкову систему числення здійснюється послідовним діленням десяткового числа на основу системи числення $q = 2$.

Приклад 8.1.

$$\begin{array}{r}
 1993 \quad 16 \\
 - \quad 16 \quad | \quad 124 \quad | \quad 16 \\
 \hline
 39 \quad | \quad 112 \quad | \quad 7 \quad \leftarrow \cdot 16^2 \\
 - \quad 32 \quad | \quad 12 \quad \leftarrow \cdot 16^1 \\
 \hline
 73 \\
 - \quad 64 \\
 \hline
 9 \quad \leftarrow 16^0
 \end{array}
 \left. \vphantom{\begin{array}{r} 1993 \\ 124 \\ 112 \\ 73 \\ 64 \\ 9 \end{array}} \right\} 7 \cdot 16^2 + 12 \cdot 16^1 + 9 \cdot 16^0 = 1993_{10} = 7C9_{16}$$

8.1.2. Виконання операцій над двійковими числами

При виконанні арифметичних операцій над двійковими числами (які ще мають назву “числове слово” або “операнд”) необхідно враховувати знак числа.

Правила виконання арифметичних операцій над двійковими числами:

1. Усі операції виконуються в оберненому або доповняльному кодах і результат отримується відповідно в оберненому або доповняльному коді.

2. Як і в десятковій системі числення, операція додавання двох n -розрядних чисел $A\{a_{n-1}...a_1a_0\}$ та $B\{b_{n-1}...b_1b_0\}$ виконується порозрядно, починаючи з молодшого. Розрядні числа a_i та b_i додаються “за модулем 2” з урахуванням переносу з попереднього (молодшого) розряду P_{i-1} . Сума $S_i = 1$, якщо кількість доданків зі значенням 1 непарна, а перенос у наступний розряд $P_{i-1} = 1$, якщо кількість таких доданків не менше двох. Якщо в результаті додавання в знаковому розряді утвориться перенос, то ця одиниця додається до молодшого розряду суми (приклад 8.2). Таке додавання одиниці називають *циклічним переносом*.

3. Операцію віднімання в цифровій техніці реалізують за допомогою додавання. Для цього від’ємник інвертується (враховуючи знаковий розряд), а знак “-” замінюється на “+” (приклад 8.3).

Недоліком використання оберненого коду є утворення циклічного переносу, який призводить до повторення операції додавання, що значно збільшує час виконання дій.

При використанні доповняльного коду дії додавання і віднімання виконуються так само, як з використанням оберненого коду, але простіше і швидше, тому що відпадає необхідність у циклічному переносі зі знакового розряду. Однак перетворення числа у доповняльний код технічно реалізується складніше, ніж в обернений.

Приклад 8.2

Знак	Число
$A_2 = 0.$	$01101 = A_{10} = 13$
+	
$\overline{B}_2 = 1.$	$11000 = B_{10} = -7$
	10.00101
+	
	$S_2 = 0.00110 = S_{10} = 6$

Циклічний перенос (додавання)
одиниці знакового розряду до

Приклад 8.3

Знак	Число	
$\overline{A}_2 = 1.$	$101101 = A_{10} = -18$	числа
—		
$\overline{B}_2 = 1.$	$111010 = B_{10} = -5$	
$\overline{A}_2 = 1.$	101101	
+		
$B_2 = 0.$	$000101 = B_{10} = 5$	← Переведене число $\overline{B}_2$
$\overline{S}_2 = 1.$	$110010 = S_{10} = -13$	до прямого коду

Множення двійкових багаторозрядних чисел у принципі не відрізняється від множення десяткових чисел, тому що окремий добуток багаторозрядного числа на **1** дорівнює цьому числу, а множення на **0** дає нулі у всіх розрядах. Дія множення зводиться, таким чином, до операцій зсуву і додавання окремих добутків. Очевидно, що при цьому можна опустити проміжний результат множення на нуль і просто зсунути вліво наступний проміжний добуток (приклад 8.4).

Двійкове ділення теж аналогічне десятковому, але завдяки меншій кількості цифр у системі числення дія виходить більш простою. Це видно з прикладу 8.5 (ділення $101101_2 = 45_{10}$ на $1001_2 = 9_{10}$).

Приклад 8.4

$$\begin{array}{r}
 A_2 = 1\ 0\ 0\ 1 = A_{10} = 9 \\
 \times \\
 B_2 = 1\ 0\ 1 = B_{10} = 5 \\
 \hline
 1\ 0\ 0\ 1 \\
 + \\
 1\ 0\ 0\ 1 \\
 \hline
 R_2 = 1\ 0\ 1\ 1\ 0\ 1 = R_{10} = 45
 \end{array}$$

Приклад 8.5

$$\begin{array}{r}
 10101 \overline{) 1001} \\
 - 1001 \\
 \hline
 1001 \\
 - 1001 \\
 \hline
 1001 \\
 - 1001 \\
 \hline
 0000
 \end{array}
 \quad
 101 \leftarrow R_2 = R_{10} = 5$$

Ділення зводиться до дій зсуву і віднімання.

Розглянуті вище арифметичні дії відносяться до дій з двійковими числами, поданими у формі з *фіксованою комою*, коли число відображається у вигляді єдиного цілого, кожен його розряд має незмінну вагу. На практиці з метою розширення діапазону чисел використовують форму з *комою, що плаває*. Тут число складається з двох частин: *мантиси* m , яка містить значення цифри числа, і *порядку* p , який показує степінь, до якого треба піднести основу q , щоб отримане при цьому число, помножене на мантису, давало істинне значення числа, яке подається:

$$A_q = m q^p. \quad (8.3)$$

Арифметичні дії над числами з комою, що плаває, реалізуються дещо складніше, ніж над числами з фіксованою комою, тому що тут необхідно виконувати операції як над мантисами, так і над порядком.

8.1.3. Реалізація множення та ділення

Базовою операцією при організації арифметичного множення чисел в цифровій техніці є арифметичне додавання, яке можна виконати за допомогою суматорів. На рис 8. 1 показана схема для множення двох дворазрядних чисел A та B .

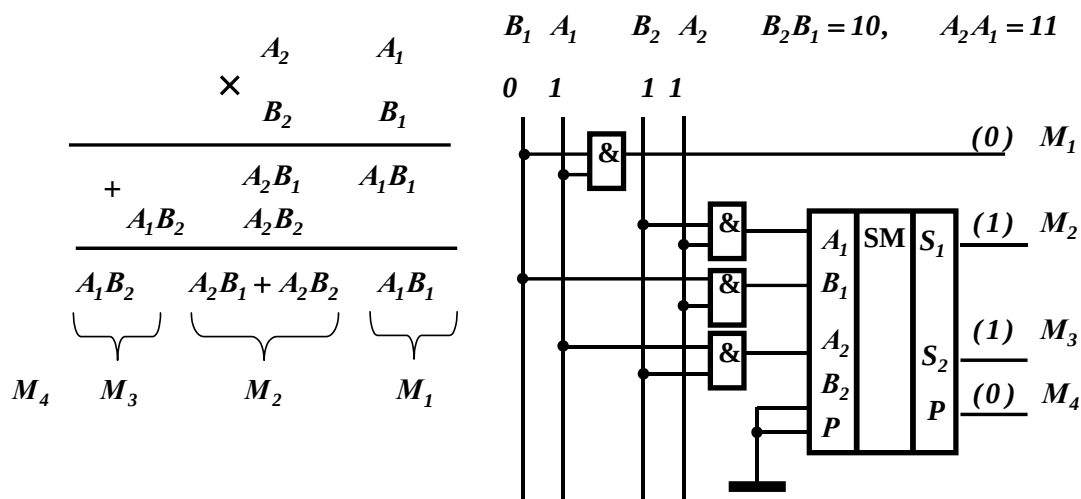


Рис. 8.1. Схема множення двох дворазрядних чисел

Чотирьохразрядний результат на виході формується за рахунок паралельного множення множеного на кожний розряд множника логічними елементами **НІ-І** і складання проміжних результатів із зсуванням на один розряд суматором. При цьому виконуються дії, що зображені на рис.8.1. Тут M_3 – біт (1 або 0) переносу з попереднього розряду. Використання логічних елементів **І** в даному випадку закономірно, оскільки в рамках одного розряду і арифметичне множення, і логічне (кон'юнкція) підкоряються однаковим правилам [1].

Цифри в дужках біля виводів мікросхем відносяться до прикладу множення двох чисел $A = 11$, і $B = 10$. Послідовність дій така:

У випадку необхідності розрядність такого множителя може бути збільшена нарощуванням.

$$\begin{array}{r}
 \begin{array}{cc} 1 & 1 \\ \times & \\ \hline 1 & 0 \\ 0 & 0 \\ 1 & 1 \\ \hline 1 & 1 & 0 \end{array}
 \end{array}
 \left. \begin{array}{l} \\ \\ \end{array} \right\} \begin{array}{l} \text{Проміжні результати} \\ \text{Кінцевий результат} \end{array}$$

Ділення апаратними засобами складніше, ніж інші арифметичні дії. Тут доводиться враховувати і те, що мають місце випадки, коли результат є ірраціональним число і його потрібно округляти.

Крім того, ділення вимагає більше часу. Для ділення часто використовують методи, основані на послідовному зменшенні дільника з діленого за допомогою зменшувача (субстрактора). В ході кожного зменшування на виході субстрактора з'являється імпульс, який фіксується лічильником часного.

Приклад використання лічильників для ділення двох чисел A / B ($A > B$) наведений на рис. 8.2. Тут A – постійне або програмуємо число, а B – змінне. Операція виконується шляхом циклічного зменшення B імпульсів з безперервної послідовності, яка вміщує A імпульсів. У процесі кожного такого зменшення утворюється один з імпульсів часного, яке складається лічильником часного.

Число, яке ділиться, забезпечується лічильником-дільником **DD1**, який має $K_{\text{діл}} = A$. У Вихідному положенні він запертий сигналом $Q = 0$ з виходу тригера **DD3**. Лічильник дільника **DD2** працює у режимі зворотного рахування (вхід " ± 1 " = 0). До подачі імпульсу "пуск" в лічильник через елемент **DD4.2** записується число B . Пусковий імпульс включає лічильники **DD1** і **DD2**. В момент обнуління лічильника **DD2** імпульс з його виходу P вдруге введе число B . Цей же імпульс поступить на лічильник часного (на схемі не показаний). Процес періодично повторюється до появи стопового імпульсу на виході лічильника **DD1**. З киданням тригера **DD3** операція зупиняється.

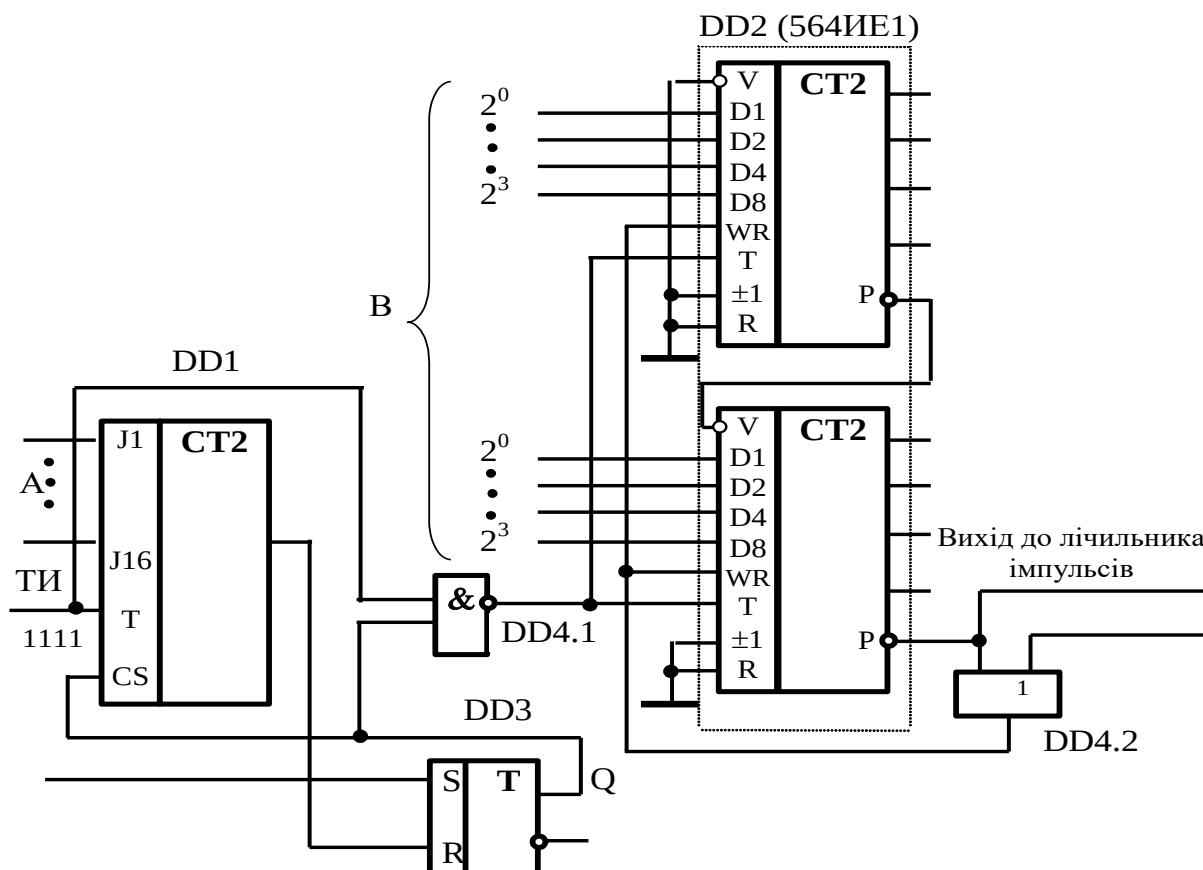


Рис. 8.2. Виконання операції ділення за допомогою лічильників, що зменшують

Абсолютна похибка при виконанні операції рівняється залишку від ділення. Тривалість операції $t = \frac{A}{f_{Bx}}$, де f_{Bx} – частота вхідних тактових імпульсів.

8.1.4. Арифметико-логічні пристрої

Цікаві властивості притаманні спеціалізованим мікросхемам, які виконують відповідно до програми на входах арифметичні та логічні перетворення двійкової інформації. Ці мікросхеми так і називають – *арифметико-логічні пристрої* (АЛП). Порівняно з приборами, які працюють за жорсткою, наперед заданою логікою, АЛП становить пристрій більш високого класу. В комп'ютерній схемотехніці АЛП є базовим елементом.

АЛП дорожче простих мікросхем, але завдяки універсальним властивостям використання їх в апаратурі в багатьох випадках виявляється доцільним.

Арифметико-логічним пристроєм (АЛП) називається цифровий пристрій комбінаційного типу, призначений для виконання логічних, арифметичних операцій і операцій зрушення над вхідними операндами, поданими двійковим кодом.

Умовне графічне позначення АЛП наведене на рис. 8.3. Для позначення функціонального призначення використовується аббревіатура **ALU** (*Arithmetic Logic Unit*).

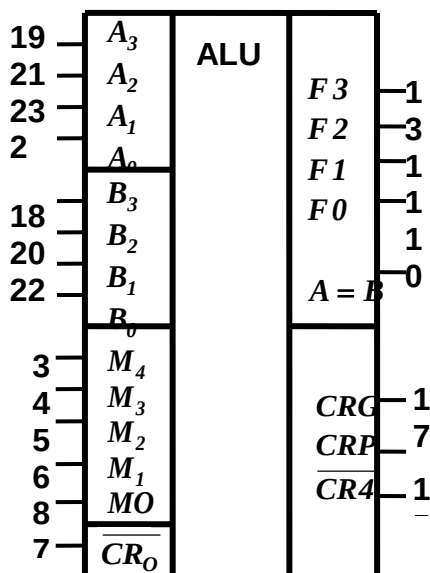


Рис. 8.3. Умовне графічне позначення арифметико-логічного пристрою K155ИПЗ

Вибір виконуваної операції здійснюється через входи $M_4 \dots M_0$.

Операції виконуються над вхідними операндами, заданими на входах: $A_3 \dots A_0$ – входи чотирьохрозрядного операнда A ; $B_3 \dots B_0$ – входи чотирьохрозрядного операнда B .

Для каскадування декількох мікросхем з метою нарощування розрядності передбачається можливість використання значення переносу з молодших розрядів CR_0 (вхід $\overline{CR_0}$).

Результат виконання заданої операції одержують на виходах $F_3 \dots F_0$. При виконанні заданої операції формується значення переносу CR_4 (вихід

$\overline{CR4}$), результат порівняння значень операндів A і B – $A = B$ (вихід $A = B$), і значення групових переносів CRG і CRP (виходи CRG і CRP). Значення групових переносів використовують при каскадуванні декількох ІС.

Залежно від типу АЛП розрядність вхідних операндів, перелік виконуваних над ними операцій а також вихідні сигнали можуть бути різними.

Прикладом побудови АЛП є ІС К155ІПЗ (рис. 8.3). Вона становить 4-розрядний АЛП.

Операція задається кодом операції на входах $M_4 \dots M_0$ (див. табл. 8.2).

Таблиця 8.2

Таблиця функціонування АЛП К155ІПЗ

Код операції				Логічна операція	Арифметична операція
M_4	M_3	M_2	M_1	$MO = 1$	$MO = 0$
0	0	0	0	$\overline{A}$	$A + \overline{CR_0}$
0	0	0	1	$\overline{A \vee B}$	$(A + B) + \overline{CR_0}$
0	0	1	0	$\overline{A \wedge B}$	$(A + \overline{B}) + \overline{CR_0}$
0	0	1	1	0	$-1 + \overline{CR_0}$
0	1	0	0	$\overline{A \wedge B}$	$A + (A \cdot \overline{B}) + \overline{CR_0}$
0	1	0	1	$\overline{B}$	$(A + B) + (A \cdot \overline{B}) + \overline{CR_0}$
0	1	1	0	$A \oplus B$	$A - B - 1 + \overline{CR_0}$
0	1	1	1	$A \wedge \overline{B}$	$(A \cdot \overline{B}) - 1 + \overline{CR_0}$
1	0	0	0	$\overline{A \vee B}$	$A + (A \cdot B) + \overline{CR_0}$
1	0	0	1	$\overline{A \oplus B}$	$A + B + \overline{CR_0}$
1	0	1	0	B	$(A + \overline{B}) + (A \cdot B) + \overline{CR_0}$
1	0	1	1	$A \wedge B$	$(A \cdot B) - 1 + \overline{CR_0}$
1	1	0	0	1	$A + A + \overline{CR_0}$
1	1	0	1	$A \vee \overline{B}$	$(A + B) + A + \overline{CR_0}$
1	1	1	0	$A \vee B$	$(A + \overline{B}) + A + \overline{CR_0}$
1	1	1	1	A	$A - 1 + \overline{CR_0}$

Причому молодший розряд (M_0) задає тип операції: логічна або арифметична. Розходження між ними полягає в тому, що логічні операції виконуються

порозрядно, без урахування переносів з молодших розрядів у старші, а арифметичні операції переноси враховують.

Функціональна побудова 0-го розряду АЛП показана на рис. 8.4.

Як впливає з функціональної схеми, для виконання операцій можуть бути обрані значення операнда **A** або пряме чи інверсне значення операнда **B**. Інверсне значення нульового розряду операнда **B** отримують за допомогою інвертора **DD1**.

Вибір виконуваної операції здійснюється за допомогою завдання відповідного стану на лініях керування вибором операції $M_4 \dots M_1$. Кожна лінія керує “власним” логічним елементом **DD3...DD6**, на який подаються значення операндів у різних співвідношеннях прямих або інверсних значень.

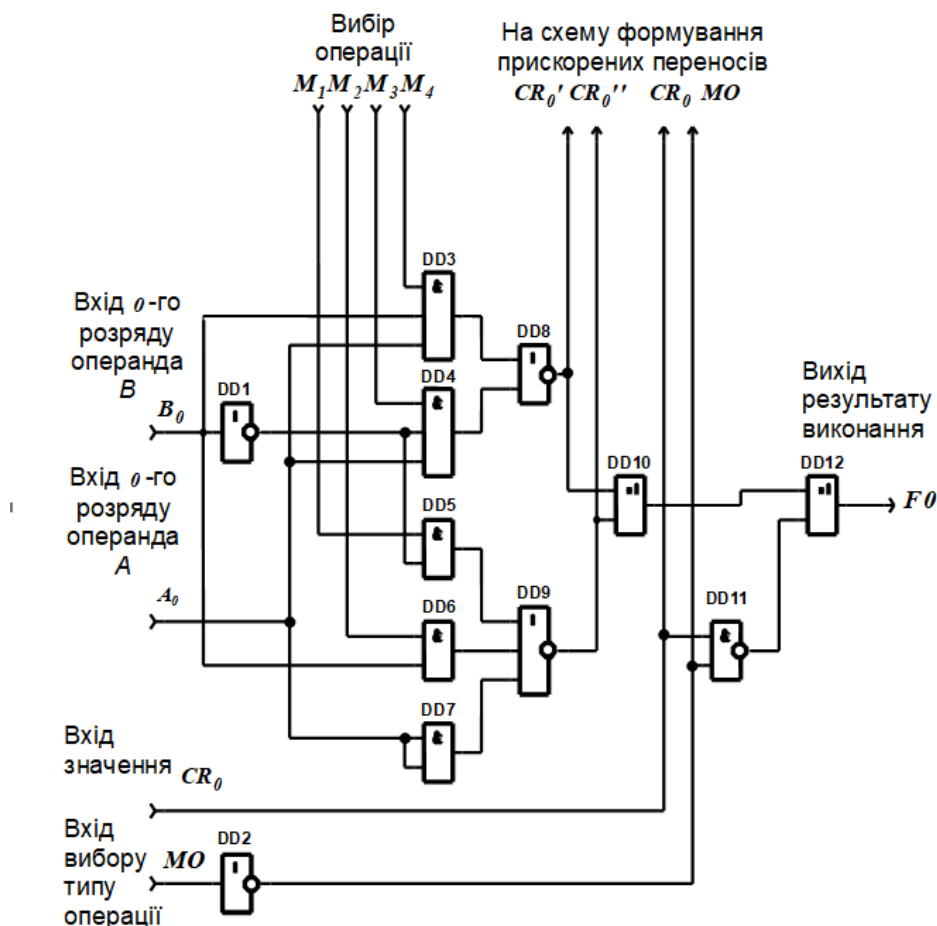


Рис. 8.4. Функціональна схема побудови 9-го розряду АЛП

Вихідні значення виконаних операцій поєднуються логічними елементами **НІ-ЧИ**, (**DD8, DD9**) і логічним елементом **ЧИ, ЩО ВИКЛЮЧАЄ** (**DD10**).

Урахування значення переносу здійснюється за допомогою логічного елемента **DD12**.

Побудова старших розрядів **АЛП** відрізняється наявністю схем формування значень прискореного переносу.

За допомогою логічного елемента **DD11** здійснюється керування обліком значення переносу від попередніх розрядів або типу виконуваної операції (логічна – без урахування переносу, арифметична – з урахуванням значення переносу від молодшого розряду) [7].

Поточні значення вихідних функцій (CR_0' , CR_0'' , CR_0 , M_0) передаються на схеми формування значень старших розрядів. Для цього використовуються схеми формування прискорених переносів.

8.2. Таймери

Таймером називається електронний пристрій, призначений для формування імпульсних сигналів з тривалістю та *скважністю*, що регулюється. Під дане визначення підпадають як відповідні вузли цифрових пристроїв, так і спеціалізовані **ІС**, що використовуються для розробки різного роду пристроїв, здатних задавати часові інтервали.

Усі існуючі на сьогоднішній день таймери можна розбити на два класи: однотактні та багатотактні з вбудованим лічильником.

Однотактні таймери призначені для формування часових інтервалів тривалістю від одиниць мікросекунд до одиниць години. За своєю суттю вони становлять комбінацію аналогової частини (компаратора) з цифровою схемою. Можливий варіант структурної схеми такого пристрою показаний на рис. 8.5.

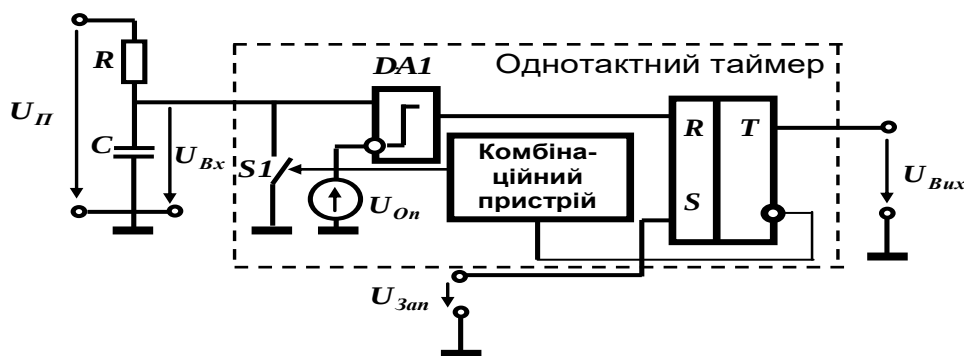


Рис. 8.5. Структурна схема однотактного таймера

Тривалість часового інтервалу, що формується таким пристроєм, визначається параметрами зовнішнього RC -ланцюга, який задає величину часового інтервалу. За активного значення сигналу U_{3an} RS -тригер встановлюються в

одиничний стан, що приводить до розмикання ключа **S1**. Починається заряд конденсатора **C** зовнішнього ланцюга, що задає величину часового інтервалу. У момент, коли напруга на конденсаторі досягає рівня опорної напруги U_{on} , відбувається спрацьовування компаратора **DA1**, і його вихідний сигнал скидає RS-тригер. Ключ **S** при цьому замикається і конденсатор **C** розряджається.

Однотактний таймер, побудований за описаною схемою, може формувати на виході лише поодинокі імпульси. Для забезпечення можливості формування послідовності імпульсів схема пристрою повинна бути доповнена другим компаратором.

Багатотактні таймери або *таймери з вбудованими лічильниками* розроблені для формування імпульсів наднизької частоти з тривалістю імпульсу до декількох десятків годин. У свою чергу, їх можна розбити на дві підгрупи: таймери, що програмуються, тобто такі, у яких часовий інтервал задається програмним способом. У простішому випадку це здійснюється установкою на виводах лічильника зовнішніх перемичок спеціалізованих таймерів, лічильник яких має жорстко заданий коефіцієнт перерахунку.

Структурна схема багатотактного таймера (рис. 8.6) зазвичай містить однотактний таймер і двійковий лічильник, спільна робота яких організовується додатковим логічним блоком.

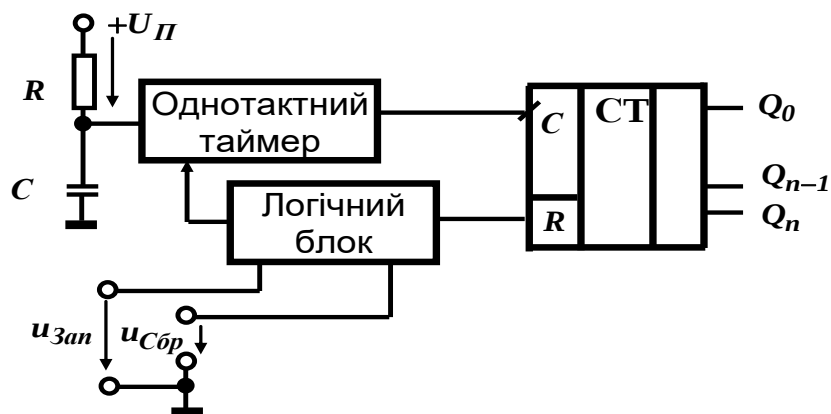


Рис. 8.6. Структурна схема багатотактного таймера

У багатотактному таймері фактично відбувається множення постійної часу зовнішнього **RS**-ланцюга на модуль рахунку лічильника **СТ**. При подачі сигналу, що запускає $u_{зан}$, включається мультивібратор, виконаний на однотактному таймері. Його вихідні імпульси надходять на рахунковий вхід лічильника. На виходах останнього може бути сформовано декілька послідовностей імпуль-

сів з періодом від T_i до $(2^N - 1)T_i$, де T_i – період імпульсів, що знімаються з виходу одноктактного таймера; N – кількість тригерів у лічильнику СТ. Далі розглянемо тільки схеми, побудовані з використанням одноктактних таймерів.

Таймер може працювати як у режимі очікування, так і в автоколивальному режимі. В основі принципу роботи таймера покладено зарядження (розрядження) конденсатора. Як елементи, що фіксують досягнення заданих рівнів зарядження (розрядження), використовуються компаратори. Приклад схеми таймера наведений на рис. 8.7 а).

У таймері резистори $R1, R2, R3$ створюють опорні напруги U_{O1} і U_{O2} для компараторів КН1 і КН2. На другі входи компараторів подаються відповідно напруга з конденсатора C , який задає час, і сигнал запуску. Результати порівняння вхідних напруг фіксуються RS-тригером і, після посилення транзисторами $VT1$ і $VT3$, видаються на вихід таймера у вигляді логічних рівнів. Транзистор $VT1$ – це ключ, який служить для швидкого зарядження конденсатора на етапі відновлення. Резистор R_i конденсатор C зовнішні.

Рис. 8.7 ілюструє роботу таймера в очікувальному режимі.

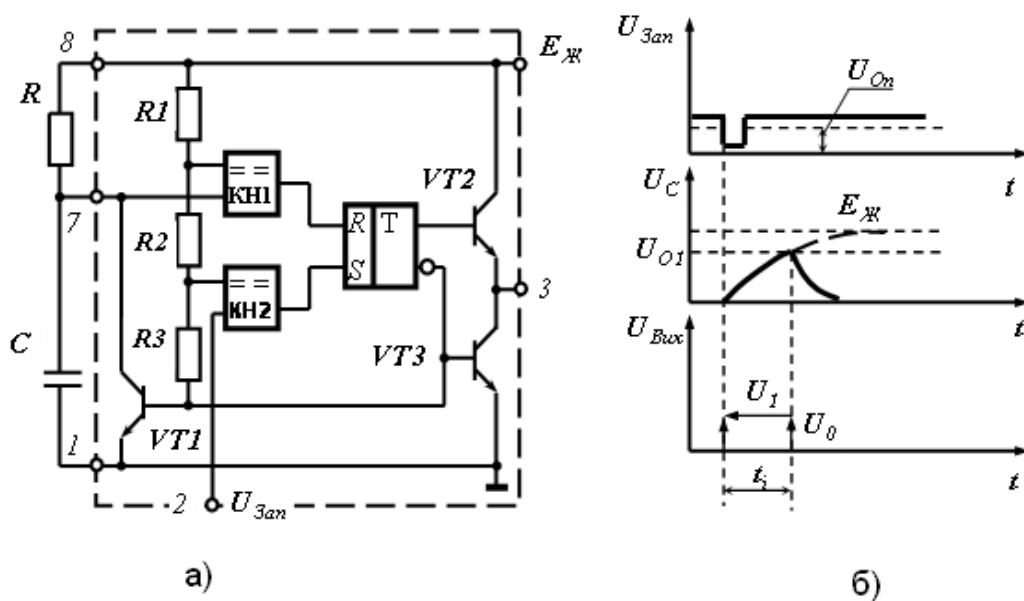


Рис. 8.7. Схема (а) та часові діаграми (б) таймера в режимі формувача

У вихідному стані на вхід 2 таймера подана напруга, яка перевищує U_{O2} . На виходах обох компараторів логічні нулі. RS-тригер встановлений у нульовий стан. Транзистор $VT2$ закритий, а транзистори $VT1$ і $VT3$ відкриті й насичені.

Конденсатор заряджений ($U_{C0} \approx 0$) на виході таймера $U_{Вих} = U^0 \approx 0$. При подачі імпульсу запуску негативної полярності вихідна напруга компаратора **КН2** відповідає логічній одиниці, і RS-тригер переводиться в одиничний стан. Транзистори перемикаються. На виході таймера $U_{Вих} = U^1$. Конденсатор починає заряджатися від джерела живлення $E_{Ж}$ через резистор R . Коли напруга на ньому досягає рівня U_{O1} , компаратор **КН1** спрацьовує, у результаті чого RS-тригер повертається у вихідний стан.

Час заряду конденсатора визначається наступним виразом (див. рис. 8.7 б):

$$t_i = RC \ln \frac{E_{жс}}{E_{жс} - U_{O1}} = RC \ln \frac{1}{1 - \gamma}, \quad (8.4)$$

де $\gamma = \frac{R2 + R3}{R1 + R2 + R3}$.

Прийнято вважати оптимальним $\gamma = 0,66$. При цьому найкраще задовольняються суперечливі вимоги до повноти використання експоненти U_C і достатності її крутості, від якої залежить похибка компаратора.

На рис. 8.8 зображена схема ввімкнення таймера в автоколивальний режим і графічна ілюстрація його роботи.

Нехай у момент часу $t = 0$ починається заряджання конденсатора через резистори R' і R'' . Доки напруга на виводі 7, яка складається з напруги U_C на конденсаторі і резисторі R'' , менша опорної $U_{O1} = 0,66 E_{Ж}$, компаратор **КН1** увімкнений і на його виході U^0 .

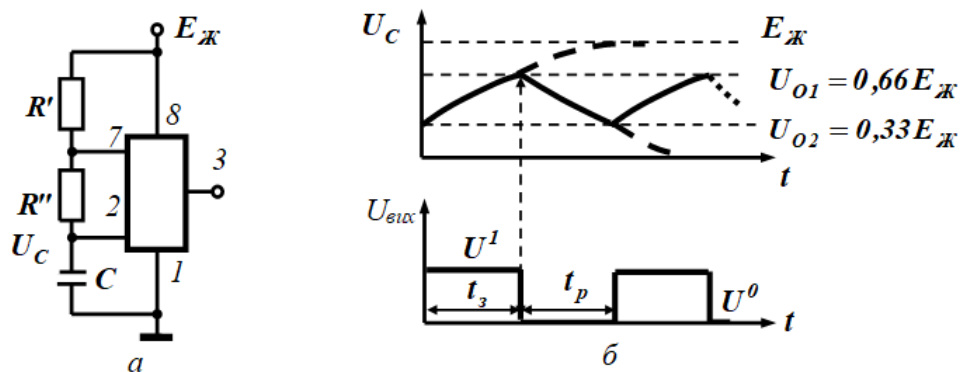


Рис. 8.8. Схема (а) та часові діаграми (б) таймера у режимі генератора імпульсів

Напруга на вході 2 таймера вища опорної $U_{O2} = 0,33E_{ж}$, тому на виході **КН2** теж логічний нуль. Тригер в одиничному стані, на виході таймера $U_{Вих} = U^1$. Коли напруга на виході 7 досягає рівня U_{O1} , компаратор **КН1** спрацьовує і встановлює тригер у нульовий стан. На виході таймера $U_{Вих} = U^0$. Конденсатор починає розряджатися через R'' і відкритий **VT1**. У момент порівняння U_C з опорною напругою U_{O2} компаратор **КН2** перемикає таймер у вихідний стан. Далі процеси повторюються.

З рис. 8.8 б) випливає, що час зарядження конденсатора

$$t_z = (R' + R'')C \ln \frac{E_{ж} - 0,33E_{ж}}{E_{ж} - 0,66E_{ж}} \approx 0,7(R' + R'')C,$$

час розрядження:

$$t_p = R''C \ln \frac{0,66E_{ж}}{0,33E_{ж}} \approx 0,7R''C.$$

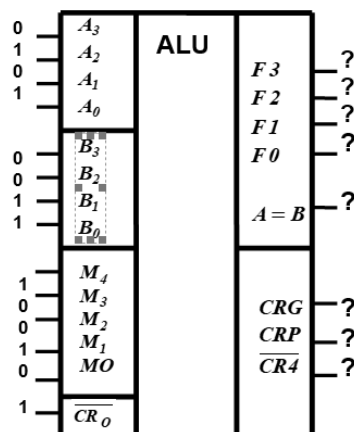
період коливань:

$$T = t_z + t_p \approx 0,7(R' + 2R'')C.$$

Ці формули дозволяють розраховувати параметри імпульсів для автоколивального режиму таймера.

КОНТРОЛЬНІ ПИТАННЯ

1. Подати число $[169]_{10}$ у двійковій, вісімковій, шістнадцятковій та двійково-десятковій формі.
2. Дати характеристику арифметико-логічному пристрою.
3. Пояснити призначення входів та виходів арифметико-логічного пристрою **К155ИПЗ**.
4. Розставити значення вихідних сигналів на виходах АЛП **К155ИПЗ**.



5. Виконати за допомогою **АЛПК 155ИПЗ** (розставити відповідні значення вхідних та вихідних сигналів) логічну операцію заперечення диз'юнкції (**ИІ-І**).

6. За яких умов на виході $\overline{CR4}$ мікросхеми **К155ИПЗ** з'явиться сигнал логічної одиниці?

7. Яким чином на базі мікросхем **АЛП К155ИПЗ** можна зробити нарощування розрядності? Запропонувати схему для виконання операцій над 8-розрядними операндами.

8. Дати визначення таймеру.

9. Пояснити принцип роботи однотоктного та багатотоктного таймера. Навести їх структурні схеми.

10. Навести співвідношення для розрахунку періоду імпульсів таймера.

9. ПРОГРАМУВАЛЬНІ ЛОГІЧНІ ІНТЕГРАЛЬНІ СХЕМИ (ПЛІС)

9.1. Загальні відомості

У даний час програмувальні логічні інтегральні схеми – зручна в освоєнні та застосуванні елементна база, альтернативи якій часто немає. Доказом перспективності нової елементної бази є зростання частки **ПЛІС** у загальному обсязі мікросхем, які випускаються, щорічна поява нових поколінь, які мають більш досконалу архітектуру.

Наведемо класифікацію **ПЛІС** за структурною ознакою, тому що вона дає найбільше повне уявлення про клас задач, придатних для вирішення тією або іншою **ПЛІС**. Основними критеріями такої класифікації є наявність, вид і способи комутації елементів логічних матриць. За цією ознакою можна виділити такі класи **ПЛІС**.

1. *Програмувальні логічні матриці* – найбільш традиційний тип **ПЛІС**, що має програмувальні матриці **I** та **ЧИ** (Field Programmable Logic Array, FPLA і Field Programmable Logic Sequencers, FPLS). Недолік такої архітектури – слабе використання ресурсів програмувальної матриці **ЧИ**, тому подальший розвиток одержали мікросхеми, побудовані за архітектурою програмувальної матричної логіки (**ПМЛ**, **Programmable Array Logic, PAL**) – це **ПЛІС**, які мають програмувальну матрицю **I** та фіксовану матрицю **ЧИ**. До цього класу відносяться більшість сучасних **ПЛІС** невеликого ступеня інтеграції.

2. *Програмувальна макрологіка*. Вона містить єдину програмувальну матрицю **НІ-I** або **НІ-ЧИ**, але за рахунок численних інверсних зворотних зв'язків здатна формувати складні логічні функції.

Дві перелічені вище архітектури **ПЛІС** містять невелику кількість чарунків і застосовуються для реалізації відносно простих пристроїв, для яких не існує готових **ІС** середнього ступеня інтеграції. Для реалізації алгоритмів, наприклад, цифрової обробки сигналів (**ЦОС**), вони непридатні.

3. *Програмувальна матрична логіка (**Programmable Logic Device, PLD**)* має архітектуру досить зручну для реалізації цифрових автоматів. Розвиток цієї архітектури – програмувальні матричні блоки, які комутуються. Це **ПЛІС**, яка містить кілька матричних логічних блоків (**МЛБ**), об'єднаних комутаційною матрицею. Кожен **МЛБ** становить структуру типу **ПМЛ**, тобто програмувальну матрицю **I**, фіксовану матрицю **ЧИ** та макрочарунку – **Complex Programmable Logic Devices (CPLD)**.

4. *Програмувальні вентильні матриці (ПВМ)* – складаються з логічних блоків (ЛБ) і комутуючих зв’язків (програмувальних матриць з’єднань). Логічні блоки таких ПЛІС складаються з одного або декількох відносно простих логічних елементів. В основі цих блоків лежить таблиця перекодування (**Look-up-table, LUT**), програмувальний мультиплексор, **D**-тригер, а також коло керування. Таких простих елементів може бути досить велика кількість (у сучасних ПЛІС ємністю до 1 мільйона вентилів кількість логічних елементів досягає декількох десятків тисяч). За рахунок такої великої кількості логічних елементів вони містять значну кількість тригерів, також деякі сімейства ПЛІС мають убудовані модулі пам’яті, які реконфігуруються (**РМП, Embedded Array Block, EAB**). Це робить ПЛІС даної архітектури досить зручним засобом реалізації алгоритмів цифрової обробки сигналів, основними операціями в яких є перемноження, множення на константу, підсумовування і затримка сигналу (**Field Programmable Gate Array, FPGA**).

Подальший розвиток архітектур йде шляхом створення комбінованих архітектур, які поєднують зручність реалізації алгоритмів цифрової обробки сигналів (**FPGA**) зі зручностями реалізації цифрових автоматів (**CPLD**).

Зростання рівня інтеграції дало можливість розміщувати на кристалі схеми, складність яких відповідає цілим системам, так звані “системи на кристалі” (**System-on-Chip, SOC**).

Особливістю сучасних ПЛІС є можливість тестування вузлів за допомогою порту **JTAG** (без фізичного доступу до кожного виводу мікро-схеми).

Розглянемо приклади різних сучасних архітектур ПЛІС на конкретних зразках мікросхем.

9.2. ПЛІС типу CPLD

У даний час випускають ПЛІС сімейства **MAX7000, MAX7000A, MAX7000B, MAX7000E, MAX7000S**. ПЛІС сімейства **MAX7000S** є подальшим розвитком сімейства **MAX7000**, які допускають можливість програмування в системі.

У табл. 9.1 наведені основні характеристики ПЛІС сімейства **MAX7000S**.

Усі ПЛІС сімейства **MAX7000S** підтримують технологію програмування в системі **ISP (In-System Programmability)** і периферійного сканування (**JTAG**). Матриця з’єднань має безперервну структуру, яка дозволяє реалізувати час затримки поширення сигналу до 5 нс.

Таблиця 9.1

Основні характеристики ПЛІС сімейства MAX7000S

Параметр	ЕРМ703 2S	ЕРМ706 4S	ЕРМ712 8S	ЕРМ716 0S	ЕРМ719 2S	ЕРМ725 6S
Логічна ємність, кількість еквівалентних вентилів	600	1 250	2 500	3 200	3 750	5 000
Кількість макрочарунок	32	64	128	160	192	256
Кількість логічних блоків	2	4	8	10	12	16
Кількість програмувальних користувачем виводів	36	68	100	104	124	164
Затримка поширення сигналу вхід/вихід, [нс]	5	5	6	6	7,5	7,5
Час установки глобального тактового сигналу, [нс]	2,9	2,9	3,4	3,4	4,1	3,9
Затримка глобального тактового сигналу до виходу, [нс]	2,5	2,5	2,5	2,5	3,0	3,0
Максимальна глобальна тактова частота, [МГц]	175,4	175,4	147,1	149,3	125,0	128,2

ПЛІС сімейства **MAX7000S** мають можливість апаратної емуляції виходів з відкритим колектором і задовольняють вимогам стандарту **PCI**.

Є можливість індивідуального програмування кіл скидання, установки і тактування тригерів, які входять у макрочарунку. Передбачено режим зниженого енергоспоживання [8].

Програмувальний логічний розширник дозволяє реалізувати на одній макрочарунці функції до **32** змінних. Є також можливість завдання біта таємності для захисту від несанкціонованого тиражування розробки.

На відміну від архітектури ПЛІС сімейства **MAX7000**, ПЛІС сімейства **MAX7000S** (рис. 9.1) мають можливість використання двох глобальних тактових сигналів.

Основними елементами структури ПЛІС сімейства **MAX7000** та **MAX7000S** є:

- логічні блоки (**ЛБ, Logic array blocks, LAB**);
- макрочарунки (**МЧ, macrocells**);
- логічні розширники (**expanders**) (паралельний (**parallel**) та роздільний (**shareble**));
- програмувальна матриця з'єднань (**ПМЗ, Programmable Interconnect Array, PIA**);
- елементи введення/виведення (**ЕВВ, I/O control block**).

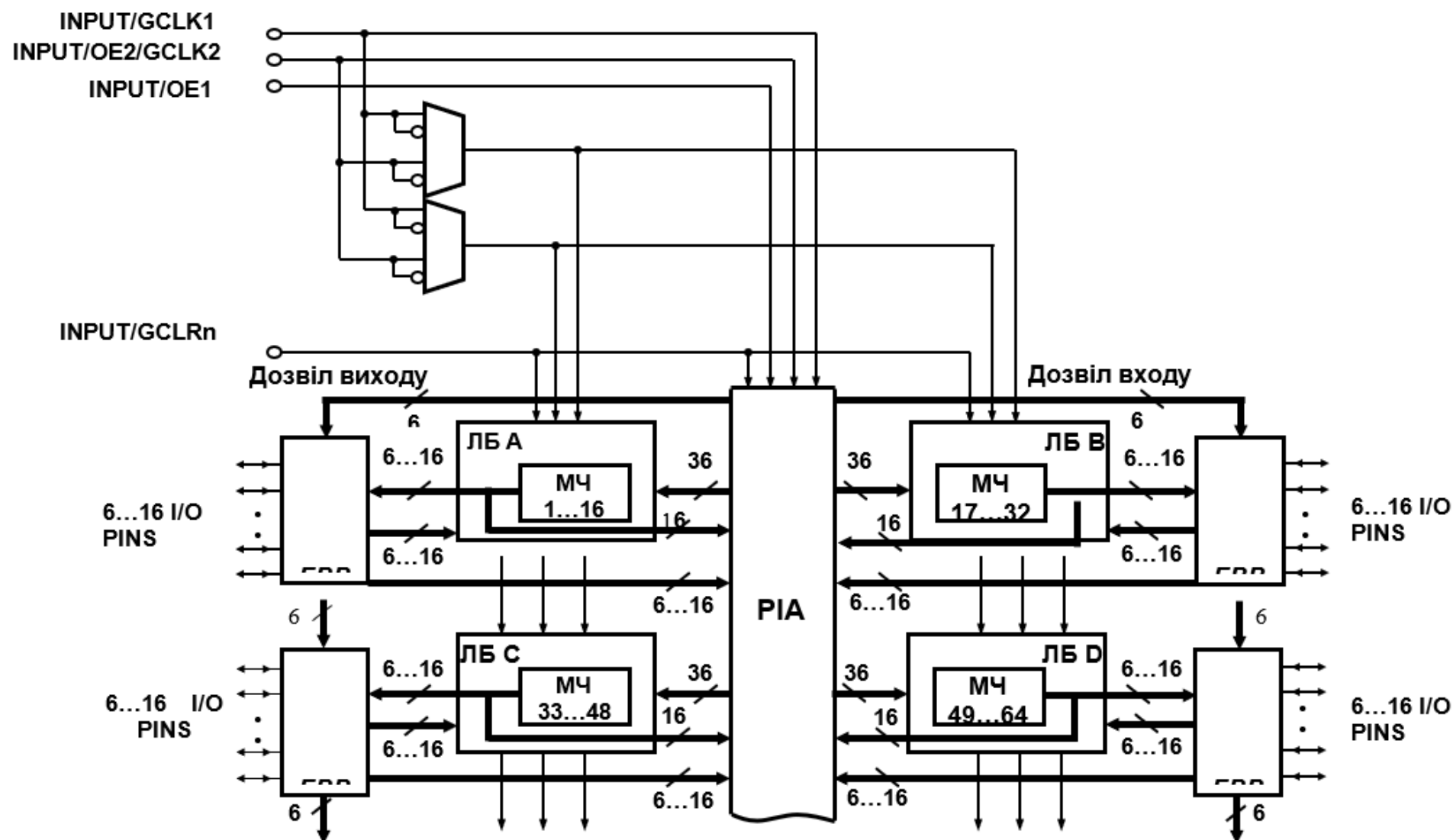


Рис. 9.1. Архітектура ПЛІС сімейства MAX7000S

Як видно з рис. 7.1, в основі архітектури ПЛІС сімейства **MAX7000** та **MAX7000S** лежать логічні блоки, які складаються з **16** макрочарунок кожний. Логічні блоки з'єднуються за допомогою програмувальної матриці з'єднань. Кожний логічний блок має **36** входів з ПМЗ.

Комбінаційні функції реалізуються на локальній програмувальній матриці та матриці розподілу термів, яка дозволяє об'єднувати логічні множення за допомогою логічної операції **ЧИ (OR)**, або **ЧИ, ЩО ВИКЛЮЧАЄ (XOR)**. Окрім того, матриця розподілу термів дозволяє комутувати кола керування тригером **МЧ**.

9.3. ПЛІС типу FPGA

ПЛІС сімейств **FLEX10K**, **FLEX10KA**, **FLEX10KE** на даний момент є, мабуть, найбільш популярною елементною базою для реалізації алгоритмів ЦОС, побудови складних пристроїв обробки даних та інтерфейсів.

Це можна пояснити тим, що завдяки великій логічній ємності, зручній архітектурі (що включає вбудовані блоки пам'яті (**EAB – Embedded Array Block**), досить високій надійності і вдалому співвідношенню ціна/логічна ємність, дані ПЛІС задовольняють різноманітним вимогам, що виникають у розроблювача як систем ЦОС, так і пристроїв керування, обробки даних тощо.

У табл. 9.2 наведені основні відомості про ПЛІС сімейства **FLEX10K**.

Таблиця 9.2

ПЛІС сімейства FLEX10K

Параметр	EP1 0K10	EP1 0K20	EP1 0K30	EP1 0K40	EP1 0K50	EP1 0K70	EP1 0K100	EP1 0K130	EP1 0K250
Логічна ємність, кількість екв. вентилів	10000	20000	30000	40000	50000	70000	100000	130000	250000
Кількість ЛЕ	576	1 152	1 728	2 304	2 880	3 744	4 992	6 656	12 160
Вбудовані блоки пам'яті	3	6	6	8	10	9	12	16	20
Обсяг пам'яті [біт]	6 144	12 288	12 288	16 384	20 480	18 432	24 576	32 768	40 960
Максимальна кількість виводів користувача	150	189	246	189	310	358	406	470	470

У даний час випускаються ПЛІС сімейства **FLEX10KE**, які мають ємність вбудованого блока пам'яті **4096** бітів, на відміну від ПЛІС інших сімейств, що мають ємність **EAB 2048** бітів.

Узагальнена функціональна схема ПЛІС сімейства **FLEX10K** наведена на рис. 9.2. Архітектура ПЛІС за основу має логічні блоки, які містять **8 ЛЕ** і локальну матрицю з'єднань. Глобальна матриця з'єднань (**ГМЗ**) розділена на рядки і стовпці, має безперервну структуру (**Fast Track Interconnect**). Посередині рядка розташовуються вбудовані блоки пам'яті. Крім того, є глобальні кола керування, синхронізації і керування введенням/виведенням.

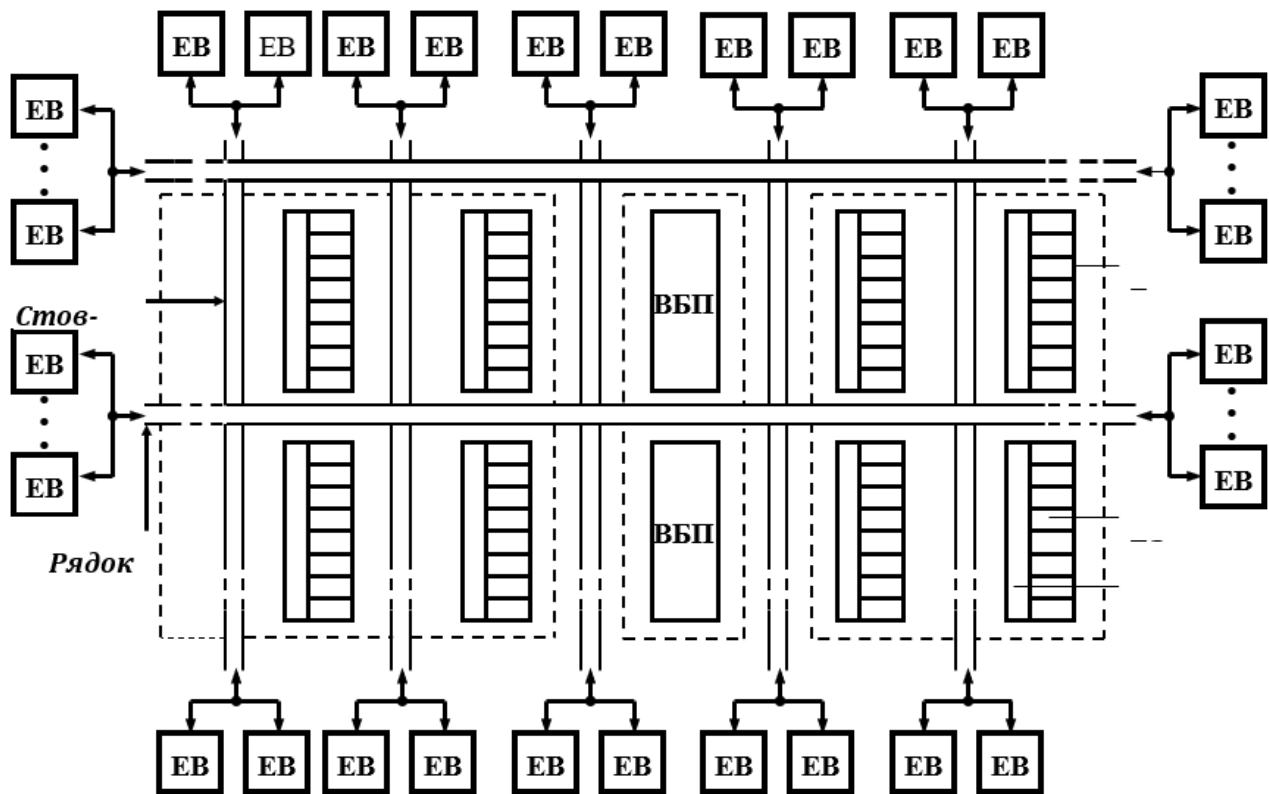


Рис. 9.2. Архітектура ПЛІС сімейства FLEX10K

Вбудований блок пам'яті (**ВБП**) становить оперативний запам'ятовуючий пристрій ємністю **2048 (4096)** бітів і складається з локальної матриці з'єднань, власне модуля пам'яті, синхронних буферних регістрів, а також програмувальних мультиплексорів (рис.9.3).

Сигнали на вхід **ЛМЗ ВБП** надходять з рядка **ГМЗ**. Тактові і керуючі сигнали надходять із глобальної шини керуючих сигналів. Вихід **ВБП** може бути скомутований як на рядок, так і на стовпець **ГМЗ**.

Наявність синхронних буферних регістрів і програмувальних мультиплексорів дозволяє конфігурувати **ВБП** як запам'ятовуючий пристрій з організацією **256×8, 512×4, 1024×2, 2048×1**.

Наявність **ВБП** дає можливість табличної реалізації таких елементів пристроїв **ЦОС**, як перемножники, арифметичний логічний пристрій, суматор тощо, які мають швидкодію до **100 МГц** (звичайно, при най-сприятливіших умо-

вах реальна швидкодія арифметичних пристроїв, реалізованих на базі ВБП, складає 10...50 МГц).

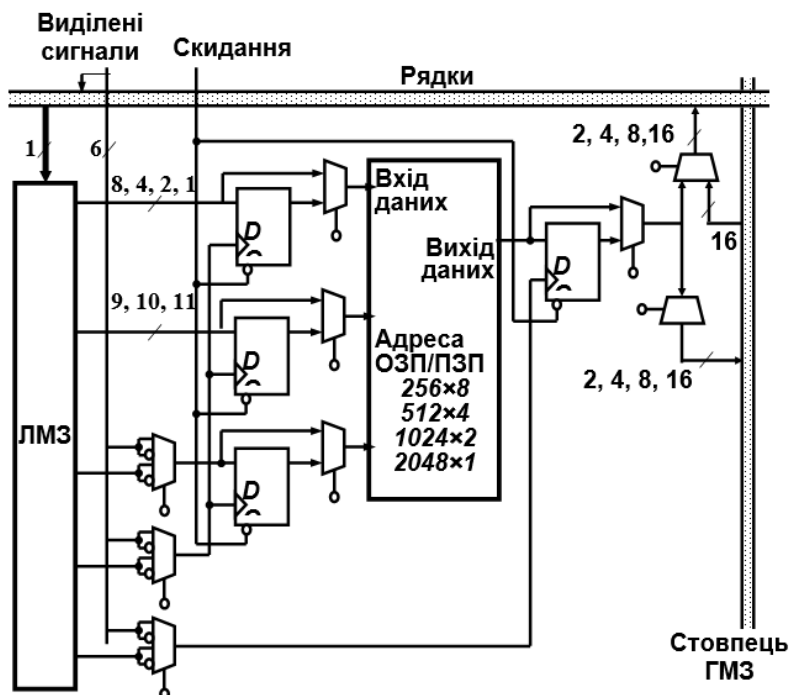


Рис. 9.3. Вбудований блок пам'яті

Усі ПЛІС сімейства **FLEX10K** сумісні за рівнями із шиною **PCI**, мають можливість як послідовного, так і паралельного завантаження, цілком підтримують стандарт **JTAG**.

Структура логічного блока ПЛІС сімейства **FLEX10K** наведена на рис. 9.4. Сигнали на вхід ЛМЗ надходять як з рядка ГМЗ, так і з виходів кожного з 8 ЛЕ, які входять до складу ЛБ. Сигнали з виходу ЛБ надходять як на рядок, так і на стовпець ГМЗ.

Як правило, архітектура ЛЕ всіх сімейств **FLEX** практично однакова. За допомогою схем організації переносів (**carry chain**) і каскадування (**cascade chain**) можливе розширення можливостей ЛЕ.

Елемент введення/виведення ПЛІС сімейства **FLEX10K** з'єднує канал рядка або стовпця ГМЗ із виводом мікросхеми. ЕВВ здійснює введення/виведення біта даних з різними швидкостями, тимчасове збереження даних, емуляцію відкритого колектора.

Наявність вхідного (**input register**) і вихідного (**output register**) регістрів дозволяє зберігати дані, що знижує логічне навантаження на ЛЕ і вивільняє ресурси ПЛІС для реалізації інших функцій. Швидкість перемикування буфера

ЕВВ може бути задана при конфігурації **ПЛІС**. Знижена швидкість переключення дозволяє знизити рівень імпульсних завад і “дзвін” у системі. Варто пам’ятати, що режим емуляції відкритого колектора забезпечує не занадто потужний вихідний сигнал, тому при необхідності узгодження з зовнішніми схемами краще використовувати спеціалізовані буфери. Принаймні при впливі високої напруги простіше (і дешевше) поміняти буфер, а не всю **ПЛІС**.

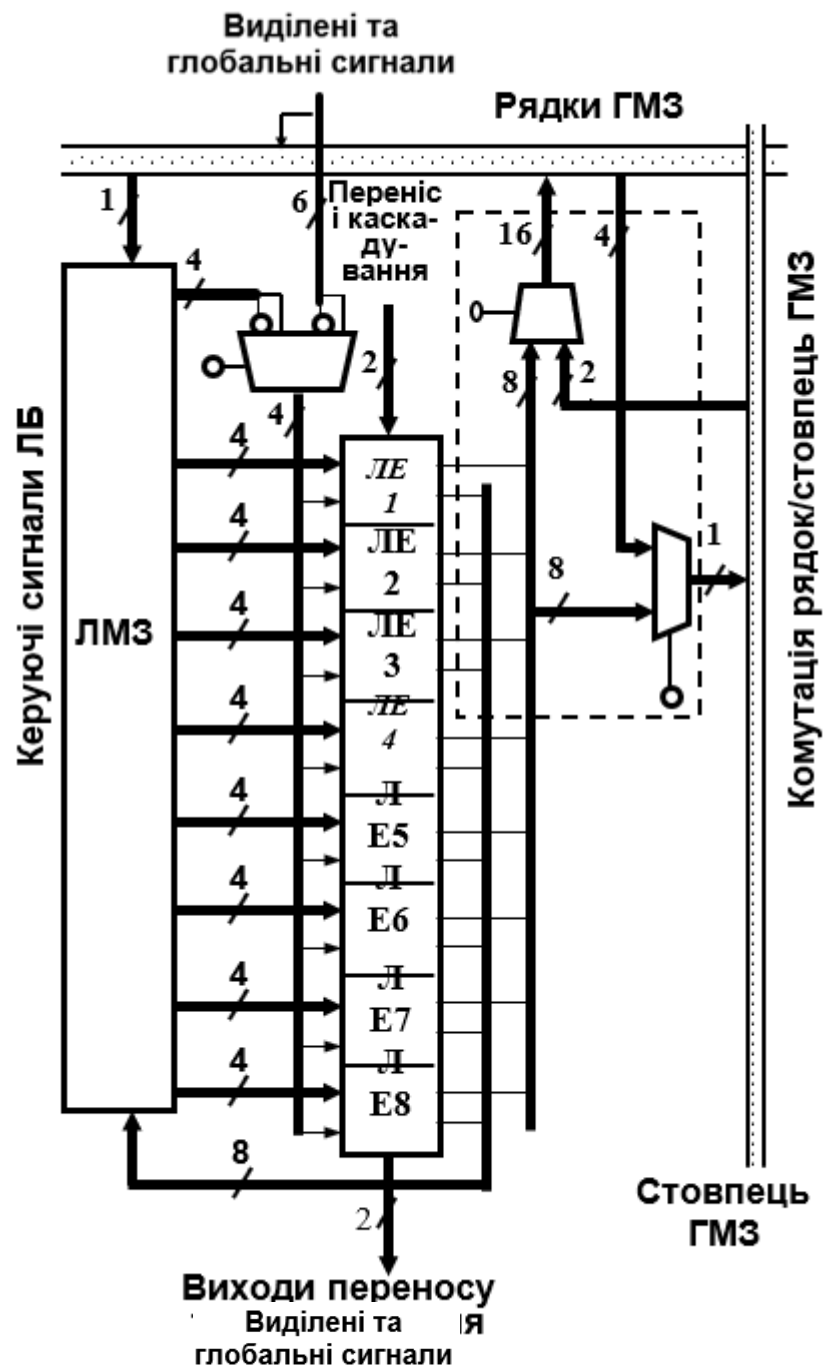


Рис. 9.4. Логічний блок ПЛІС сімейства FLEX10К

9.4. ПЛІС типу SOC

Розвиток і різноманітність архітектур функціональних перетворювачів, які є основою базових вузлів ПЛІС, привели до того, що в останні роки ПЛІС стають основою для “систем на кристалі”. В основі ідеї SOC лежить інтеграція всієї електронної системи в одному кристалі (наприклад, у випадку персонального комп’ютера такий чіп поєднує процесор, пам’ять та ін.). Компоненти цих систем розробляються окремо і зберігаються у вигляді файлів модулів, які параметризуються. Остаточна структура SOC-мікросхеми виконується на базі цих “віртуальних компонентів”, які називаються також “блоками інтелектуальної власності”, за допомогою програм автоматизації проектування електронних пристроїв. Завдяки стандартизації в одне ціле можна поєднувати “віртуальні компоненти” від різних розроблювачів [1].

Ідея побудови “систем на кристалі” спонукала ведучих виробників ПЛІС до випуску виробів з еквівалентною ємністю **1 000 000** еквівалентних вентилів і більше.

Прикладом нових сімейств ПЛІС, придатних для реалізації SOC, є сімейство **APEX20K**, основні характеристики якого наведені в табл. 9.3.

Таблиця 9.3

Основні характеристики ПЛІС сімейства APEX20K

Параметр	EP20K100	EP20K160	EP20K200	EP20K300	EP20K400	EP20K600	EP20K1000
Логічна ємність, кількість екв. вентилів, $\times 10^3$	263	404	526	728	1 052	1 537	2 670
Кількість ЛЕ	4 160	6 400	8 320	11 520	16 640	24 320	42 240
Вбудовані блоки пам’яті	26	40	52	72	104	152	264
Макс. обсяг пам’яті [біти]	53 248	81 920	106 496	147 456	212 992	311 296	540 672
Кількість макрочарунок	416	640	832	1 152	1 664	2 432	4 224
Кількість виводів користувача	252	320	382	420	502	620	780

Архітектура ПЛІС сімейства **APEX20K** (рис. 9.5) поєднує у собі переваги **FPGA** ПЛІС з їх таблицями перекодувань, що входять до складу логічного елемента, логіку обчислення ДДНФ, характерну для ПЛІС **CPLD**, а також вбудовані модулі пам’яті.

Характерною рисою ПЛІС сімейства **APEX20K** є об’єднання **ЛБ** у так званий мега-блок (**megaLAB**) який має власну безперервну матрицю з’єднань (**megaLAB interconnect**).

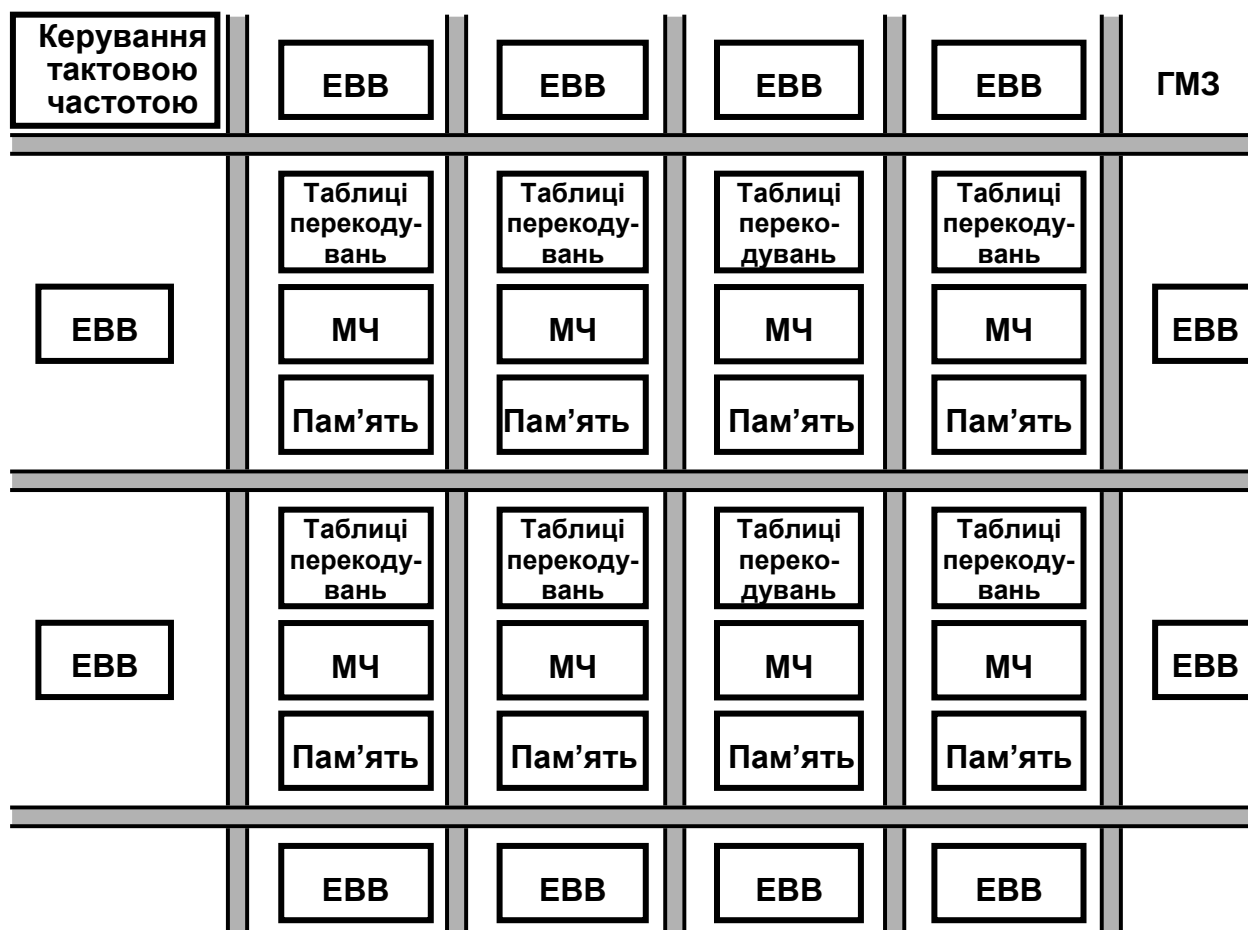


Рис. 9.5. Архітектура ПЛІС сімейства АРЕХ20К

Така організація з'єднань дозволяє виділити додаткові ресурси для трасування, крім того, у кожному мегаблоці може бути цілком розведена та або інша функціонально закінчена частина системи, що дозволяє при її модифікації не перетрасувати цю ділянку й тим самим зберегти всі задані часові параметри. Також така організація ПЛІС дозволяє більш досконало організувати відповідне програмне забезпечення і створити засоби колективної роботи над проектом.

На рис. 9.6 зображена структура ЛБ ПЛІС сімейства АРЕХ20К. Кожен ЛБ складається з 10 ЛЕ, які мають складну структуру.

Ця структура ЛБ поєднує все краще, що напрацьовано в попередніх сімействах ПЛІС. Кожен ЛЕ має можливість комутації на два стовпці ГМЗ, подібно ПЛІС сімейства FLEX6000. Матриця з'єднань мегаблока (МЗМ) комутується на ЛМЗ ЛБ і на рядки ГМЗ.

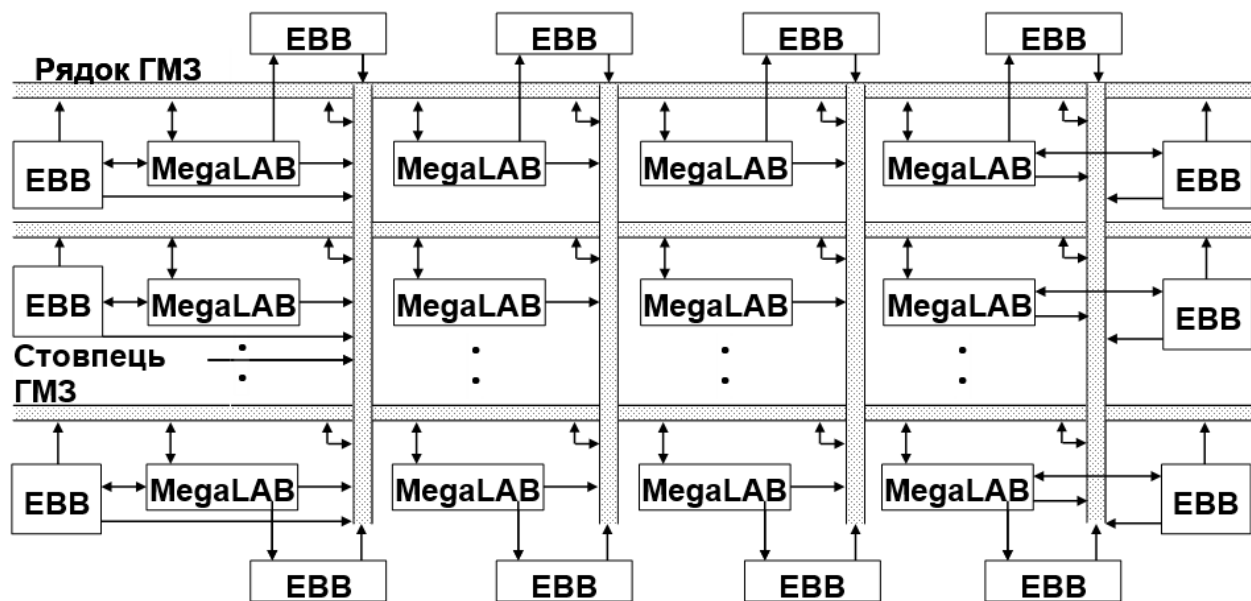


Рис. 9.6. Структура з'єднань ПЛІС сімейства APEX20K

9.5. Проектування цифрових пристроїв на ПЛІС

Всі сучасні методики проектування ЦП на базі складних ВІС/ЗВІС, що програмуються, засновані на використанні САПР. Правильний вибір САПР – важлива умова ефективного проектування та прискорення випуску продукції. Для проектування різного роду цифрових пристроїв існують або системи автоматичного проектування (САПР), або так звані мови опису апаратури, які дозволяють складати схеми за допомогою мови програмування. САПР і мови опису апаратури сумісні між собою, тобто схема, яка складена в тому або іншому середовищі, “розуміється” і САПР, і середовищем мови.

9.5.1. Засоби описування проекту

Використання САПР вимагає ефективних, наочних, керованих та контролюємих засобів описування проекту. Описати пристрій, що проектується, можна різними засобами, причому, як правило, використовують засоби, що здатні описати весь проект. Методи описування, які застосовуються виключно для окремих фрагментів пристрою, відносяться до числа рідких.

У наш час до найбільш розповсюджених універсальних способів описування, що використовується для будь-якого рівня ієрархії проекту, відносять *графічний* та *текстовий*. Кожний із способів описування проекту має свої переваги та недоліки.

Графічне представлення проекту створюється у базі допустимих для вибраної САПР бібліотечних елементів, наприклад, в базисі елементів стандартної

серії ТТЛ (Ш). Головна перевага графічного способу – традиційність та наочність, що пов'язана із звичністю розробників до сприйняття зображення схем. Зрозуміло, що ця перевага проявляється тільки при правильному ієрархічному та структурному розділенні проекту.

Сучасні мови описування апаратури (**HDL, Hardware Description Languages**) допускають описування пристрою, що проектується, як з точки зору його поведінки, так і з точки зору його структури. Ці можливості роблять все більш розповсюдженим представлення проекту у формі текстового описування алгоритмів функціонування його фрагментів та текстового описування меж блочних з'єднань для складних проектів.

Перевагою текстового способу описування проекту є його компактність та відносна простота автоматизації будь-яких перетворень. Дуже важлива можливість використання стандартних універсальних мов типу **HDL**, що забезпечує простоту переносу проекту з однієї апаратної платформи на іншу та перехід від одної **САПР** до іншої.

Мовний опис апаратури отримує все більше розповсюдження. Текстові описування мають два основних різновиди – *мови низького рівня* (аналогі мов програмування типу асемблера) та *високого рівня*.

Мови низького рівня ближчі до апаратних засобів, внаслідок чого надають компіляторам потенціальні можливості створення проектів з більш виграними часовими параметрами. Розрахунком за це є, як правило, жорстка орієнтація на конкретне обладнання (апаратуру) та фірму, що її виробляє. Прикладами таких мов є мова **AHDL (Altera HDL)** та **ABEL** (фірми **Xilinx**).

9.5.2. Мова опису апаратури AHDL

Мова опису апаратури **AHDL** призначена для опису комбінаційних і послідовних логічних пристроїв, групових операцій, цифрових автоматів і таблиць істинності з урахуванням архітектурних особливостей ПЛІС.

Оператори й елементи мови **AHDL** є достатнім потужним і універсальним засобом опису алгоритмів функціонування цифрових пристроїв, зручним у використанні. Мова опису апаратури **AHDL** дає можливість створювати ієрархічні проекти в рамках однієї цієї мови або ж в ієрархічному проекті використовувати як **TDF**-файли, розроблені мовою **AHDL**, так і інші типи файлів. Для створення проектів на **AHDL** можна користуватися будь-яким текстовим редактором, але текстовий редактор системи **MAX+PLUSII** надає ряд додаткових можливостей для введення, компіляції та налагодження проектів.

Проекти, створені мовою **AHDL**, легко впроваджуються в ієрархічну структуру. Система **MAX+PLUSII** дозволяє автоматично створити символ компонента, алгоритм функціонування якого описується **TDF**-файлом, і потім вставити його у файл схемного опису (**GDF**-файл). Подібним же чином можна вводити власні функції розроблювача у будь-який **TDF**-файл.

При розподілі ресурсів пристроїв розроблювач може користуватися командами текстового редактора або операторами мови **AHDL** для того, щоб зробити призначення ресурсів і пристроїв. Крім того, розроблювач може тільки перевірити синтаксис або виконати повну компіляцію для налагодження та запуску проекту. Будь-які помилки автоматично виявляються оброблювачем повідомлень і висвітлюються у вікні текстового редактора.

9.5.3. Мова опису апаратури VHDL

Мови високого рівня – менш пов'язані з апаратними платформами і тому більш універсальні. Серед них найбільш розповсюджені мови **HDL** та **Verilog**. Ці мови, як й інші алгоритмічні мови високого рівня, в принципі дозволяють описувати будь-який алгоритм у послідовній формі, тобто через послідовність операторів присвоювання і прийняття рішення.

Мови опису апаратури (Hardware Description Language) є формальним записом, що може бути використаним на всіх етапах розробки цифрових електронних систем. Це можливо тому, що мова легко сприймається як машиною, так і людиною, вона може використовуватися на етапах проектування, верифікації, синтезу і тестування апаратури, також для передачі даних про проект, модифікацію і супровід. Найбільш універсальною і розповсюдженою мовою опису апаратури є **VHDL**. На цій мові можливий як поведінковий, так і структурний і потоковий опис цифрових схем.

Мова **VHDL** використовується в багатьох системах для моделювання цифрових схем, проектування **ПЛІС**, базових матричних кристалів, замовлених інтегральних схем. З погляду програміста, мова **VHDL** складається начебто з двох компонентів (загальноалгоритмічного та проблемно-орієнтованого).

Загальноалгоритмічний компонент **VHDL** – це мова за синтаксисом і семантикою близька до сучасних мов програмування типу “**Паскаль**”, “**С**” та ін. Мова відноситься до класу суворо типізованих. Крім вбудованих (пакет **STANDART**), простих (скалярних) типів даних (цілий, дійсний, булевий, бітовий), даних типу час, даних типу посилання (покажчик), користувач може вводити свої типи даних (дані які перелічуються, діапазонний тип та ін.).

Крім скалярних даних, можна використовувати агрегати: масиви **ARRAY** (у тому числі й бітові вектори **BIT_VECTOR**, і символьні рядки **STRING**), масиви запису **RECORD**, файли **FILE**.

Послідовно виконувані (послідовні) оператори **VHDL** можуть використовуватися в описі процесів, процедур і функцій. До їх складу входять:

- оператор присвоювання змінної (**:=**);
- послідовний оператор призначення сигналу (**<=>**);
- послідовний оператор твердження (**ASSERT**);
- умовний оператор (**IF**);
- оператор вибору (**CASE**);
- оператор циклу (**LOOP**);
- порожній оператор (**NULL**);
- оператор повернення процедури функції (**RETURN**);
- оператор послідовного виклику процедури.

9.5.4. Мова опису апаратури *Verilog HDL*

На відміну від **VHDL**, структура та синтаксис якого нагадують такі “складні” мови, як АДА або АЛГОЛ, синтаксис **Verilog** нагадує мову “C++” (дуже популярну у середовищі як програмістів, так і розроблювачів убудованих систем і систем ЦОС). **Verilog** дозволяє досить ефективно виконати опис і провести моделювання (**simulate**) та синтез цифрових схем завдяки застосуванню вбудованих примітивів (**built-in primitives**), примітивів користувача (**user-defined primitives**), засобів часового контролю (**timing checks**), моделювання затримки поширення від входу до виходу (**pin-to-pin delay simulation**), можливістю завдання зовнішніх тестових сигналів (**external stimulus**).

У мові **Verilog** підтримується набір типів логічних вентилів (**Gate types**). Для логічних вентилів визначені ключові слова (**keywords**): **AND** (І), **NAND** (І-НІ), **OR** (ЧИ), **NOR** (ЧИ-НІ), **XOR** (ЧИ, ЩО ВИКЛЮЧАЄ), **XNOR** (ЧИ-НІ, ЩО ВИКЛЮЧАЄ), **BUF** (буферний елемент), **NOT** (заперечення, НІ).

У **Verilog** при використанні вентилів необхідно задати входи і виходи елемента, а також (не обов’язково) ім’я вентиля. Наприклад, вентиля **AND** і **OR** повинні мати один вихід і два (або більше) входи. Так, для вентиля **AND** маємо:

and <name> <list of arguments>
and myand (out, in1, in2, in3).

У **Verilog** підтримується два типи коментарів. Якщо необхідно прокоментувати один рядок, то використовуються дві косі риски на її початку:

// це коментарі

Щоб прокоментувати кілька рядків, використовується така конструкція:

```
/* Це  
коментарі...*/
```

У **Verilog** існує три типи операторів (з одним, двома і трьома операндами). Унарні оператори розташовуються ліворуч від операнда, бінарні – між операндами, а тернарний оператор розділяє три операнди двома операторами.

Приклади операторів:

```
clock = ~ clock; // унарний оператор заперечення  
// clock – операнд  
c = a || b; // || – бінарний оператор ЧИ; a, b – операнди  
r = s ? t : u; // ?: – тернарний оператор, читається як  
r = [if s true then t else u]
```

9.5.5. Система проектування **MAX+PLUSII**

Назва **САПР MAX+PLUSII** є аббревіатурою від **Multiple Array Matrix Programmable Logic User System** (Користувальницька Система Програмування Логіки Упорядкованих Структур). Система **MAX+PLUSII** забезпечує багатоплатформне архітектурно-незалежне середовище створення дизайну, яке легко пристосовується для конкретних вимог користувача, а також безпосереднє програмування пристроїв.

Система **MAX+PLUSII** пропонує повний спектр можливостей логічного дизайну: різноманітні засоби опису проекту для створення проекту з ієрархічною структурою, потужний логічний синтез, компіляція із заданими часовими параметрами, розподіл на частини, функціональне і часове тестування (симуляція), тестування декількох зв'язаних пристроїв, аналіз часових параметрів системи, автоматична локалізація помилок, а також програмування і верифікація пристроїв. У системі **MAX+PLUSII** можна як читати, так і записувати файли мовою **AHDL**, а також файли трасування у форматі **EDIF**, файли на мовах опису апаратури **Verilog HDL** і схемні файли **OrCAD**.

Показаний на рис. 9.7 склад програмного забезпечення системи **MAX+PLUSII** є повним комплектом, який забезпечує створення логічних дизайнів.

Можливий опис проекту (**Design Entry**) у вигляді файлу мовою опису апаратури, створеного або в зовнішньому редакторі, або в текстовому редакторі **MAX+PLUSII (Text Editor)**, у вигляді електричної принципової схеми за до-

помогою графічного редактора (**Graphic Editor**), у вигляді часової діаграми, створеної в сигнальному редакторі (**Waveform Editor**).

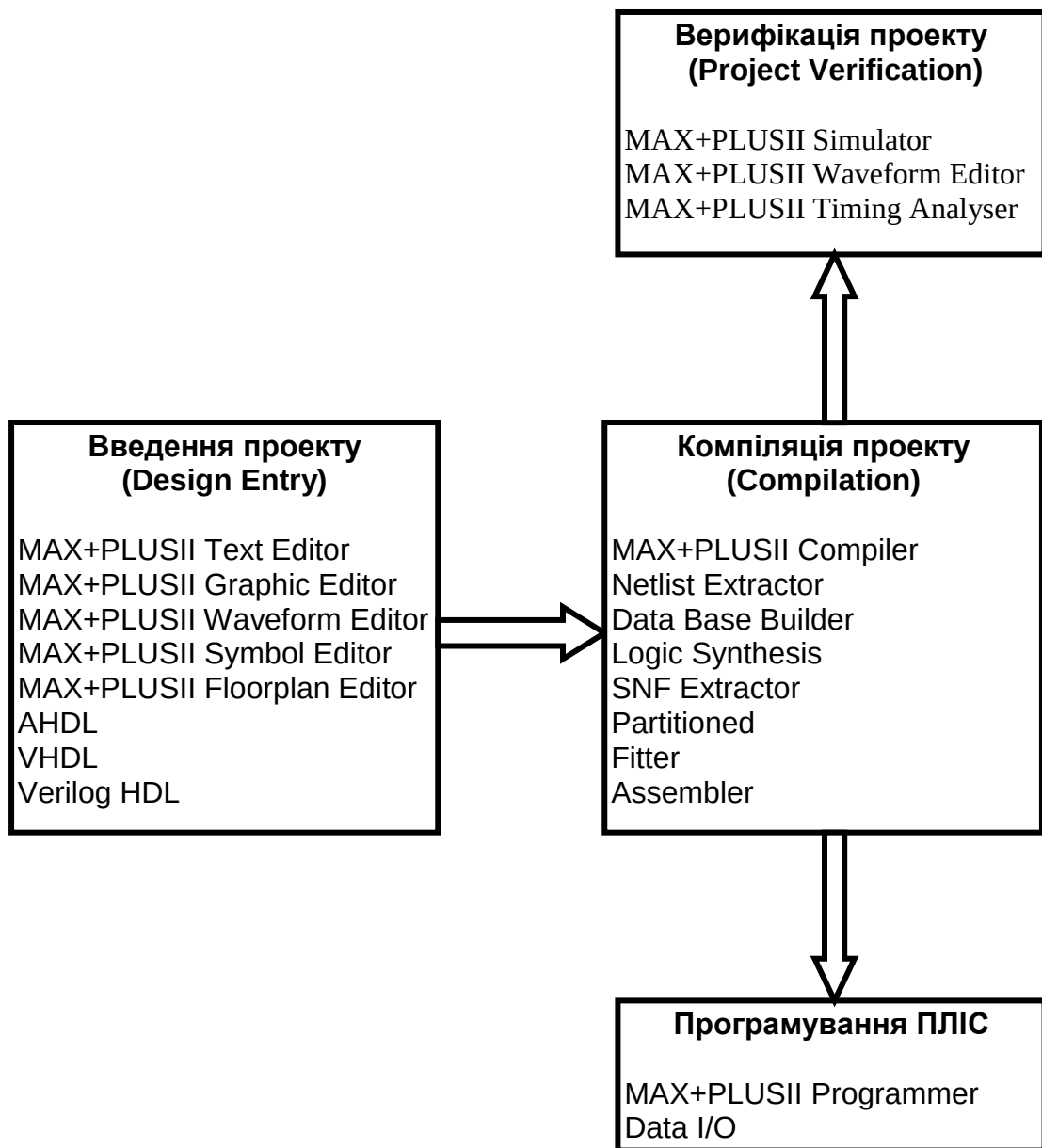


Рис. 9.7. Середовище проектування в системі MAX+PLUSII

Основою системи **MAX+PLUSII** є компілятор, який забезпечує потужні засоби обробки проекту, при цьому можна задавати потрібні режими роботи компілятора.

Для зручності роботи зі складними ієрархічними структурами кожному піддизайну відповідає символ, редагування якого здійснюється за допомогою графічного редактора (**Symbol Editor**). Розміщення вузлів по ЛБ і виводах ПЛІС виконують за допомогою порівневого планувальника (**Floorplan Editor**).

Компіляція проекту, включаючи витяг списку з'єднань (**Netlist Extractor**), побудову бази даних проекту (**Data Base Builder**), логічний синтез (**Logic Synthesis**), витяг часових, функціональних параметрів проекту (**SNF Extractor**), розбивку на частини (**Partitioned**), трасування (**Fitter**), формування файлу програмування або завантаження (**Assembler**) виконуються за допомогою компілятора системи (**Compiler**).

Верифікація проекту (**Project Verification**) виконується за допомогою симулятора (**Simulator**), результати роботи якого зручно переглянути в сигнальному редакторі **Waveform Editor**, у ньому ж створюються текстові впливи.

Безпосереднє програмування або завантаження конфігурації пристроїв з використанням відповідного апаратного забезпечення виконуються з використанням програматора (**Programmer**).

Автоматична локалізація помилки, видача повідомлення та чимала документація про помилки прискорюють і полегшують проведення змін у дизайні. Можна створювати вихідні файли в різних форматах з різною метою, таких, як робота функцій, зв'язок декількох пристроїв, аналіз часових параметрів, програмування пристрою [1].

Процедуру розробки нового проекту від концепції до завершення можна спрощено подати у такий спосіб:

- створення нового файлу (**design file**) проекту або ієрархічної структури декількох файлів проекту з використанням різних редакторів розробки проекту в системі **MAX+PLUSII**, тобто графічного, текстового і сигнального редакторів;
- завдання імені файлу проекту верхнього рівня (**Top of Hierarchy**) як ім'я проекту (**project name**);
- призначення сімейства ПЛІС для реалізації проекту. Користувач може сам призначити конкретний пристрій або надати це компілятору для того, щоб оцінити необхідні ресурси;
- відкриття вікна компілятора та його запуск для початку компіляції проекту;
- у випадку успішної компіляції можливі тестування та часовий аналіз;
- програмування або завантаження конфігурації шляхом запуску модуля програматора з наступною вставкою пристрою в програмуючий адаптер програматора **MPU (Master Programming Unit)** або підключення пристроїв **MasterBlaster, BitBlaster, ByteBlaster** чи кабелю завантаження (**FLEX download cable**) до пристрою, який програмується в системі.

КОНТРОЛЬНІ ПИТАННЯ

1. Дати характеристику **ПЛІС**. У чому полягають особливості використання **ПЛІС**?
2. Навести класифікацію **ПЛІС** за структурною ознакою.
3. Пояснити принцип роботи **ПЛІС** типу **CPLD**. Які основні елементи складають структуру **ПЛІС** сімейства **MAX7000**?
4. Дати характеристику **ПЛІС** типу **FPGA**. Пояснити принцип побудови архітектури **ПЛІС** сімейства **FLEX10K**.
5. Дати характеристику **ПЛІС** типу **SOC**. Пояснити принцип побудови архітектури **ПЛІС** сімейства **APEX20K**.
6. Дати характеристику графічному та текстовому засобам опису проекту.
7. Мова опису апаратури **AHDL**. Сфера застосування цієї мови, переваги та недоліки.
8. Мова опису апаратури **VHDL**. Сфера застосування цієї мови, переваги та недоліки. Пояснити значення наступних операторів: **(:=)**; **(<=)**; **(ASSERT)**; **(IF)**; **(CASE)**; **(LOOP)**; **(NULL)**; **(RETURN)**.
9. Мова опису апаратури **Verilog HDL**. Сфера застосування цієї мови, переваги та недоліки. Дати характеристику елемента (функціональне призначення, кількість входів та виходів), який описаний мовою **Verilog HDL** наступним чином:

and <name> <list of arguments>
and myand (out, in1, in2, in3).

Для наведеного фрагменту програми здійснити доповнення відсутніх символів:

```
clock = ~ clock; _/ унарний оператор заперечення
// clock – операнд
c = a || b; // | _– бінарний оператор ЧИ; a, b – операнди
r=s ? t : u; // ?: – тернарний оператор, читається як
r = [if s true then t else u]
```

10. ЦИФРОВІ КОМП'ЮТЕРИ

10.1. Арифметико-логічний пристрій (АЛП)

Класична ЕОМ складається з трьох основних пристроїв: *арифметико-логічного пристрою*, пристрою управління і запам'ятовуючого пристрою. Розглянемо особливості організації цих пристроїв. Насамперед, розглянемо структуру *арифметико-логічного пристрою*.

У сучасних ЕОМ *арифметико-логічний пристрій* не є самостійним схемотехнічним блоком. Він входить до складу мікропроцесора, на якому будується комп'ютер. Однак знання структури і принципів роботи *АЛП* дуже важливі для розуміння роботи комп'ютера в цілому. Для кращого розуміння цих питань проведемо синтез арифметичного пристрою, призначеного для виконання тільки однієї операції – *множення* чисел з фіксованою комою, заданих в прямому коді, зі старших розрядів множника. У ході цього процесу також звернемо увагу на особливості використання розглянутих вище основних схемотехнічних елементів ЕОМ.

Синтез *АЛП* проходить у кілька етапів. Спочатку необхідно обрати метод, за яким передбачається виконання операції, і скласти алгоритм відповідних дій. Виходячи з алгоритму і формату вихідних даних, слід визначити набір складових елементів *АЛП*.

Потім потрібно визначити зв'язки між елементами, встановити порядок функціонування пристрою і тимчасову діаграму керуючих сигналів, які повинні бути подані на *АЛП* від пристрою керування.

Нехай операнди мають вигляд:

$$\begin{aligned} [X]_{\text{ПК}} &= x_0 x_1 x_2 \dots x_n \\ [Y]_{\text{ПК}} &= y_0 y_1 y_2 \dots y_n, \end{aligned}$$

де x_0, y_0 - знакові розряди.

Операція *множення* чисел з фіксованою комою, заданих в прямому коді, зі старших розрядів множника виконується за наступною формулою:

$$\text{Sign}Z = \text{Sign}X \oplus \text{Sign}Y$$

$$|Z| = y_1 \cdot |X| \cdot 2^{-1} + y_2 \cdot |X| \cdot 2^{-2} + \dots + y_n \cdot |X| \cdot 2^{-n}$$

$$[X]_{i\bar{E}} = 0.1101; \text{Sign}X = 0$$

$$[Y]_{i\bar{E}} = 1.1011; \text{Sign}Y = 1$$

$$\text{Sign}Z = 0 \oplus 1 = 1$$

$$|X| = 0.1101$$

$$|Y| = 0.1011$$

	$y_1 y_2 y_3 y_4$ +0.00000000	$ Z =0$
$y_1 = 1$	0.01101000 +0.01101000	$1 \cdot X \cdot 2^{-1}$ $ Z = Z + X \cdot 2^{-1}$
$y_2 = 0$	0.00000000 +0.01101000	$0 \cdot X \cdot 2^{-2}$ $ Z = Z +0$
$y_3 = 1$	0.00011010 +0.10000010	$1 \cdot X \cdot 2^{-3}$ $ Z = Z + X \cdot 2^{-3}$
$y_4 = 1$	0.00001101 -0.10001111	$1 \cdot X \cdot 2^{-4}$ $ Z = Z + X \cdot 2^{-4}$

Алгоритм обчислень наведено на рис. 10.1.

Кожній змінній, що представлена в алгоритмі, у схемі повинен відповідати елемент зберігання. Розрядність модуля добутку дорівнює сумі розрядності співмножників.

Множення двійкового числа на 2^{-i} забезпечується зсувом цього числа праворуч на відповідну кількість розрядів. Перехід до аналізу чергового розряду множника ($i=i+1$) може бути забезпечений зсувом регістра множника на один розряд в бік старших розрядів.

Виходячи з цього, визначимо склад устаткування, необхідного для реалізації АЛП заданого типу для $n=4$ (табл. 10.1).

Структурну схему пристрою наведено на рис. 10.2.

Часову діаграму сигналів, що надходять на арифметико-логічний пристрій, наведено на рис.10.3.

Робота схеми

Такт 1. Завантаження модулів операндів у регістри RGX, RGY, а їх знаків - в тригери TX і TY. Скидання в "0" регістра результату RGZ.

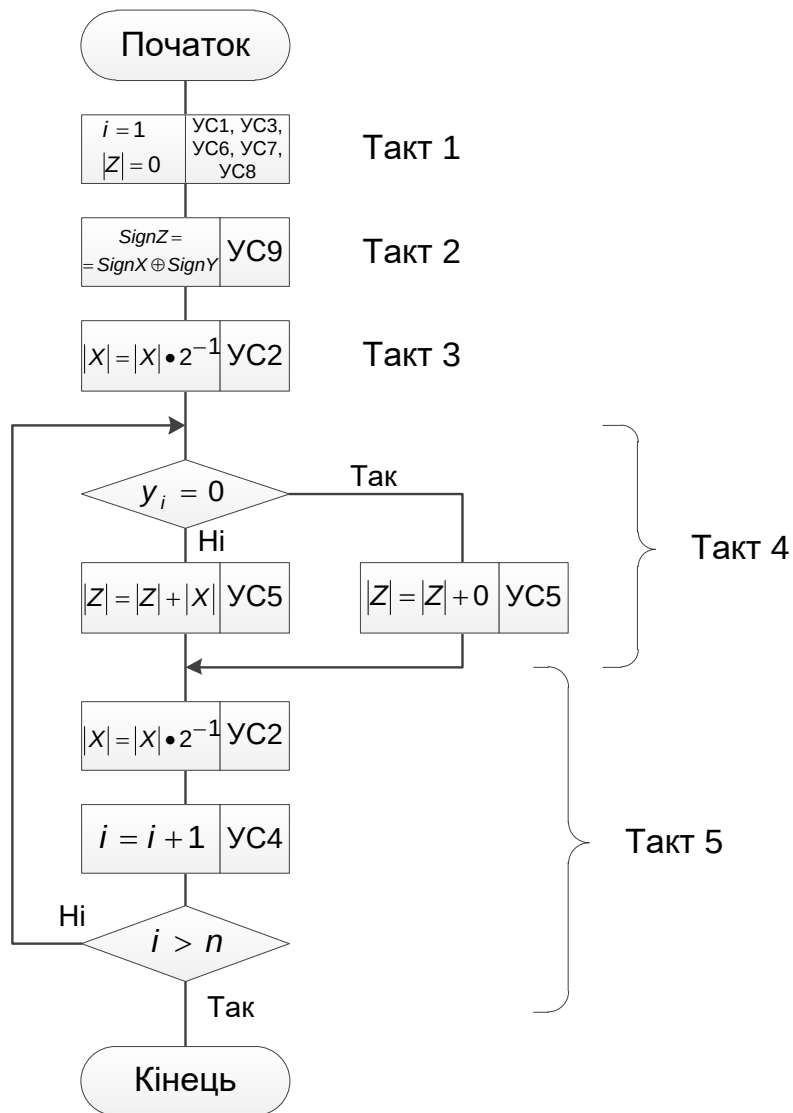


Рис. 10.1. Алгоритм операції множення чисел з фіксованою комою, заданих в прямому коді, зі старших розрядів множника

Такт 2. Запис знака результату в тригер TZ.

Такт 3. Зсув регістра RGX на один розряд праворуч. Через час, рівний затримці на перемикання регістрів і комбінаційних схем, на виході комбінаційного суматора і, отже, на вході регістру RGZ встановлюється результат $0 + y_1 \cdot |X| \cdot 2^{-1}$.

Такт 4. Завантаження RGZ: $|Z| = |Z| + y_1 \cdot |X| \cdot 2^{-1}$.

Такт 5. Зсув RGX на 1 розряд праворуч: $|X| = |X| \bullet 2^{-1}$.

Зсув RGY на 1 розряд ліворуч: $i = i + 1$.

Пристрій управління перевіряє умову закінчення операції: $i > n$.

Такти (6,7), (8,9), (10,11) Повтор дій тактів (4,5) з аналізом інших значень. У такті 10 в регістрі RGZ формується модуль добутку. Такт 11 використовується лише для визначення умови закінчення операції множення.

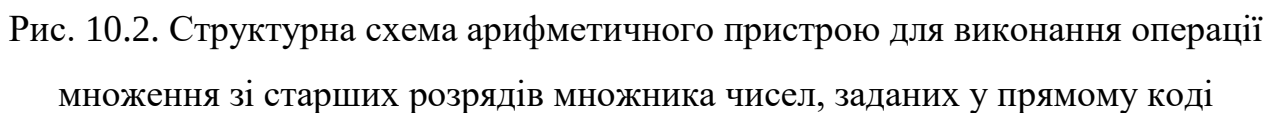
Таблиця. 10.1

Склад устаткування, необхідного для реалізації АЛП заданого типу для $n = 4$

Схема	Розрядність	Функції	Керуючий сигнал
Регістр модуля, що множиться RGX	8	Завантаження. Зсув у бік молодших розрядів.	УС1 УС2
Регістр модуля співмножника RGY	4	Завантаження. Зсув у бік старших розрядів.	УС3 УС4
Регістр модуля результату RGZ	8	Завантаження. Встановлення в «0».	УС5 УС6
Тригер знака, що множиться TX		Завантаження	УС7
Тригер знака TY		Завантаження	УС8
Тригер знака результату TZ		Завантаження	УС9
АЛП	8	Комбінаційний суматор	-
Комбінаційні схеми		Отримання на вході АЛП сигналів «0» або RGX в залежності від значення Y_i	-

10.2 Пристрій керування

1. Комп'ютер умовно можна розділити на два основних блоки: операційний і керуючий. Для реалізації будь-якої команди необхідно на відповідні управляючі входи будь-якого пристрою комп'ютера подати певним чином розподілену в часі послідовність керуючих сигналів. Частина цифрового обчислювального пристрою, призначена для вироблення цієї послідовності, називається *пристроєм керування*.



Будь-яка дія, що виконується в операційному блоці, описується деякою *мікропрограмою* і реалізується за один або кілька тактів. Елементарна функціональна операція, що виконується за один тактовий інтервал і приводиться в дію керуючим сигналом, називається *мікрооперацією*. Наприклад, в спроектованому АЛП для множення чисел в першому такті виконуються наступні *мікрооперації*: $TX = 0$, $TY = 0$, $RGX = |X|$, $RGY = |Y|$, $RGZ = 0$.

Сукупність *мікрооперацій*, які виконуються в одному такті, називається *мікрокоманда* (МК). Якщо всі такти повинні мати одну і ту ж довжину, а саме це і відбувається при роботі комп'ютера, то вона встановлюється відповідно до найтривалішої *мікрооперації*. *Мікрокоманди*, призначені для виконання певної функціонально закінченої послідовності дій, які утворюють *мікропрограму*. Наприклад, мікропрограму утворює набір мікрокоманд для виконання команди множення [2].

Пристрій керування призначений для вироблення керуючих сигналів, під впливом яких відбувається перетворення інформації в арифметико-логічному пристрої, а також операції запису та читання інформації в/з запам'ятовуючого пристрою.

Пристрої керування поділяються на:

- ПК з жорсткою, або схемною логікою;
- ПК з програмованою логікою (*мікропрограмні ПК*).

У *пристроях керування* першого типу для кожної команди, що задається кодом операції, будується набір комбінаційних схем, які в потрібних тактах виробляють необхідні керуючі сигнали.

У *мікропрограмних ПК* кожній команді ставлять у відповідність сукупність збережених в спеціальній пам'яті слів - *мікрокоманд*. Кожна з *мікрокоманд* містить інформацію про *мікрооперації*, що підлягають виконанню в даному такті, і вказівка, яке слово повинно бути вибрано з пам'яті в наступному такті.



Рис. 10.4. Функціональна схема схемного пристрою керування

Схемний пристрій керування

Пристрій керування схемного типу (рис. 10.4) складається з:

– *датчика сигналів*, що виробляє послідовність імпульсів, рівномірно розподілену в часі за своїми шинам (рис. 10.5) (n – загальна кількість керуючих сигналів, необхідних для виконання будь-якої операції; m – кількість тактів, за яку виконується найдовша операція);

– *блоку керування операціями*, що здійснює вироблення керуючих сигналів, тобто комутацію сигналів, що надходять з ДП, у відповідному такті на потрібну керуючу шину;

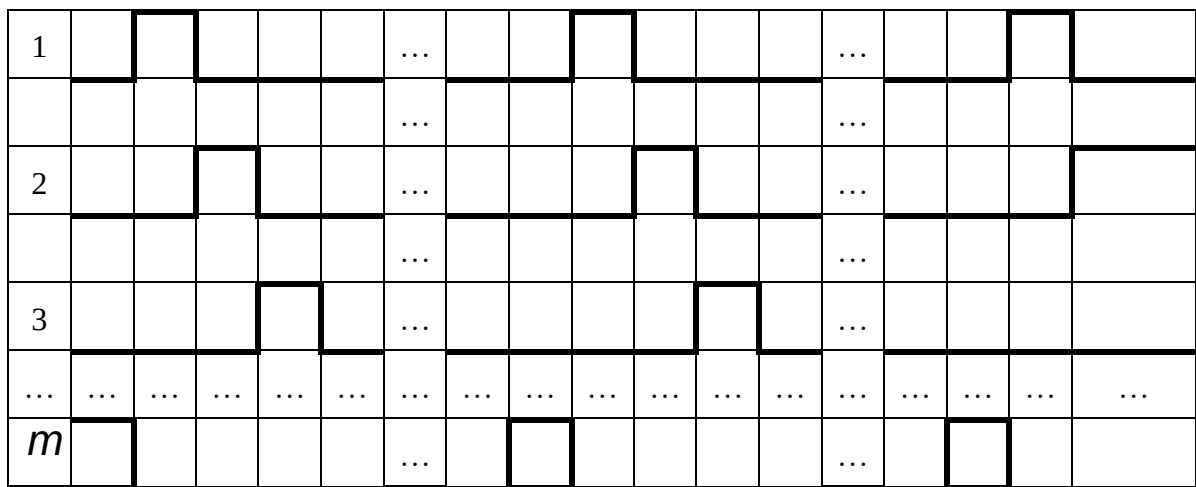


Рис. 10.5. Часова діаграма роботи датчика сигналів

– *дешифратора коду операцій*, який дешифрує код операції команди, яка присутня у даний момент в регістрі команд, і збуджує одну шину, що відповідає даній операції; цей сигнал використовується блоком керування операціями для вироблення потрібної послідовності керуючих сигналів.

– *Датчик сигналів* зазвичай реалізується на основі лічильника з дешифратором або на регістрі зсуву.

Датчик сигналів на основі лічильника з дешифратором

Реалізацію *датчика сигналів* на основі лічильника з дешифратором наведено на рис. 10.6. По задньому фронту кожного тактового імпульсу, що надходить на *пристрій керування* з системного генератора імпульсів, лічильник збільшує свій стан; виходи лічильника з'єднані з входами дешифратора, виходи якого і є виходами *датчика сигналів* (рис. 10.7).

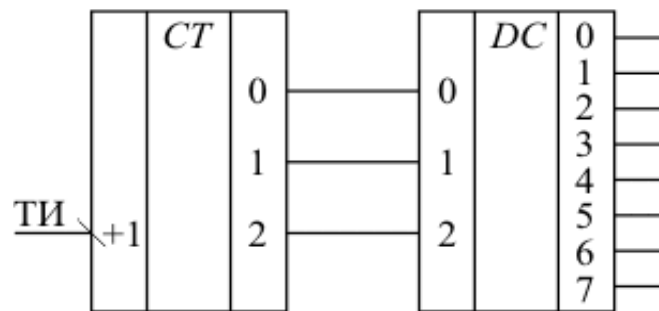


Рис. 10.6. Схема датчика сигналів на основі лічильника з дешифратором

Датчик сигналів на регістрі зсуву

Проектування *датчика сигналів* на регістрі зсуву вимагає лише його "закільцьовування", тобто сполуки виходу останнього розряду з входом, через який в регістр заноситься інформація при зсуві, і первісної установки (рис. 10.7).

У початковому стані регістр містить "1" тільки в розряді 0.

Входи паралельного завантаження регістра для його початкової установки і відповідний цій операції керуючий вхід регістра на схемі не показані.

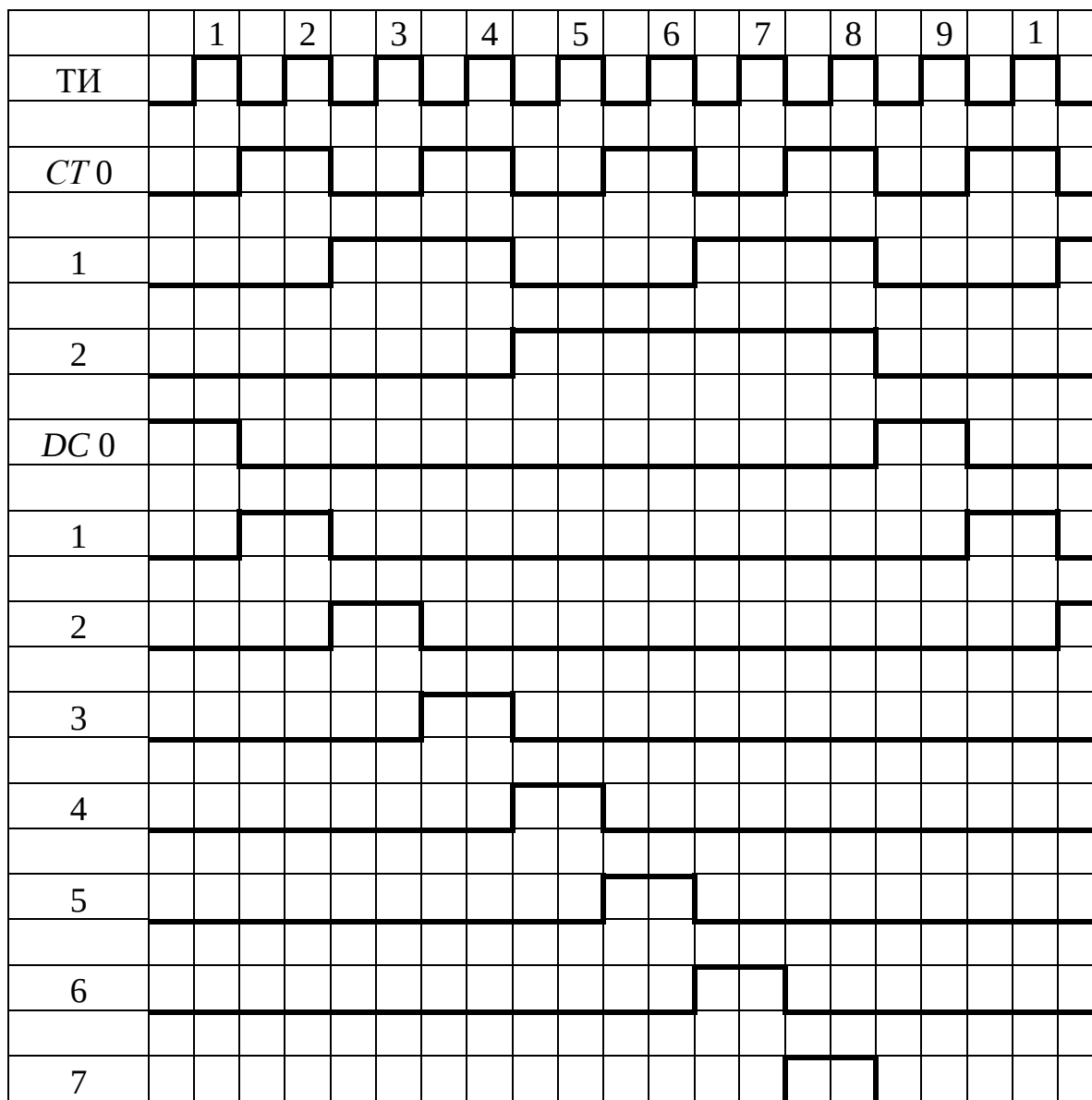


Рис. 10.7. Часова діаграма роботи датчика сигналів на основі лічильника з дешифратором

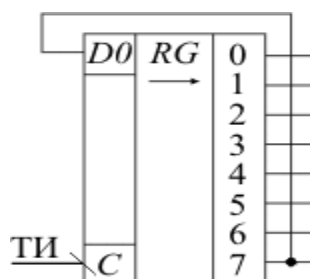


Рис. 10.8. Схема датчика сигналів на основі регістра зсуву

Часову діаграму роботи цієї схеми наведено на рис. 10.9.

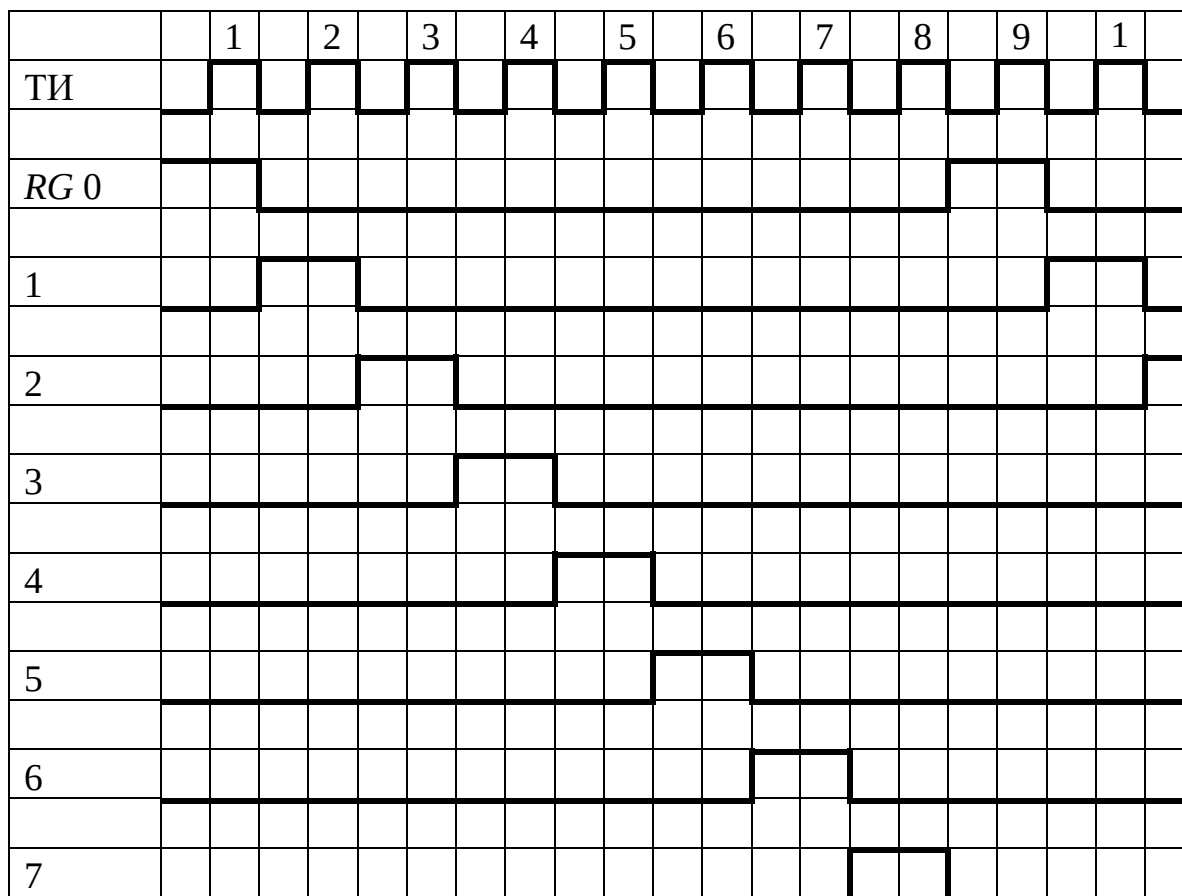


Рис. 10.9. Часова діаграма роботи датчика сигналів на основі регістра зсуву

Найбільш складною частиною схемного пристрою керування є блок керування операціями. Він являє собою нерегулярну схему, структура якої визначається системою команд і складом устаткування процесора. Таке ПК може бути реалізоване у вигляді спеціалізованої інтегральної схеми.

Структурна схема мікропрограмного пристрою керування

Мікропрограмний пристрій керування наведено на рис. 10.8. Перетворювач адреси мікрокоманди перетворює код операції команди, яка присутня у даний момент в регістрі команд, в початкову адресу мікропрограми, що реалізує дану операцію, а також визначає адресу наступної мікрокоманди мікропрограми, що виконується за значенням адресної частини поточної мікрокоманди.

В табл. 10.2. наведено приклад мікропрограми для виконання операції множення чисел в додатковому коді. Передбачається, що початкова адреса мікропрограми дорівнює 300, кількість розрядів множника дорівнює 2, а адресна

частина мікрокоманди містить адресу мікрокоманди, яка повинна бути обрана в наступному такті. В останній мікрокоманді в регістр команд завантажиться чергова команда, код операції якої визначить початкову адресу чергової мікропрограми [10].

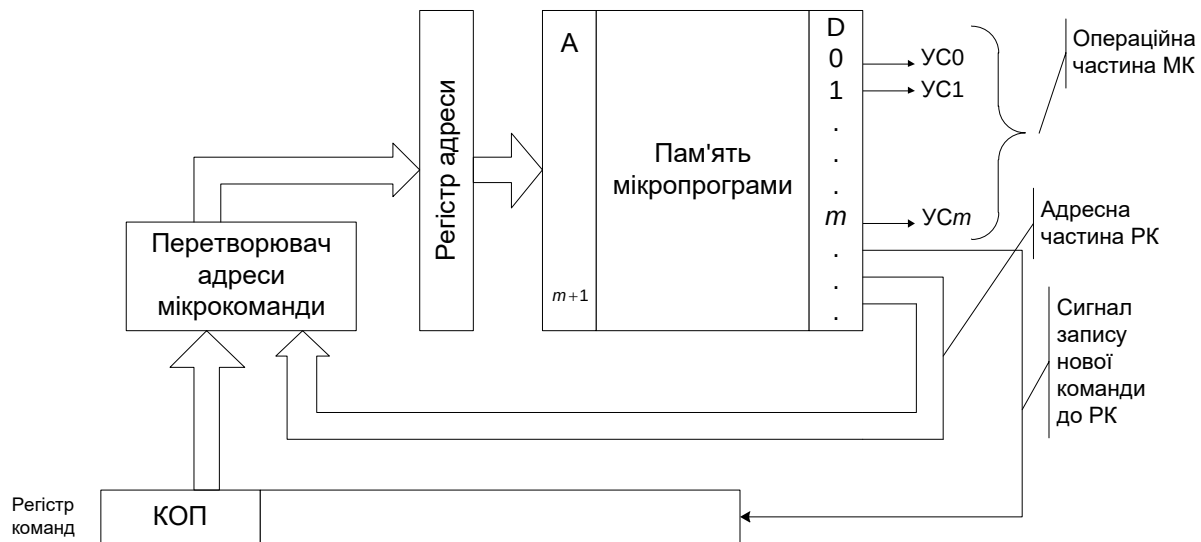


Рис. 10.8. Функціональна схема мікропрограмного пристрою керування (КС і - керуючі сигнали, що виробляються пристроєм керування)

У реальних мікропрограмних пристроях керування формування адреси наступної мікрокоманди проводиться більш складним чином, що враховує можливості розгалуджень і циклічного повторення окремих фрагментів мікропрограми.

Таблиця 10.2

Вбудоване виконання операції множення

Адреса МК	УС1	УС2	УС3	УС4	УС5	УС6	Сигнал запису в РК	Адреса наступної МК
300	1	0	1	0	0	1	0	301
301	0	0	0	0	1	0	0	302
302	0	1	0	1	0	0	0	303
303	0	0	0	0	1	0	0	304
304	0	1	0	1	0	0	1	X

З аналізу структури і принципів роботи *схемного і мікропрограмного пристроїв керування* видно, що ПК першого типу мають складну нерегулярну структуру, яка вимагає спеціальної розробки для кожної системи команд і повинна практично повністю перероблятися за будь-яких модифікацій системи команд. У той же час вона має досить високу швидкодію, яка визначається швидкодією використовуваного елементного базису.

Пристрій керування, реалізований за мікропрограмним принципом, може легко налаштовуватися на можливі зміни в операційній частині ЕОМ. При цьому налаштування багато в чому зводиться лише до заміни *мікропрограмної пам'яті*. Однак ПК цього типу мають гірші часові показники в порівнянні з *пристроями керування* на жорсткій логіці.

10.3 Структура мікропроцесорної системи

Структура мікропроцесорної системи

МПС – складна система, що включає в себе велику кількість різних пристроїв. Основою її є мікропроцесор. Зв'язок пристроїв ЕОМ між собою здійснюється за допомогою поєднань, які в обчислювальній техніці називаються інтерфейсами.

Інтерфейс – це сукупність програмних і апаратних засобів, призначених для передачі інформації між компонентами ЕОМ і включають в себе електронні схеми, лінії, шини і сигнали адрес, даних і управління, алгоритми передачі сигналів і правила інтерпретації сигналів пристроями.

У широкому сенсі інтерфейс включає також механічну частину (сумісність з типороз'ємом) і допоміжні схеми, що забезпечують електричну сумісність пристроїв за рівнями логічних сигналів, вхідним і вихідним струмам і т. д.

Докладне вивчення інтерфейсів і системних шин не входить в завдання даного курсу. Тому ці питання ми будемо розглядати лише з точки зору загального представлення про організацію роботи *мікропроцесорної системи* та принципи взаємодії її складових пристроїв.

Основним засобом організації МПС є магістрально-модульний (рис. 10.9): всі пристрої, включаючи і мікропроцесор, представляються у вигляді модулів, які з'єднуються між собою загальною магістраллю. Обмін інформацією по магістралі задовольняє вимогам деякого загального інтерфейсу, встановленого для магістралі даного типу. Кожен модуль підключається до магістралі за допомогою спеціальних інтерфейсних схем.

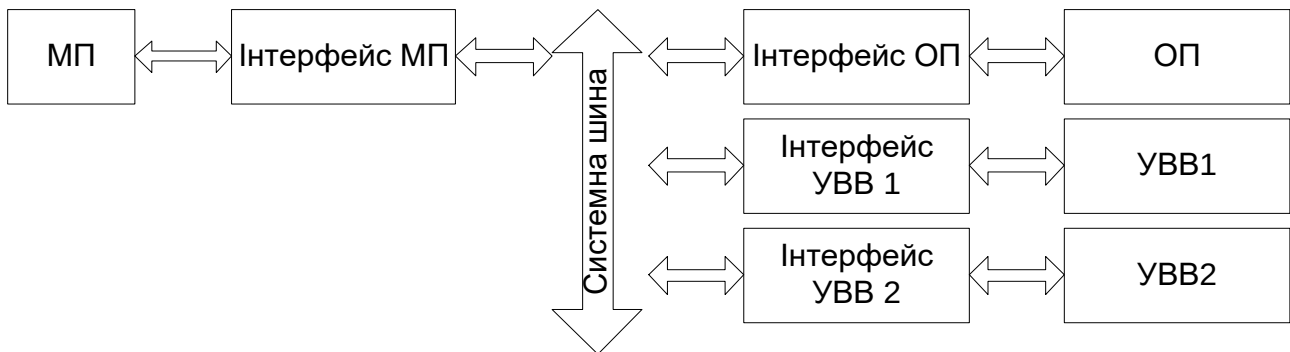


Рис. 10.9. Магістрально-модульний принцип побудови мікропроцесорної системи

На інтерфейсні схеми модулів покладаються такі завдання:

- забезпечення функціональної та електричної сумісності сигналів і протоколів обміну модулів і системної магістралі;
- перетворення внутрішнього формату даних модуля в формат даних системної магістралі і назад;
- забезпечення сприйняття єдиних команд обміну інформацією і перетворення їх в послідовність внутрішніх керуючих сигналів.

Ці інтерфейсні схеми можуть бути досить складними. Зазвичай вони виконуються у вигляді спеціалізованих мікропроцесорних ВІС. Такі схеми прийнято називати *контролерами*.

Контролери мають високий ступінь автономності, що дозволяє забезпечити паралельну в часі роботу периферійних пристроїв і виконання програми обробки даних мікропроцесором.

Крім того, попередньо буферуючи дані, контролери забезпечують пере-силання відразу для багатьох слів, розташованих за адресами, що йдуть підряд, що дозволяє використовувати так званий "вибухового" (burst) режиму роботи шини - 1 цикл адреси і наступні за ним численні цикли даних.

Недоліком магістрально-модульного способу організації ЕОМ є неможливість одночасної взаємодії більше двох модулів, що ставить обмеження на продуктивність комп'ютера [1].

Взаємодію мікропроцесора з оперативною пам'яттю (ОП) і зовнішніми пристроями (ЗП) наведено на рис. 10.10.

Мікропроцесор формує адресу зовнішнього пристрою чи комірки оперативної пам'яті і виробляє керуючі сигнали – або IOR / IOW при зверненні до читання/запису із зовнішнього пристрою, або MR/MW для читання/запису з оперативної пам'яті.

Для обміну інформацією з зовнішніми пристроями в МП є лише 2 команди:

- команда введення IN AX, DX записує в регістр AX число із зовнішнього пристрою, адреса якого знаходиться в регістрі DX, при цьому виробляється сигнал IOR (INput / OUTput Read);

- команда виведення OUT DX, AX виводить інформацію з регістра AX у зовнішній пристрій, адреса якого знаходиться в регістрі DX, при цьому виробляється сигнал IOW (INput / OUTput Write).

Сигнали IOR / IOW формуються при виконанні тільки цих команд.

Формування сигналів MR / MW відбувається в командах, для яких операнд і / або приймач результату розташовуються в оперативній пам'яті, наприклад, ADD [EBX], AX.

У зв'язку з цим можливі два основних способи організації адресного простору *мікропроцесорної системи*:

- із загальним адресним простором зовнішніх пристроїв і оперативної пам'яті;

- з незалежними адресними просторами.

У першому випадку до портів введення/виведення можна звертатися як до комірок оперативної пам'яті.

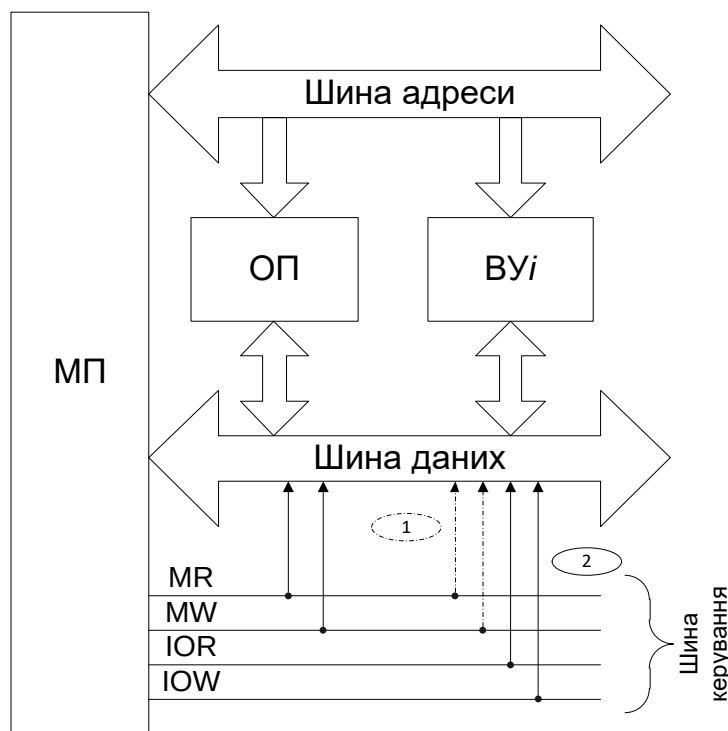


Рис. 10.10. Взаємодія мікропроцесора з оперативною пам'яттю і зовнішніми пристроями

Перевагою такого підходу є можливість використання різних режимів адресації при зверненні до зовнішніх пристроїв, а також виконувати над вмістом портів введення/виведення різних арифметико-логічних операцій. Але в той же час при цьому скорочується ємність ОП, що адресується, і знижується захищеність системи, так як вона позбавляється додаткових засобів захисту, пов'язаних з виконанням команд введення/виведення (не працює поле IOPR реєстра прапорів).

До того ж порушення в логіці роботи програми (формування невірної адреси оперативної пам'яті) може призвести до помилкового спрацювання зовнішнього пристрою.

Якщо перший недолік не настільки істотний при сучасних обсягах запам'ятовуючих пристроїв, то другий може дуже негативно позначитися на роботі

МПС. Можливість використання складних режимів адресації при зверненні до зовнішніх пристроїв для мікропроцесорних систем на основі універсальних МП не настільки важлива. Тому в даний час при побудові МПС перевага віддається другому підходу.

Розглянемо особливості обміну інформацією мікропроцесора з зовнішніми пристроями. Спрощену часову діаграму цього процесу наведено на рис.10.11.

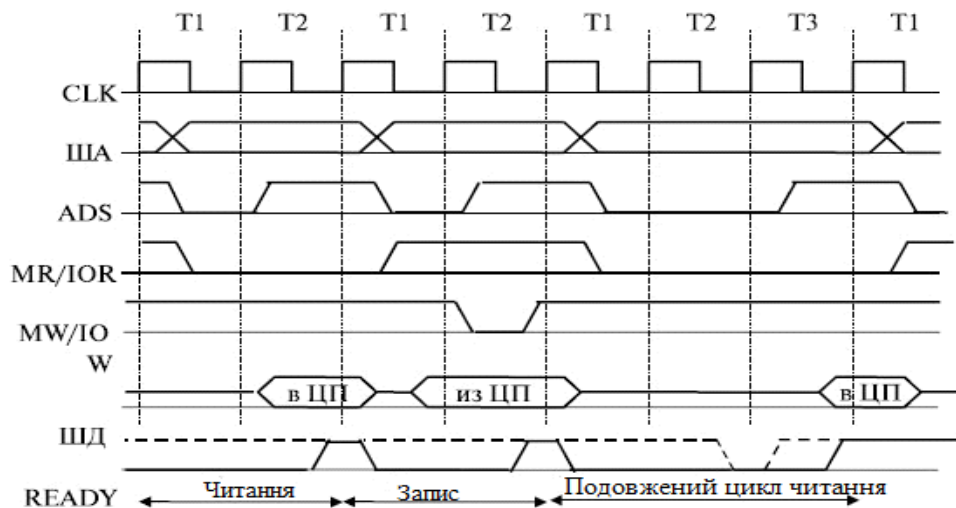


Рис. 10.11. Часова діаграма обміну інформацією між мікропроцесором і зовнішнім пристроєм

При такому обміні мікропроцесор використовує стандартний цикл шини тривалістю 2 періоди тактового генератора, початок якого визначається сигналом ADS шини. Однак якщо в обміні задіяні відносно повільні зовнішні пристрої або оперативна пам'ять, які не встигають завершити обмін у поточному циклі роботи мікропроцесора, то вони скидають сигнал READY, що перевіряється мікропроцесором в кінці другого тактового сигналу циклу.

При $READY = 0$ МП вставляє додаткові такти очікування в цикл шини, щоб дати можливість зовнішньому пристрою завершити операцію введення/виведення. Проте сигнал неготовності зовнішнього пристрою не може тривати більше встановленого часу (для шини ISA - 2,5 мкс), в іншому випадку генерується переривання помилково введення/виведення. Це запобігає зависанню системи у випадку, наприклад, механічного пошкодження даного контакту.

У загальному випадку стандартний цикл шини при обміні вже включає декілька додаткових тактів очікування. При цьому цикли звертання до пам'яті коротше циклів звернення до портів введення/виведення. Мінімальна тривалість циклу визначається чипсетом і може програмуватися опціями BIOS Setup завданням тактів очікування. Якщо ОП або зовнішній пристрій не потребують додаткових тактів, то їх контролери виставляють на шину сигнал OWS, за яким додаткові такти не вставляються.

Прямий доступ до пам'яті

У МПС використовуються два основних способи організації передачі даних між пам'яттю і периферійними пристроями: *програмно керована передача і прямий доступ до пам'яті* ПДП (Direct Memory Access - DMA).

Програмно керована передача даних здійснюється за безпосередньої участі і під управлінням процесора. Наприклад, при пересиланні блоку даних із зовнішнього пристрою в оперативну пам'ять процесор повинен виконати наступну послідовність кроків:

1. сформувати початкову адресу області обміну ОП;
2. занести довжину переданого масиву даних в один зі своїх внутрішніх реєстрів, який буде грати роль лічильника;
3. видати команду читання інформації з ВП, при цьому на шину адреси з МП видається адреса ВП, на шину управління - сигнал читання даних з ВП, а лічені дані заносяться у внутрішній реєстр МП;
4. видати команду запису інформації в ОП; при цьому на шину адреси з МП видається адреса комірки оперативної пам'яті, на шину управління - сигнал запису даних в ОП, а на шину даних виставляються дані з реєстра МП, в який вони були поміщені при читанні з ВП;
5. модифікувати реєстр, який містить адресу оперативної пам'яті;
6. зменшити лічильник довжини масиву на довжину переданих даних;
7. якщо передані не всі дані, то повторити кроки 3-6, в іншому випадку закінчити обмін.

Як ми бачимо, програмно керований обмін призводить до нераціонального використання потужності мікропроцесора, який змушений виконувати велику кількість відносно простих операцій, припиняючи роботу над основною програмою. При цьому дії, пов'язані зі зверненням до оперативної пам'яті і до зовнішнього пристрою, зазвичай вимагають подовженого циклу роботи мікропроцесора, що призводить до ще більш істотних втрат продуктивності.

Альтернативою програмно керованого обміну служить *прямий доступ до пам'яті* – спосіб швидкодіючого підключення зовнішніх пристроїв, при якому він звертається до оперативної пам'яті, не перериваючи роботи процесора.

Такий обмін відбувається під керуванням окремого пристрою – *контролера прямого доступу до пам'яті* (КПДП).

Схему включення КПДП до складу мікропроцесорної системи наведено на рис. 10.12.

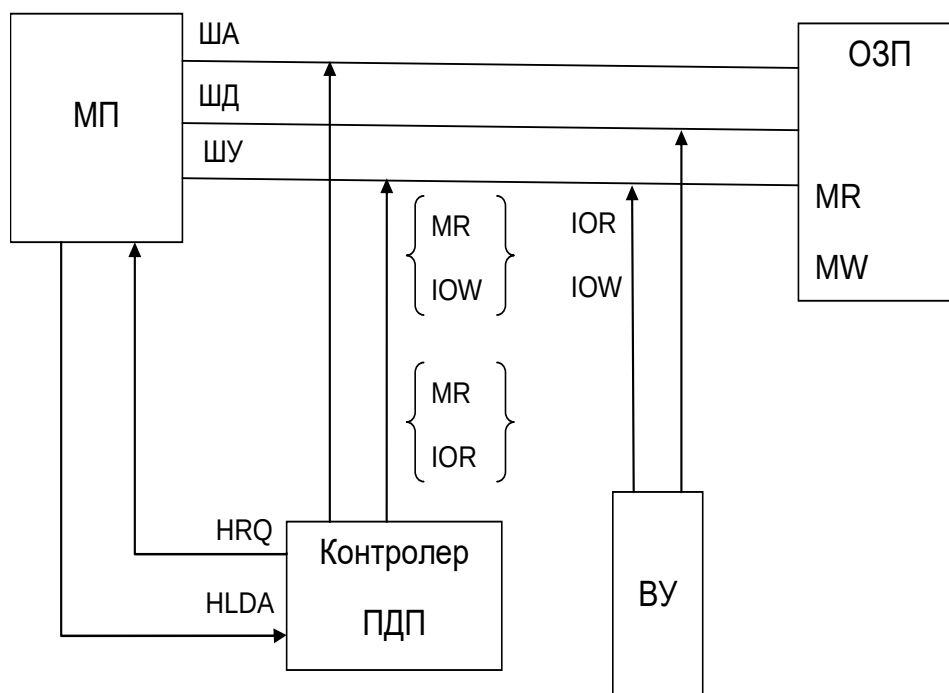


Рис. 10.12. Структура МШС з контролером прямого доступу до пам'яті

Перед початком роботи *контролер ПДП* необхідно ініціалізувати: занести початкову адресу області ОП, з якою проводиться обмін, і довжину переданого масиву даних. Надалі по сигналу запиту прямого доступу контролер фактично

виконує всі ті дії, які забезпечував мікропроцесор при програмно керованій передачі.

Послідовність дій КПДП при запиті на прямий доступ до пам'яті з боку зовнішнього пристрою наступна:

1. Прийняти запит на ПДП (сигнал DRQ) від ВП.
2. Сформуванати запит до МП на захоплення шин (сигнал HRQ).
3. Прийняти сигнал від МП (HLDA), що підтверджує факт переведення мікропроцесором своїх шин в третій стан.
4. Сформуванати сигнал, що повідомляє пристрою введення-виведення про початок виконання циклів прямого доступу до пам'яті (DACK).
5. Сформуванати на шині адреси комп'ютера адресу комірки пам'яті, призначеної для обміну.
6. Виробити сигнали, що забезпечують управління обміном (IOR, MW для передачі даних з ВП в оперативну пам'ять і IOW, MR для передачі даних з оперативної пам'яті в ВП).
7. Зменшити значення в лічильнику даних на довжину переданих даних.
8. Перевірити умову закінчення сеансу прямого доступу (обнулення лічильника даних або зняття сигналу запиту на ПДП). Якщо умову закінчення не виконано, то змінити адресу в регістрі поточної адреси на довжину переданих даних і повторити кроки 5-8.

9. Прямий доступ до пам'яті дозволяє здійснювати обмін даними між зовнішнім пристроєм і оперативною пам'яттю паралельно з виконанням процесором програми. Структуру КПДП наведено на рис. 10.13.

Контролер складається з 4 ідентичних каналів, що дозволяють підключити до системи до чотирьох пристроїв, що працюють в режимі ПДП.

До складу кожного каналу входять такі регістри.

1. MR (Mode Register) – регістр режиму. Визначає наступні параметри передачі:

- порядок зміни (збільшення або зменшення) адреси ОП при передачі;
- можливість автоініціалізації;

- режим обслуговування;
- одиночна передача (контролер повертає магістраль процесору після кожного циклу ПДП);
- блокова передача (контролер володіє магістраллю протягом передачі всього масиву);
- на вимогу (закінчення передачі визначається зняттям сигналу DRQ або подачею сигналу IEOP на зовнішній вхід КППД);
- каскадування;

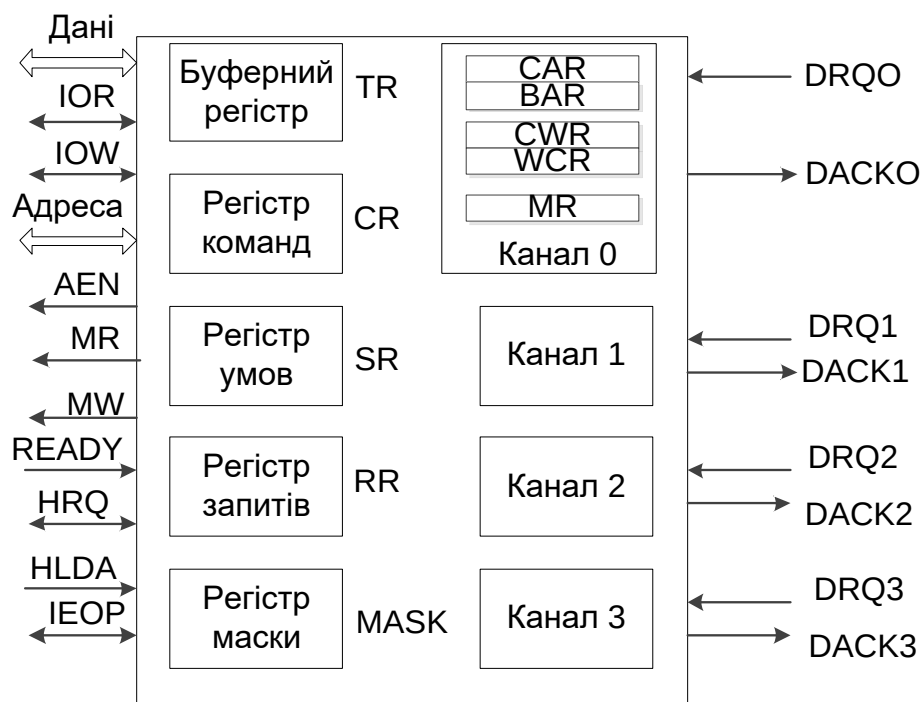


Рис. 10.13. Структура контролера прямого доступу до пам'яті

2. CAR (Current Address Register) – реєстр поточної адреси;
3. BAR (Base Address Register) – базовий реєстр адреси;
4. CWR (Current Word Register) – поточний лічильник даних;
5. WCR (W0ord ConTRol Register) – базовий лічильник даних.

Керуючі реєстри, спільні для всього контролера.

1. CR (Command Register) – реєстр команд – визначає: режим пам'ять-пам'ять або звичайний. У режимі пам'ять-пам'ять здійснюється обмін за схемою

ПДП між двома областями ОП (тільки для каналів 0 і 1) з використанням буферного регістру TR (Temporary Register);

2. заборона / дозвіл ПДП;
3. порядок зміни пріоритетів каналів;
4. фіксований пріоритет;
5. циклічна зміна пріоритету після обробки запиту на ПДП по одному з каналів;
6. явна вказівка найбільш пріоритетного каналу;
7. рівень сигналів DRQ і DACK (настройка на активний рівень сигналів під особливості роботи зовнішніх пристроїв);

SR (Status Register) – регістр умов: використовується для фіксації ознак обміну по кожному з каналів і програмних запитів на ПДП.

Щоб уникнути помилкових спрацьовувань зовнішніх пристроїв, що не використовують в даний момент режим прямого доступу, контролер ПДП під час режиму ПДП виробляє сигнал AEN, який блокує роботу інших зовнішніх пристроїв:

$$AEN = \begin{cases} 1 - \text{шиной керує контролер ПДП} \\ 0 - \text{шиной керує МП або інший задатчик} \end{cases}$$

Для збільшення кількості зовнішніх пристроїв, які можуть бути підключені до мікропроцесора в режимі ПДП, використовується каскадне включення КПДП (рис. 10.14).

При цьому відомий КПДП підключається до одного з каналів ведучого контролера за схемою підключення зовнішніх пристроїв, а безпосередньо з мікропроцесором зв'язується тільки той контролер, що веде. У стандартній конфігурації персональної ЕОМ застосовуються два КПДП (той, що веде і той, якого ведуть), які дозволяють підключити до 7 зовнішніх пристроїв, причому 2 канали закріплені за накопичувачами на жорсткому і гнучкому дисках. У максимальній конфігурації при використанні того, що веде і 4 тих, яких ведуть КПДП,

підключених до кожного з каналів того, що веде, до МП можна підключити до 16 зовнішніх пристроїв, що працюють в режимі ПДП [2].

Призначення і функції чипсета в мікропроцесорній системі

Чипсет (CHIPset) - це набір ВІС (зазвичай 1-3 мікросхеми), функціонально еквівалентний мікросхемам, що входять в стандартну конфігурацію мікропроцесорної системи.

Як правило, чипсет інтегрує в собі функції таких пристроїв:

- контролера оперативної пам'яті;
- контролерів кеш-пам'яті 2-го і / або 3-го рівня;
- контролерів ПДП;
- контролерів пріоритетних переривань;
- контролера клавіатури;
- контролера миші PS/2;
- контролера інфрачервоного порту;
- таймера реального часу;
- моста шини PCI;
- моста шини ISA і ін.

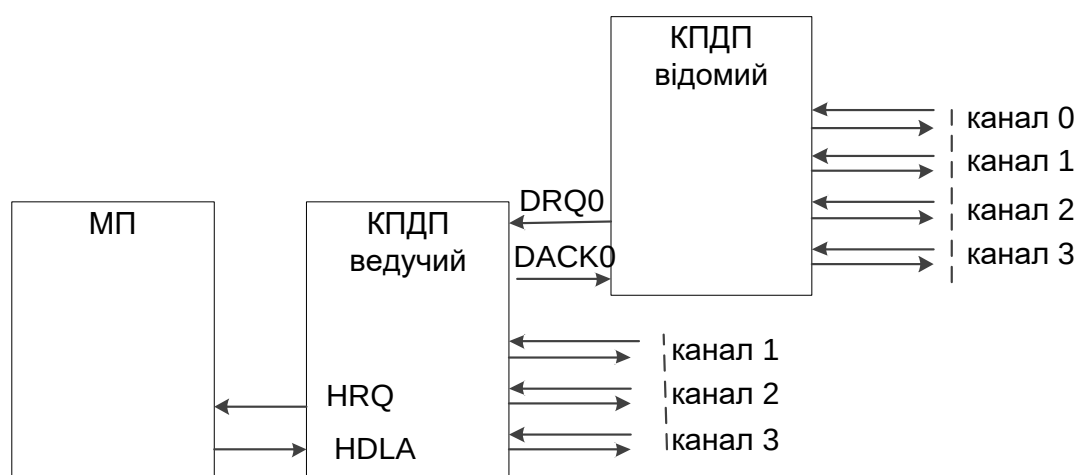


Рис. 10.14. Каскадне включення контролерів прямого доступу до пам'яті

Зазвичай у складі чипсета виділяють:

- північний міст (North Bridge) – системний контролер, в який входить контролер системної шини, шин AGP і PCI, ОЗП і кеш-пам'яті (для наборів під звичайний Pentium);
- південний міст (SOUTH Bridge) – периферійний контролер, що включає контролери EIDE, клавіатури, моста PCI-to-PCI, послідовних / паралельних портів, шини USB і інших подібних пристроїв.

Вибір чипсету багато в чому визначає конфігурацію МПС та її продуктивність. Якщо МП можна замінити, а ємність ОЗП збільшити, то заміна чипсету однозначно пов'язана з заміною системної плати, а обмеження чипсету також однозначно обмежують можливості заміни інших елементів МПС: МП, ОЗП, зовнішніх пристроїв.

Чипсет накладає обмеження на наступні функціональні характеристики системи в цілому: тип пам'яті, тип кеш-пам'яті другого і / або третього рівня, тип МП, максимальна частота системної шини, тип шини PCI (32 - або 64-розрядна); підтримка багатопроцесорної конфігурації і деякі інші характеристики.

Практика показує, що різниця в продуктивності системних плат різних фірм, побудованих із застосуванням одного і того ж чипсету, складає, від сили, кілька відсотків, тим часом як той же параметр для різних чипсетів може відрізнятися на порядок.

Розглянемо використання чипсету на прикладі організації мікропроцесорної системи на базі МП Pentium III на ядрі Katmai (рис. 10.15).

Частота шини PCI складає 33 МГц і при частоті системної шини і 66, і 100 МГц (використовується розподіл частоти на 2 і 3 відповідно). AGP (Accelerated Graphics Port) – 32-розрядна шина (тактова частота завжди 66 МГц), що використовується для підключення графічного адаптера і дозволяє прискорити обмін даними між графічним адаптером і основною пам'яттю. У режимі AGP 2х ефективна частота в 133 МГц забезпечується використанням обох фронтів імпульсів частоти 66 МГц. Шина ISA працює на частоті 8 МГц. За її функціону-

вання відповідає міст PCI/ISA, що є звичайним PCI-пристроєм (просто вбудованим в чипсет).

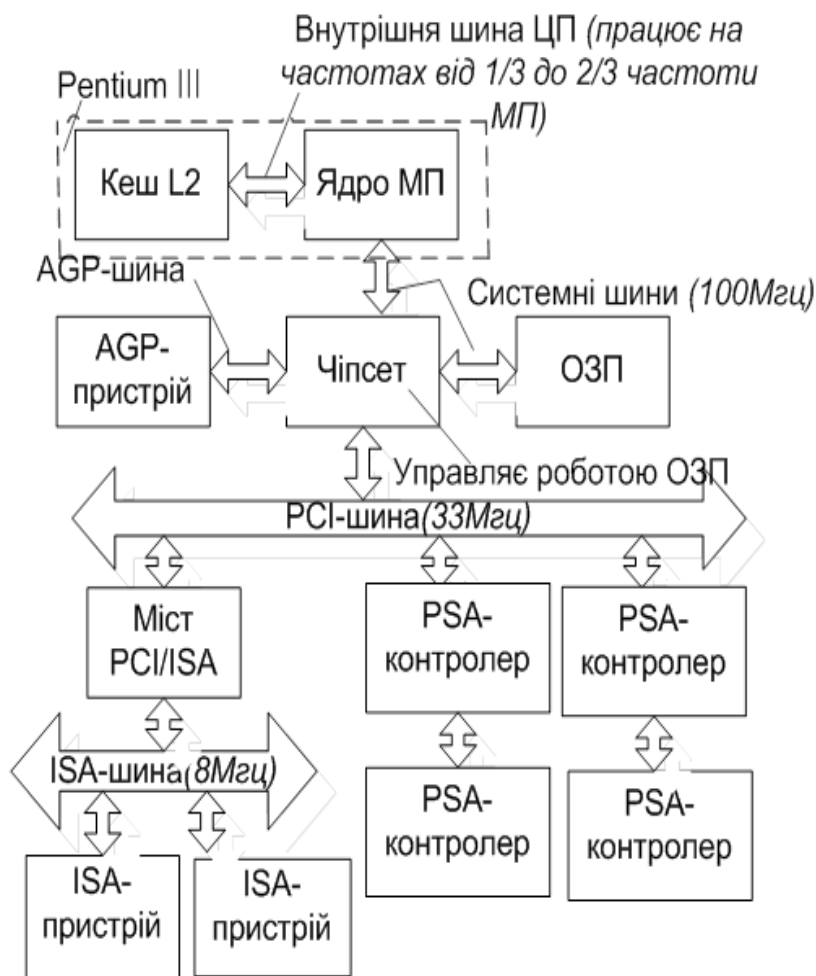


Рис. 10.15. Структура мікропроцесорної системи з використанням чипсета

Системна шина може працювати на різній частоті, але всі ці варіанти сумісні і з розводки, і електрично, що створює ґрунт для "розгону".

Таким чином, при зміні частоти системної шини, наприклад, з 66 на 100 МГц заміні підлягають лише три пристрої: процесор, чипсет і пам'ять.

Ніякі інші пристрої не помітять підміни шини - ні AGP відеоадаптери, ні будь-який з PCI-контролерів, ні старі ISA-картки. У цьому й полягає принципова відмінність шини 100 МГц від шини 66 МГц, «розігнаної» до 75/83 МГц. У першому випадку новий чипсет забезпечує роботу PCI на частоті 33 (100 / 3) МГц, у другому старий чипсет і не підозрює про "розгоні" і продовжує ділити

частоту системної шини на 2, а в підсумку частота PCI-шини дорівнює 37,5 або 41,5 МГц, що може призвести до неправильної роботи деяких PCI-пристроїв.

10.4 Введення-виведення інформації

Інтерфейс – це сукупність програмних і апаратних засобів, призначених для передачі інформації між компонентами ЕОМ і включають в себе електронні схеми, лінії, шини і сигнали адрес, даних і управління, алгоритми передачі сигналів і правила інтерпретації сигналів пристроями.

Інтерфейси характеризуються наступними параметрами:

- пропускна здатність – кількість інформації, яка може бути передана через *інтерфейс* в одиницю часу;
- максимальна частота передачі інформаційних сигналів через *інтерфейс*;
- максимально допустима відстань між пристроями, сполучаються;
- загальне число дротів (ліній) в *інтерфейсі*;
- інформаційна ширина *інтерфейсу* – число біт або байт даних, переданих паралельно через *інтерфейс*.

До динамічних параметрів *інтерфейсу* належить час передачі окремого слова і блоку даних з урахуванням тривалості процедур підготовки та завершення передачі.

Розробка систем введення-виведення вимагає вирішення цілого ряду проблем, серед яких виділимо наступні:

- необхідно забезпечити можливість реалізації ЕОМ зі змінним складом обладнання, в першу чергу, з різним набором пристроїв введення-виведення, з тим, щоб користувач міг вибирати конфігурацію машини відповідно до її призначення, легко додавати нові пристрої і відключати ті, використання яких йому не потрібно;
- для ефективного і високопродуктивного використання обладнання комп'ютера слід реалізувати паралельну в часі роботу процесора над обчислю-

вальною частиною програми та виконання периферійними пристроями процедур введення-виведення;

- необхідно спростити для користувача і стандартизувати програмування операцій введення-виведення, забезпечити незалежність програмування введення-виведення від особливостей того чи іншого периферійного пристрою;
- в ЕОМ повинно бути забезпечено автоматичне розпізнавання і реакція процесора на різноманіття ситуацій, що виникають у ПВВ (готовність пристрою, відсутність носія, різні порушення нормальної роботи та ін.).

Головним напрямком вирішення зазначених проблем є *магістрально-модульний спосіб* побудови ЕОМ (рис. 10.16): всі пристрої, з яких складається комп'ютер, включаючи і мікропроцесор, організовуються у вигляді модулів, які з'єднуються між собою загальною магістраллю. Обмін інформацією по магістралі задовільняє вимогам деякого загального *інтерфейсу*, встановленого для магістралі даного типу. Кожен модуль підключається до магістралі за допомогою спеціальних інтерфейсних схем.

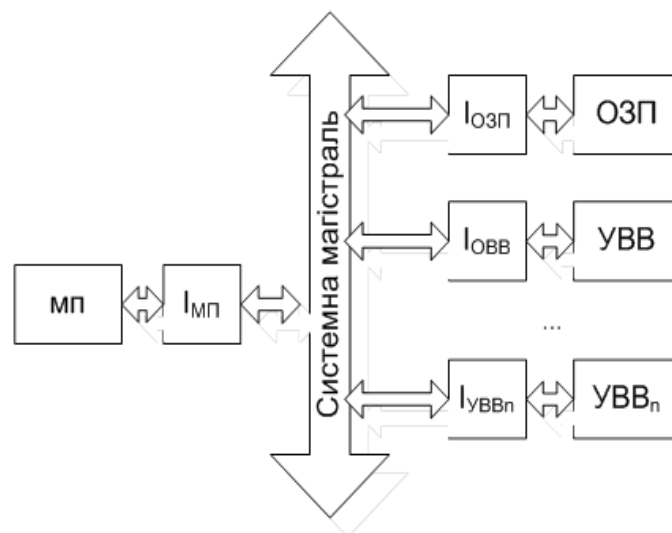


Рис. 10.16. Магістрально-модульний принцип побудови ЕОМ

На інтерфейсні схеми модулів покладаються такі завдання:

- забезпечення функціональної та електричної сумісності сигналів і протоколів обміну модуля і системної магістралі;

- перетворення внутрішнього формату даних модуля в формат даних системної магістралі і назад;
- забезпечення сприйняття єдиних команд обміну інформацією і перетворення їх в послідовність внутрішніх керуючих сигналів.

Ці інтерфейсні схеми можуть бути досить складними і за своїми можливостями відповідати універсальним мікропроцесорам. Такі схеми прийнято називати контролерами.

Контролери мають високий ступінь автономності, що дозволяє забезпечити паралельну в часі роботу периферійних пристроїв і виконання програми обробки даних мікропроцесором.

Недоліком *магістрально-модульного способу* організації ЕОМ є неможливість одночасної взаємодії більше двох модулів, що ставить обмеження на продуктивність комп'ютера. Тому цей спосіб, в основному, використовується в ЕОМ, до характеристик яких не висовуються дуже високі вимоги, наприклад в персональних ЕОМ.

У ЕОМ використовуються два основних способи організації передачі даних між пам'яттю і периферійними пристроями: *програмно-керована передача* і *прямий доступ до пам'яті (ПДП)*.

Програмно-керована передача даних здійснюється за безпосередньої участі і під управлінням процесора. Наприклад, при пересиланні блоку даних з периферійного пристрою в оперативну пам'ять процесор повинен виконати наступну послідовність кроків:

1. сформувані початкову адресу області обміну ОП;
2. занести довжину переданого масиву даних в один з внутрішніх регістрів, який буде відігравати роль лічильника;
3. видати команду читання інформації з ПВВ; при цьому на шину адреси з МП видається адреса ПВВ, на шину управління - сигнал читання даних з ПВВ, а лічені дані заносяться у внутрішній регістр МП;
4. видати команду запису інформації в ОП; при цьому на шину адреси з МП видається адреса комірки оперативної пам'яті, на шину керування - сигнал

запису даних в ОП, а на шину даних виставляються дані з регістра МП, в який вони були поміщені при читанні з ПВВ;

5. модифікувати регістр, що містить адресу оперативної пам'яті;

6. зменшити лічильник довжини масиву на довжину переданих даних;

7. якщо передані не всі дані, то повторити кроки 3-6, в іншому випадку закінчити обмін.

Як видно, *програмно-керований обмін* веде до нераціонального використання потужності мікропроцесора, який змушений виконувати велику кількість відносно простих операцій, припиняючи роботу над основною програмою. При цьому дії, пов'язані зі зверненням до оперативної пам'яті і до периферійного пристрою, зазвичай вимагають подовженого циклу роботи мікропроцесора через їх більш повільно, порівняно з мікропроцесором роботи, що призводить до ще більш істотних втрат продуктивності ЕОМ.

Альтернативою *програмно-керованого обміну* служить *прямий доступ до пам'яті* – спосіб швидкодіючого підключення зовнішніх пристроїв, при якому воно звертається до оперативної пам'яті, не перериваючи роботи процесора. Такий обмін відбувається під керуванням окремого пристрою – *контролера прямого доступу до пам'яті (КПДП)*.

Структура ЕОМ, що має в своєму складі *КПДП*, представлена на рис. 10.17.

Перед початком роботи *контролер ПДП* необхідно ініціалізувати: занести початкову адресу області ОП, з якою проводиться обмін, і довжину переданого масиву даних. Надалі по сигналу запиту прямого доступу контролер фактично виконує всі ті дії, які забезпечував мікропроцесор при *програмно-керованій передачі*.

Послідовність дій *КПДП* при запиті на *прямий доступ до пам'яті* з боку пристроїв введення-виведення наступна:

1. Прийняти запит на *ПДП* (сигнал DRQ) від ПВВ.

2. Сформувані запит до МП на захоплення шин (сигнал HRQ).

3. Прийняти сигнал від МП (HLDA), що підтверджує факт переведення мікропроцесором своїх шин в третій стан.
4. Сформувати сигнал, що повідомляє пристрою введення-виведення про початок виконання циклів *прямого доступу до пам'яті* (DACK).
5. Сформувати на шині адреси комп'ютера адресу комірки пам'яті, призначеної для обміну.
6. Виробити сигнали, що забезпечують управління обміном (IOR, MW для передачі даних з ПБВ в оперативну пам'ять і IOW, MR для передачі даних з оперативної пам'яті в УБВ).
7. Зменшити значення в лічильнику даних на довжину переданих даних.
8. Перевірити умову закінчення сеансу прямого доступу (обнуління лічильника даних або зняття сигналу запиту на ПДП). Якщо умову закінчення не виконано, то змінити адресу в регістрі поточної адреси на довжину переданих даних і повторити кроки 5-8.

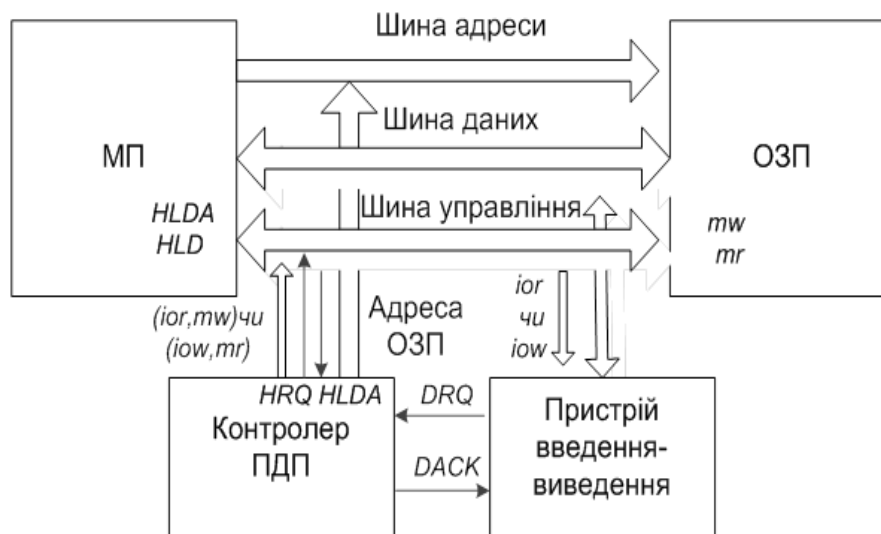


Рис. 10.17. Обмін даними в режимі прямого доступу до пам'яті

Прямий доступ до пам'яті дозволяє здійснювати паралельно в часі виконання процесором програми і обмін даними між периферійним пристроєм і оперативною пам'яттю.

Зазвичай програмно-керований обмін використовується в ЕОМ для операцій введення-виведення окремих байт (слів), які виконуються швидше, ніж при ПДП, оскільки виключаються втрати часу на ініціалізацію контролера ПДП, а в якості основного способу здійснення операцій введення-виведення використовують ПДП. Наприклад, у стандартній конфігурації персональної ЕОМ обмін між накопичувачами на магнітних дисках і оперативною пам'яттю відбувається в режимі *прямого доступу*.

Як зазначалося вище, зазвичай комп'ютер будується за *магістрально-модульним принципом*. При цьому всі складові його пристрою об'єднуються загальною шиною, по якій між ними відбувається обмін даними, адресами, а також керівними сигналами. Як приклад перелічимо основні лінії, що становлять одну з поширених системних магістралей – *шину ISA*:

- A0-A23 – шина адреси;
- D0-D15 – двонаправлена шина даних, допускає обмін як байтами, так і словами (2 байти);
- CLK – шинний тактовий сигнал, синхронізує роботу процесора, ОП і ПБВ;
- MR – керуючий сигнал читання з ОП;
- MW – керуючий сигнал запису в ОП;
- IOR – керуючий сигнал читання з ПБВ;
- IOW – керуючий сигнал запису в ПБВ;
- IRQ_i – запит переривання від i -го джерела;
- DRQ_i – запит *прямого доступу до пам'яті* за i -м каналом контролера ПДП;
- $DACK_i$ – дозвіл *прямого доступу до пам'яті* i -му каналу контролера ПДП;
- AEN – сигнал зайнятості шин обміном в режимі ПДП;
- READY – сигнал готовності ПБВ до обміну.

Магістраль забезпечує підключення до семи зовнішніх пристроїв, що працюють в режимі *прямого доступу до пам'яті*, і до 11 запитів переривань від

ПВВ. Ще чотири запити переривань зарезервовані за пристроями, що входять до складу стандартної конфігурації ЕОМ, і на магістраль не виведені.

Хоча *шина ISA* має невелику інформаційну ширину і в даний час використовується для підключення тільки до повільних периферійних пристроїв, її склад дозволяє певною мірою простежити взаємозв'язок різних розглянутих раніше пристроїв складових ЕОМ [11].

Організація ЕОМ на основі загальної шини є стримуючим фактором для підвищення продуктивності комп'ютера.

Слід зазначити, що навіть при використанні *прямого доступу до пам'яті* процесор повністю не звільняється від управління операціями введення-виведення.

Він забезпечує ініціалізацію *контролера ПДП*, а також взаємодіє з ним по деяким управляючим лініям. Більш того, під час операції передачі даних *інтерфейс* виявляється зайнятим, а зв'язок процесора з оперативною пам'яттю – блокованим.

Це істотно позначається на ефективності роботи ЕОМ, особливо в тих випадках, коли в обчислювальній системі використовується велика кількість високошвидкісних зовнішніх пристроїв.

Для вирішення цієї проблеми до складу високопродуктивних комп'ютерів іноді включають спеціалізовані процесори введення-виведення, здатні повністю розвантажити основний процесор від управління операціями обміну з зовнішніми пристроями.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке АЛП? Як проходить синтез АЛП?
2. Що таке пристрій керування. Які види ПК ви знаєте? Чим вони відрізняються?
3. Які складові частини схемного пристрою керування ви знаєте? Зазначте їх функції.
4. Що таке мікрокоманда? З чого складається мікрокоманда?

5. Опишіть алгоритм роботи мікропрограмного пристрою керування.
6. Дайте визначення поняттю інтерфейс та опишіть завдання, що покладаються на інтерфейсні схеми модулів.
7. Скільки основних способів організації адресного простору мікропроцесорної системи ви знаєте? В чому полягає логіка їх роботи?
8. Яку послідовність кроків повинен виконати процесор при пересиланні блоку даних із зовнішнього пристрою в оперативну пам'ять?
9. Опишіть принцип роботи датчика сигналів.
10. Зазначте два основних способи організації адресного простору мікропроцесорної системи.
11. Що таке чипсет? Призначення і функції чипсета в мікропроцесорній системі.
12. Дайте визначення поняттю «інтерфейс». Зазначте основні його параметри.
13. Які два основних способи організації передачі даних між пам'яттю і периферійними пристроями використовуються в ЕОМ?
14. Зазначте основні недоліки магістрально-модульного способу організації ЕОМ.
15. Яким чином здійснюється програмно керована передача даних?
16. Який спосіб є альтернативою програмно керованого обміну?
17. Що таке керуючий регістр? Які керуючі регістри ви знаєте?
18. Які параметри динамічного інтерфейсу ви знаєте?
19. Як відбувається ініціалізація контролера ПДП?
20. Дайте визначення поняттю «мікропроцесорна система»?

11. ЗАПАМ'ЯТОВУЮЧІ ПРИСТРОЇ

11.1. Запам'ятовуючі пристрої

Пам'яттю ЕОМ називається сукупність пристроїв, що служать для запам'ятовування, зберігання і видачі інформації.

Окремі пристрої, що входять в цю сукупність, називаються *запам'ятовуючими пристроями (ЗП)* того чи іншого типу.

Термін "*запам'ятовуючий пристрій*" зазвичай використовується, коли мова йде про принцип побудови деякого пристрою пам'яті (наприклад, напівпровідниковий ЗП, ЗП на жорсткому магнітному диску і т.п.), а термін «*пам'ять*» – коли хочуть підкреслити виконувану пристроєм пам'яті логічну функцію або місце розташування в складі устаткування ЕОМ (наприклад, *оперативна пам'ять* – ОП, зовнішня пам'ять і т.п.).

У тих питаннях, де ці відмінності не мають принципового значення, терміни «*пам'ять*» і «*запам'ятовуючий пристрій*» ми будемо використовувати як синоніми.

Запам'ятовуючі пристрої відіграють важливу роль в загальній структурі ЕОМ. За деякими оцінками продуктивність комп'ютера на різних класах завдань на 40-50% визначається характеристиками ЗП різних типів, що входять до його складу.

До основних параметрів, що характеризують *запам'ятовуючі пристрої*, відносять *ємність* і *швидкодію*.

Ємність пам'яті – це максимальна кількість даних, що в ній може зберігатися.

Ємність накопичувача вимірюється кількістю адресованих елементів (комірок) ЗП і довжиною комірки в бітах. Зараз практично всі *запам'ятовуючі пристрої* в якості елемента, що адресується мінімально, використовують 1 байт (1 байт = 8 двійкових розрядів (біт)).

Тому *ємність пам'яті* зазвичай визначається в байтах, кілобайтах (1Кбайт = 2^{10} байт), мегабайтах (1Мбайт = 2^{20} байт), гігабайтах (1Гбайт = 2^{30} байт) і т.д.

За одне звернення до *запам'ятовуючого пристрою* проводиться зчитування або запис деякої одиниці даних, яку називають *словом*, різної для пристроїв різного типу.

Це визначає різну *організацію пам'яті*. Наприклад, пам'ять об'ємом 1 мегабайт може бути організована як 1М слів по 1 байту, або 512К слів по 2 байти кожне, або 256К слів по 4 байти і т.д.

У той же час, у кожній ЕОМ використовується своє поняття *машинного слова*, яке застосовується при визначенні архітектури комп'ютера, зокрема при його програмуванні, і не залежить від розмірності слова пам'яті, що використовується для побудови даної ЕОМ. Наприклад, комп'ютери з архітектурою IBM PC мають машинне слово завдовжки 2 байти.

Швидкодія пам'яті визначається тривалістю операції звернення, тобто часом, що витрачається на пошук потрібної інформації в пам'яті і на її зчитування, або часом на пошук місця в пам'яті, що призначається для зберігання даної інформації, і на її запис:

$$t_{обр} = \max(t_{обрсч}, t_{обрзн}),$$

де $t_{обрсч}$ – швидкодія ЗП при зчитуванні інформації;

$t_{обрзн}$ – швидкодія ЗП при записі.

Класифікація запам'ятовуючих пристроїв

Запам'ятовуючі пристрої можна класифікувати по цілому ряду параметрів і ознак. На рис. 11.1 наведено класифікацію за типом звертання та організацією доступу до комірок ЗП.

За типом звертання ЗП діляться на пристрої, що допускають як читання, так і запис інформації, і постійні *запам'ятовуючі пристрої* (ПЗП), призначені тільки для читання записаних в них даних (*ROM* – read only memory).

ЗП першого типу використовуються в процесі роботи процесора для зберігання виконуваних програм, вихідних даних, проміжних і остаточних результатів. У ПЗП, як правило, зберігаються системні програми, необхідні для запуску комп'ютера в роботу, а також константи.

У деяких ЕОМ, призначених, наприклад, для роботи в системах управління з одним і тим же незмінним алгоритмом, все програмне забезпечення може зберігатися в ПЗП.

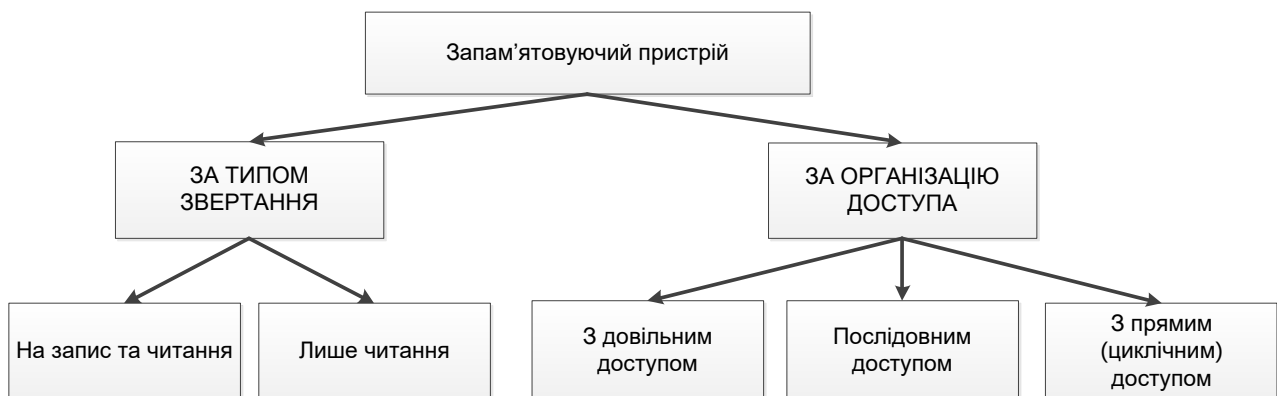


Рис. 11.1. Класифікація запам'ятовуючих пристроїв

У ЗП з довільним доступом (RAM – random access memory) час доступу не залежить від місця розташування комірки пам'яті (наприклад, ОЗП).

У ЗП з прямим (циклічним) доступом з авдяки безперервному обертанню носія інформації (наприклад, магнітний диск – МД) можливість звернення до деякої ділянки носія циклічно повторюється. Час доступу тут залежить від взаємного розташування цієї ділянки і головок читання/запису і багато в чому визначається швидкістю обертання носія.

У ЗП з послідовним доступом проводиться послідовний перегляд ділянок носія інформації, поки потрібну ділянку не займе деяке потрібне положення напроти головок читання/запису (наприклад, магнітні стрічки – МС).

Як зазначалося вище, основні характеристики запам'ятовуючих пристроїв - це ємність і швидкодія. Ідеальний запам'ятовуючий пристрій повинен володіти нескінченно великою ємністю і мати нескінченно малий час звернення.

На практиці ці параметри знаходяться в суперечності один одному: в рамках одного типу *ЗП* поліпшення одного з них веде до погіршення значення іншого. До того ж слід мати на увазі і економічну доцільність побудови *запам'ятовуючого пристрою* з тими чи іншими характеристиками при даному рівні розвитку технології.

Тому в даний час *запам'ятовуючі пристрої* комп'ютера, як це і передбачав Нейман, будуються за *ієрархічним принципом* (рис. 11.2).

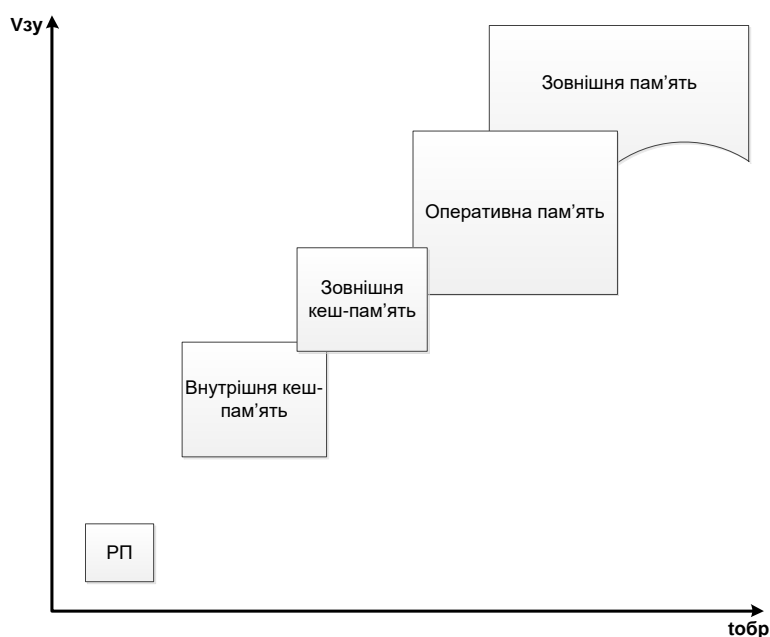


Рис. 11.2. Ієрархічна організація пам'яті в сучасних ЕОМ

Ієрархічна структура пам'яті дозволяє економічно ефективно поєднувати зберігання великих обсягів інформації з швидким доступом до інформації в процесі її обробки.

На нижньому рівні ієрархії знаходиться *регістрова пам'ять* – набір регістрів, що входять безпосередньо до складу мікропроцесора (центрального процесора – CPU). Регістри CPU програмно доступні і зберігають інформацію, що найбільш часто використовується при виконанні програми: проміжні результати, складові частини адрес, лічильники циклів і т.д.

Реєстрова пам'ять має відносно невеликий об'єм (до декількох десятків машинних слів). РП працює на частоті процесора, тому час доступу до неї мінімальний.

Наприклад, при частоті роботи процесора 2 ГГц час звернення до його реєстрів складе всього 0,5 нс.

Оперативна пам'ять – пристрій, який служить для зберігання інформації (програм, вихідних даних, проміжних і кінцевих результатів обробки), що безпосередньо використовується в ході виконання програми в процесорі. В даний час обсяг ОП персональних комп'ютерів складає декілька сотень мегабайт. *Оперативна пам'ять* працює на частоті системної шини і вимагає 6-8 циклів синхронізації шини для звернення до неї.

Так, при частоті роботи системної шини 100 МГц (при цьому період дорівнює 10 нс) час звернення до *оперативної пам'яті* складе декілька десятків наносекунд.

Для заповнення пропусків між РП і ОП за обсягом і часу звернення в даний час використовується яка організована як більш швидкодіюча (і, отже, дорожча *кеш-пам'ять*), статична *оперативна пам'ять* із спеціальним механізмом запису та зчитування інформації і призначена для зберігання інформації, яка найбільш часто використовується при роботі програми.

Як правило, частина *кеш-пам'яті* розташовується безпосередньо на кристалі мікропроцесора (внутрішній *кеш*), а частина – поза ним (зовнішня *кеш-пам'ять*). *Кеш-пам'ять* програмно недоступна. Для звернення до неї використовуються апаратні засоби процесора і комп'ютера.

Зовнішня пам'ять організується, як правило, на магнітних та оптичних дисках, магнітних стрічках. *Ємність* дискової пам'яті досягає десятків гігабайт при часі звернення менше 1 мкс. Магнітні стрічки внаслідок своїх малої *швидкодії* і великої *ємності* використовуються в даний час в основному тільки як пристрої резервного копіювання даних, звернення до яких відбувається рідко, а може бути і ніколи. Час звернення для них може досягати декількох десятків секунд.

Слід зазначити, що електронна обчислювальна техніка розвивається надзвичайно швидкими темпами.

Так, згідно емпіричному "закону Мура", продуктивність комп'ютера подвоюється приблизно кожні 18 місяців. Тому всі наведені в даному посібнику кількісні характеристики служать здебільшого тільки для відображення основних співвідношень і тенденцій у розвитку тих чи інших компонентів і пристроїв комп'ютерів.

Побудова ЗП із заданою організацією

У сучасних ЕОМ мінімальною одиницею пам'яті, що адресується, як правило є 1 байт. У зв'язку з цим обмін з пам'яттю організується блоками, кратними цій величині: байтами, словами, подвійними словами, вчетверенними словами, в залежності від виконуваної процесором команди і розрядності зовнішньої шини даних. Такий обмін проходить під управлінням спеціальних сигналів, що надходять з системної шини. Перетворення інформації з формату її подання на шині даних у формат, який враховує організацію конкретних схем пам'яті, здійснюється спеціальними інтерфейсними схемами. Великі інтегральні схеми (ВІС), на яких будуються модулі пам'яті, є виробами електронної промисловості і можуть мати різну організацію. Розробники засобів обчислювальної техніки повинні враховувати наявну у них номенклатуру ВІС пам'яті, щоб побудувати *пристрій* необхідної *ємності* і організації. Для цієї мети може проводитися об'єднання декількох ВІС або з метою збільшення кількості слів у модулі пам'яті, або для нарощування розрядності кожного слова, або з тією і іншою метою одночасно.

Розглянемо варіанти побудови блоку пам'яті необхідної організації при наявності заданих ВІС пам'яті.

Побудувати *ОЗП* з організацією $8K \times 8$ розрядів на ВІС з організацією $1K \times 8$ розрядів (рис. 11.3).

В даному випадку потрібно побудувати модуль пам'яті, що має більшу кількість слів, ніж у складових його ВІС. Модуль пам'яті буде складатися з восьми ВІС. Для звернення до модуля пам'яті використовується 13-розрядна адреса

(A12 A0), що надходить по шині адреси (ША). Три старших розряди (A12-A10) визначають ту схему, яка в даний момент включається в роботу, а кожна комірка всередині будь-якої ВІС визначається 10-ма молодшими розрядами адреси (A9-A0) (рис. 11.4).

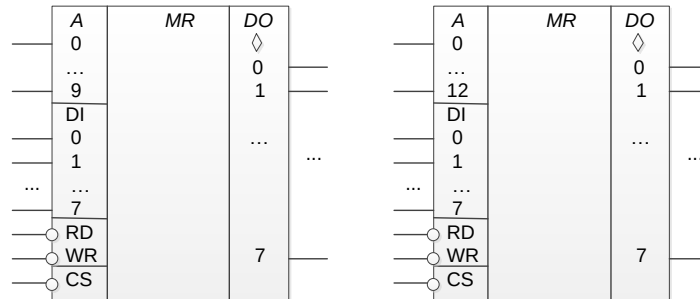


Рис. 11.3. Умовно-графічні позначення запам'ятовуючих пристроїв з різною організацією: а - 1К * 8 розрядів; б - 8К * 8 розрядів

Розряди адреси		Обрана ВІС
12 11 10 вибір ВІС	9 ... 0 вибір комірки в ВІС	
1 1 1	1...1 ... 0...0	ВІС 7
1 1 0	1...1 ... 0...0	ВІС 6
...		
0 0 1	1...1 ... 0...0	ВІС 1
0 0 0	1...1 ... 0...0	ВІС 0

Рис. 11.4. Організація модуля пам'яті

При одиничному значенні сигналу на вході вибору кристала ВІС ($CS = 1$) вихідні розряди даних знаходяться в третьому стані, тобто, як би відключені від шини ($DO = Z$). Таким чином, при будь-якому значенні коду на шині адреси завжди в роботі знаходиться одна і тільки одна з восьми ВІС (рис. 11.5).

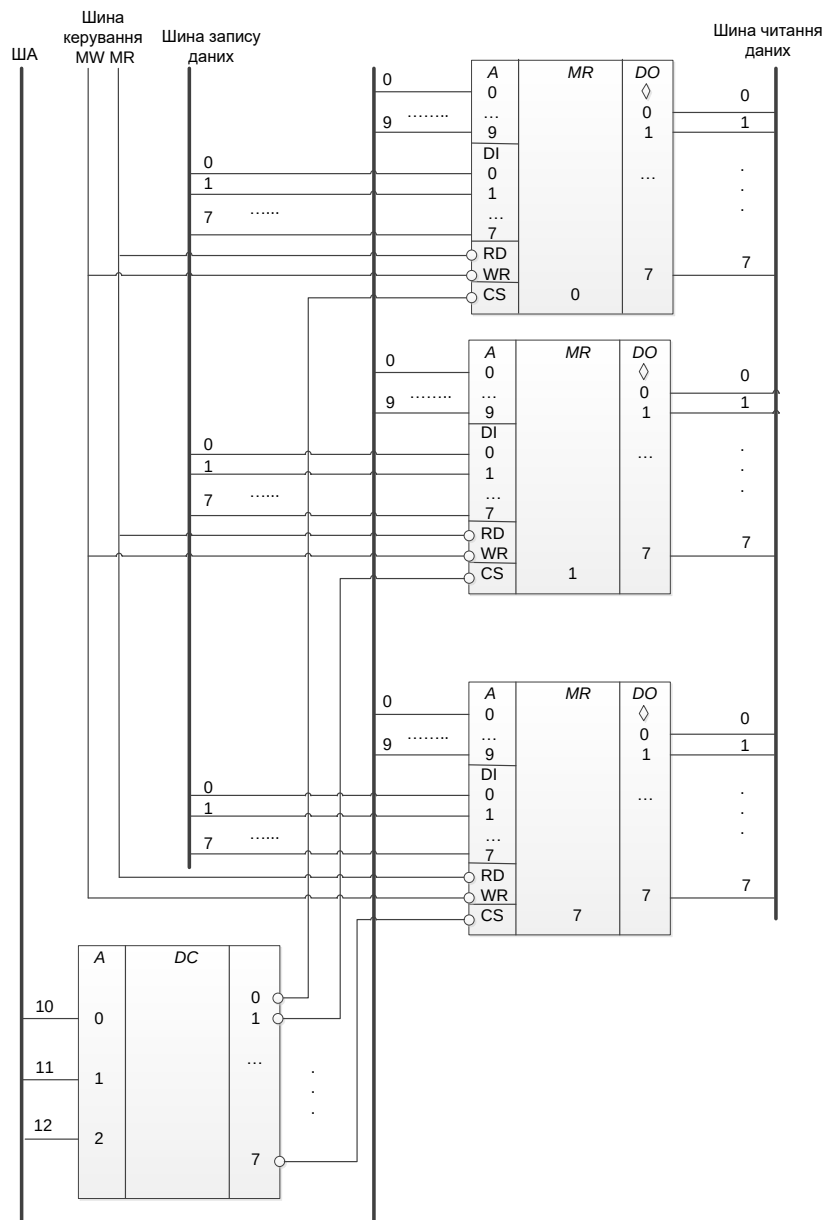


Рис. 11.5. Запам'ятовуючий пристрій об'ємом 8К * 8 розрядів на ВІС з організацією 1К * 8 розрядів

У реальних мікросхемах шини даних запису і читання (DI і DO) зазвичай представляють собою загальну двонаправлену шину.

Сигнали на шині управління означають: MW - сигнал запису в пам'ять, MR - сигнал читання з пам'яті. Побудувати ОЗП з організацією 1К * 8 розрядів на ВІС з організацією 1К * 1 розряд (рис. 11.6).

В даному випадку потрібно збільшити розрядність слова пам'яті. Так як всі розряди одного слова повинні записуватися і зчитуватися одночасно, то усі

ВІС повинні працювати паралельно. Модуль пам'яті буде складатися з восьми ВІС (рис. 11.7).

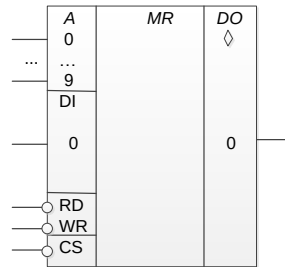


Рис. 11.6. Умовно-графічне позначення ВІС з організацією 1К*1 розряд

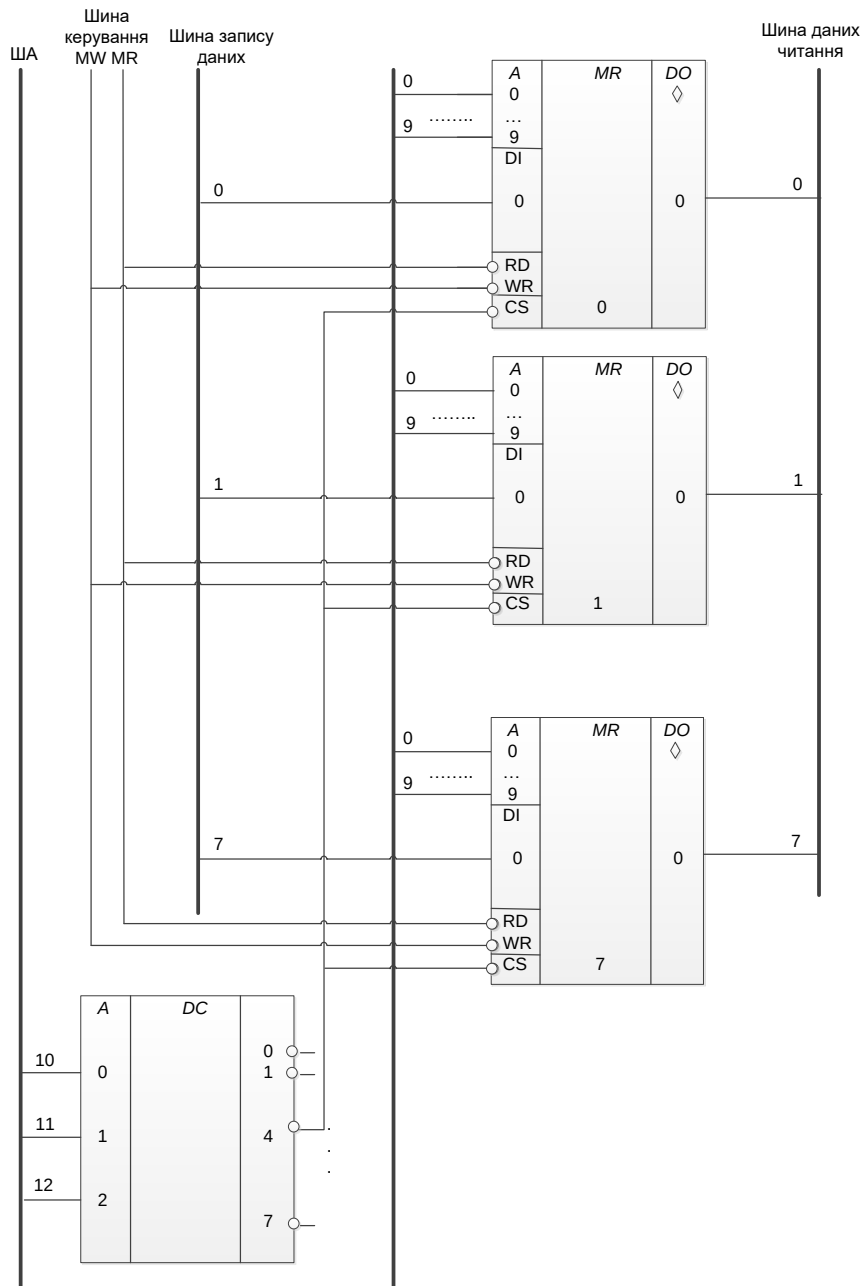


Рис. 11.7. Запам'ятовуючий пристрій об'ємом 1К * 8 розрядів на ВІС з організацією 1К * 1 розряд

Якщо розроблюваний блок є частиною модуля пам'яті, що має обсяг більше, ніж 1К слів (наприклад, 8К), то необхідний спеціальний дешифратор, який буде дешифрувати старші розряди адреси аналогічно тому, як показано на рис. 11.5 та включати в роботу даний блок.

11.2. Система управління пам'яттю

Організація розподілу пам'яті в ЕОМ

Запам'ятовуючі пристрої є однією з основних частин будь-якого комп'ютера. Їхня робота, як зазначалося раніше, будується за ієрархічним принципом. Від того, наскільки раціонально організовано використання пам'яті на кожному з рівнів ієрархії та взаємодія між ЗП різних рівнів, багато в чому залежить ефективність роботи ЕОМ.

Ключову роль в цій ієрархії грає оперативна пам'ять. Саме в ній зберігаються програми під час їх виконання, саме звідси завантажуються в регістри мікропроцесора вихідні дані для обробки. Сюди ж, як правило, передаються і остаточні результати роботи програм. Тому раціональне використання ОЗП протягом усього часу роботи ЕОМ надзвичайно важливо.

В оперативній пам'яті мультипрограмних ЕОМ зазвичай постійно зберігається ядро операційної системи. Програми ядра ОС в процесі роботи ЕОМ виконуються часто, час їх виконання невеликий. Інші частини операційної системи, як правило, знаходяться у зовнішній пам'яті, і в разі потреби необхідні модулі завантажуються в оперативну пам'ять, займаючи її частину. У частині ВП зберігається кілька програм, які виконуються в мультипрограмному режимі, і використовуються ними дані.

Розподіл пам'яті передбачає задоволення потреб, як користувачів, так і системних засобів. Ці вимоги в більшій частині суперечливі.

Системні цілі полягають, перш за все, в збільшенні ступеню використання оперативної пам'яті при паралельному розвитку декількох процесів в мульт

ти програмному режимі, а також у реалізації захисту інформації при розвитку цих процесів, забезпечення взаємодії між процесами і т. д.

Вимоги користувачів до пам'яті дуже різноманітні: швидке виконання коротких програм, виділення оперативної пам'яті в розмірах, що перевищують фізично існуючу, легкість і простота взаємодії з іншими. Внаслідок цього розподіл пам'яті завжди носить компромісний характер.

Система управління пам'яттю виконує такі основні функції:

- облік стану вільних і вже розподілених областей пам'яті і модернізація цієї інформації кожного разу, коли у розподілі пам'яті відбуваються зміни;
- розподіл пам'яті для виконання завдань (визначення, якій задачі, коли і в якій кількості виділити оперативну пам'ять);
- безпосереднє виділення завданню оперативної пам'яті; якщо вільні області оперативної пам'яті відсутні, то попереднє їх звільнення шляхом збереження інформації у зовнішній пам'яті.

Уся доступна множина адрес елементів зберігання, впорядкована за будь-якою ознакою, називається адресним простором пам'яті. Фізично адресний простір організований просто як одновимірний масив комірок, кожній з яких привласнюється свій номер, що називають фізичною адресою.

У загальному випадку, під адресою розуміють якийсь ідентифікатор, що однозначно визначає розташування елемента зберігання серед інших елементів у складі середовища зберігання.

Для адресації даних в фізичному адресному просторі програми використовують логічну адресацію. Процесор автоматично транслює *логічні адреси* у фізичні, що видаються на адресну шину і сприймаються схемами управління (контролерами) пам'яті.

Існують дві стратегії розподілу оперативної пам'яті, як і будь-якого ресурсу: статичний і динамічний розподіл.

При *статичному розподілі* вся необхідна оперативна пам'ять виділяється процесу в момент його породження. При цьому пам'ять виділяється єдиним блоком необхідної довжини, початок якого визначається базовою адресою.

Програма пишеться в адресах щодо початку блоку, а фізична адреса команди або операнда при виконанні програми формується як сума базової адреси блоку і відносної адреси в блоці. Значення базової адреси встановлюється при завантаженні програми в оперативну пам'ять. Так як у різних програмах використовуються блоки різної довжини, то при такому підході виникає проблема фрагментації пам'яті, тобто виникають вільні ділянки пам'яті, які неможливо без попереднього перетворення використовувати для обчислювального процесу.

Проілюструємо це простим прикладом.

Нехай ОП має об'єм 10 Мбайт, а для виконання програм А, В, С, D необхідні наступні обсяги пам'яті: – 2 Мбайт, В – 1 Мбайт, С – 4 Мбайт, D – 4 Мбайт. Початковий розподіл пам'яті і розподіл пам'яті після виконання деяких програм наведено на рис. 11.8.

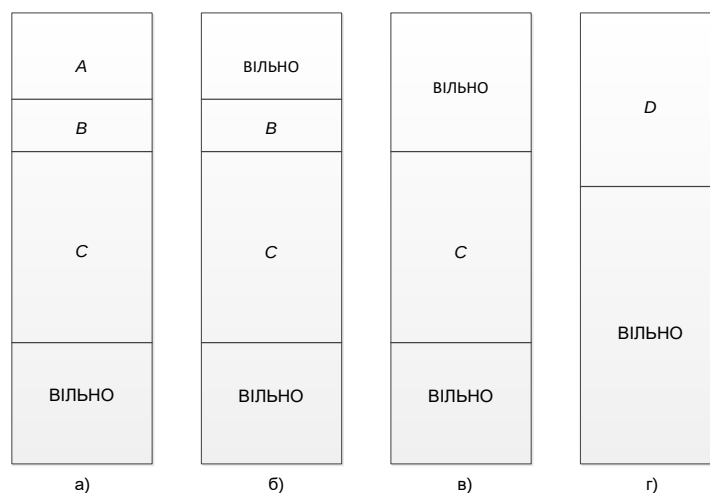


Рис. 11.8. Статичний розподіл пам'яті: а) початковий розподіл; б) після завершення програми А; в) після завершення програми В; г) після завершення програми С

На рисунку видно, що програма D при *статичному розподілі пам'яті* може бути завантажена в оперативну пам'ять лише після завершення виконання всіх попередніх програм, хоча необхідний для неї обсяг пам'яті існував вже після завершення роботи програми А. У той же час для поліпшення показників роботи мультипрограмної ЕОМ потрібно, щоб в оперативній пам'яті постійно знаходилася більша кількість програм, готових до виконання.

Дану проблему можна частково вирішити перерозподілом пам'яті після завершення виконання кожної програми з метою формування єдиної вільної ділянки, яка може бути виділена новій програмі, яка надходить на обробку (*дефрагментація пам'яті*).

Однак це вимагає трудомісткої роботи системних засобів і практично не використовується.

Сучасні системи розподілу пам'яті спираються на дві концепції: динамічного використання ресурсів та віртуалізації.

Сучасні системи розподілу пам'яті спираються на дві концепції: динамічного використання ресурсів та віртуалізації.

При *динамічному розподілі пам'яті* кожній програмі в початковий момент виділяється лише частина від усієї необхідної їй пам'яті, а інша частина виділяється по мірі виникнення реальної потреби в ній. Такий підхід базується на наступних передумовах.

По-перше, при кожному конкретному виконанні в залежності від вихідних даних деякі частини програми (до 25% її довжини) взагалі не використовуються. Слід прагнути до того, щоб ці фрагменти коду не завантажувалися в ОП.

По-друге, виконання програми характеризується так званим принципом локальності посилянь. Він передбачає, що при виконанні програми протягом деякого відносно малого інтервалу часу відбувається звернення до пам'яті в межах обмеженого діапазону адрес (як за кодом програми, так і за даними). Отже, протягом цього часу немає необхідності зберігати в ОП інші блоки програми.

При цьому системні засоби повинні відслідковувати виникнення вимог на звернення до тих частин програми, які в даний момент відсутні в ОЗП, виділяти цій програмі необхідний блок пам'яті і розміщувати там з зовнішнього ЗП необхідну частину програми. Для цього може знадобитися попереднє переміщення деяких блоків інформації з ОЗП в зовнішню пам'ять. Дані переміщення повинні бути приховані від користувача і в найменшій мірі уповільнювати роботу його програми.

Переміщення блоків інформації з ОЗУ в зовнішню пам'ять з метою звільнення місця для нової інформації відбувається зазвичай по одному з наступних алгоритмів:

- LRU (least recently used) – найбільш давно не використовується;
- FIFO – найдавніший по перебуванню в ОЗП;
- Random – випадковим чином.

Динамічний розподіл пам'яті тісно переплітається з поняттям *віртуальної пам'яті*.

Принцип *віртуальної пам'яті* припускає, що користувач при підготовці своєї програми має справу не з фізичною ОП, яка дійсно працює в складі комп'ютера і має деяку фіксовану ємність, а з віртуальною (що здається) однорівневою пам'яттю, ємність якої дорівнює всьому адресному простору, що визначається розміром адресної шини ($L_{ша}$) комп'ютера:

$$V_{virt} \gg V_{физ}, V_{virt} = 2^{L_{ша}}.$$

Для персональної ЕОМ на основі 32-розрядних мікропроцесорів:

$$V_{virt} = 2^{32} = 4 \text{ Гбайт}$$

При цьому, звісно, в ЕОМ повинен бути забезпечений достатній обсяг зовнішньої пам'яті для зберігання всіх програм, що оброблюються на комп'ютері.

Програміст має в своєму розпорядженні адресний простір, обмежений лише розрядністю адресної шини, незалежно від загального обсягу оперативної пам'яті комп'ютера і обсягів пам'яті, що використовуються іншими програмами, паралельно оброблюваними в мультипрограмній ЕОМ.

Віртуальна пам'ять, забезпечуючи програмісту можливість звертатися до дуже великого обсягу безперервного адресного простору, наданого в його монопольне розпорядження, має звичайні властивості: побайтова адресація, час доступу, порівнянне з часом доступу до оперативної пам'яті.

На всіх етапах підготовки програм, включаючи завантаження в оперативну пам'ять, програма представляється у віртуальних адресах, і лише при виконанні машинної команди віртуальні адреси перетворюються на фізичні. Для

кожної програми, що виконується в мультипрограмному режимі, створюється своя *віртуальна пам'ять*. Кожна програма використовує одні й ті ж віртуальні адреси від нульової до максимально великої в даній архітектурі.

Для перетворення віртуальних адрес у фізичні, фізична і *віртуальна пам'ять* розбивається на блоки фіксованої довжини, що називають сторінками. Обсяги віртуальної і *фізичної сторінок* збігаються. Сторінки віртуальної і фізичної пам'яті нумеруються.

Віртуальна (логічна) адреса в цьому випадку представляє собою номер віртуальної сторінки і зсув усередині цієї сторінки. У свою чергу, фізична адреса - це номер *фізичної сторінки* і зсув в ній.

Спочатку в ОП завантажується перша сторінка програми і їй передається керування. Коли в ході виконання програми відбувається звернення за межі завантаженої сторінки, операційна система перериває виконання даної програми, завантажує потрібну сторінку в ОП, після чого передає управління перерваної програми.

Правила переведення номерів *віртуальних сторінок* в номери *фізичних сторінок* зазвичай задаються у вигляді таблиці *сторінкового перетворення*. Такі таблиці формуються системою управління пам'яттю і модифікуються кожен раз при перерозподілі пам'яті.

Переведення віртуальних адрес у фізичні наведено на рис. 11.9.

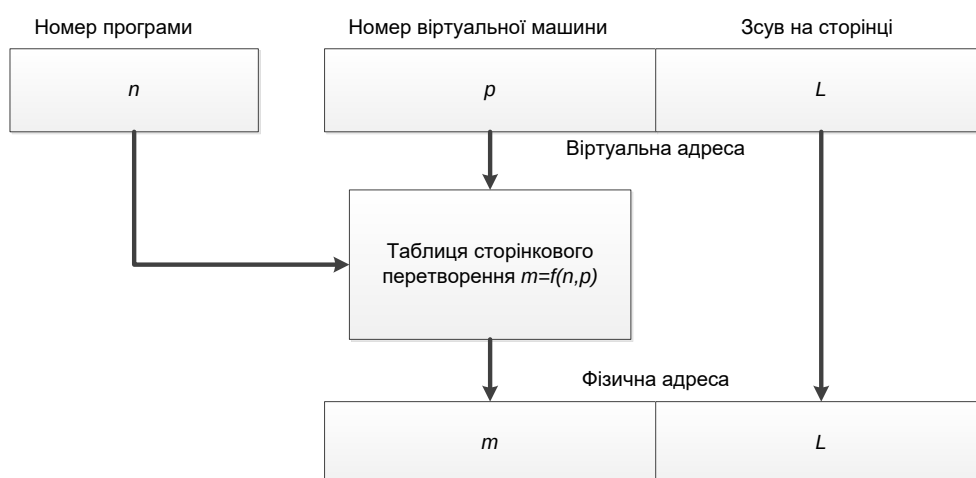


Рис. 11.9. Переведення віртуальної адреси у фізичну

Розглянемо приклад перетворення адреси *віртуальної сторінки* на адресу *фізичної сторінки*. Нехай комп'ютер має оперативну пам'ять $V_{озу} = 3$ і адресний простір, що передбачає розбиття на сторінки обсягом $V_{стр} = 1$.

Кожна програма, в свою чергу, розбивається на *віртуальні сторінки* того ж об'єму. Нехай коефіцієнт мультипрограмування даної ЕОМ дорівнює чотирьом, тобто на комп'ютері можуть одночасно виконуватися до чотирьох програм.

Перемикання між програмами відбувається через $t_k = 1$. Час виконання кожної сторінки будь-якої програми становить $t = 2t_k$. Вважаємо, що сторінки програм завантажуються в оперативну пам'ять по мірі їх необхідності і, по можливості, у вільні області ОЗУ. Якщо вся пам'ять зайнята, то нова сторінка заміщає ту, до якої найдовше не було звернень (механізм заміщення LRU). Нехай програми, що виконуються, мають наступну кількість сторінок: $V_A = 2, V_B = 1, V_C = 3, V_D = 2$. Тоді таблиця завантаження оперативної пам'яті і таблиці *сторінкового перетворення* для кожної програми будуть мати наступний вигляд (табл. 11.1):

У таблиці розподілу оперативної пам'яті виділені номери активних в даному такті сторінок.

У таблицях *сторінкового перетворення* прочерками відзначені ситуації, коли дана *віртуальна сторінка* відсутня в оперативній пам'яті. Якщо кожна сторінка має об'єм 1000 адресованих комірок, то, наприклад, в такті 9 звернення за віртуальною адресою 1100 програми А (*віртуальна сторінка* 1, зсув в сторінці дорівнює 100) призведе до звертання за фізичною адресою 2100 (*фізична сторінка* 2, зсув у *фізичній сторінці* такий же, як і у віртуальній – 100).

Таблиця *сторінкового перетворення* зберігається в ОП. У зв'язку з цим кожне звернення програми до пам'яті за командою або за операндом вимагає додаткового звернення до оперативної пам'яті для *сторінкового перетворення*, що істотно знижує продуктивність комп'ютера. Для зменшення впливу цього чинника використовуються різні підходи. Основний з них - метод, при якому

після першого перетворення номера *віртуальної сторінки* отриманий номер *фізичної сторінки* запам'ятовується у внутрішніх службових регістрах мікропроцесора. При черговому зверненні до пам'яті апаратними засобами мікропроцесора здійснюється перевірка того, чи було вже звернення до даної *віртуальної сторінки*. Якщо було, то номер відповідної їй *фізичної сторінки* вже знаходиться в мікропроцесорі. В іншому випадку перетворення виконується звичайним чином із зверненням до оперативної пам'яті. Так як програма може досить довго звертатися до адрес, що знаходяться в межах однієї сторінки, такий підхід суттєво скорочує час на *сторінкове перетворення*.

Таблиця 11.1

Таблиця завантаження оперативної пам'яті і сторінкового перетворення

Сторін-ка	Такти															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Динамічний розподіл оперативної пам'яті															
ОЗУ 0	A 0	A 0	A 0	D 0	D 0	D 0	C 0	C 0	C 0	C 1	C 1	C 1	C 1	C 1	C 1	C 1
1		B 0	B 0	B 0	A 0	A 0	A 0	D 0	D 0	D 0	D 1	D 1	D 1	D 1	D 1	D 1
2			C 0	C 0	C 0	B 0	B 0	B 0	A 1	A 1	A 1	A 1	A 1	A 1	C 2	C 2
	Таблиця станичного перетворення для програми А															
A 0	0	0	-	-	1	1	1	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	2	2	2	2	2	2	-
	Таблиця сторінкового перетворення для програми В															
B 0	-	1	1	1	-	2	2	2	-	-	-	-	-	-	-	-
	Таблиця сторінкового перетворення для програми С															
C 0	-	-	2	2	2	-	0	0	0	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2
	Таблиця сторінкового перетворення для програми D															
D 0	-	-	-	0	0	0	-	1	1	1	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	1	1	1	1	1	1

11.3. Система керування пам'яттю в персональній ЕОМ

У ЕОМ на основі 32-розрядного мікропроцесора при роботі в так званому захищеному режимі, що підтримує мультипрограмування і забезпечує адресацію операндів в максимально можливому для даної архітектури діапазоні до 2^{32} байт, *віртуальна пам'ять* організується на основі сегментно-сторінкового представлення пам'яті.

При цьому пам'ять розбивається на сегменти змінної довжини, що виділяються користувачеві під розміщення його програм і даних. Сегменти, в свою чергу, діляться на сторінки фіксованої довжини ($4K = 2^{12}$ байт), що використовуються системою керування пам'яттю для її віртуалізації [7].

Початок кожного сегмента встановлюється операційною системою через відповідний сегментний реєстр і приховано від користувача. Користувач пише свої програми в адресах щодо початку сегмента, вважаючи, що він має сегмент максимально можливої для даної архітектури довжини (2^{32} байт).

Апаратні засоби мікропроцесора спочатку проводять сегментне перетворення адреси, а потім - сторінкове.

Механізм формування фізичної адреси при сегментно-сторінковій організації пам'яті наведено на рис. 11.10.

Основою отримання фізичної адреси, що видається на адресну шину мікропроцесора, служить *логічна адреса*. Вона складається з двох частин: *селектора*, що є ідентифікатором сегмента, і зсувом в сегменті.

Зсув в сегменті (32 розряду) (ефективна адреса) обчислюється за режимом адресації операнда, що задається в команді, і є віртуальною адресою операнда. При зверненні до команди в якості зсуву виступає значення реєстра-показчика команд [12].

Селектор розміщується в сегментному реєстрі (див. рис. 11.10). Основною його частиною є номер (INDEX), по якому в одній із спеціальних таблиць дескрипторів можна знайти *дескриптор* (*описувач*) даного сегмента. Вид використовуваної таблиці визначається бітом TI (table indicator) *селектора*. Селек-

тор містить також дворозрядне поле RPL, що використовується при організації захисту пам'яті з привілеїв.

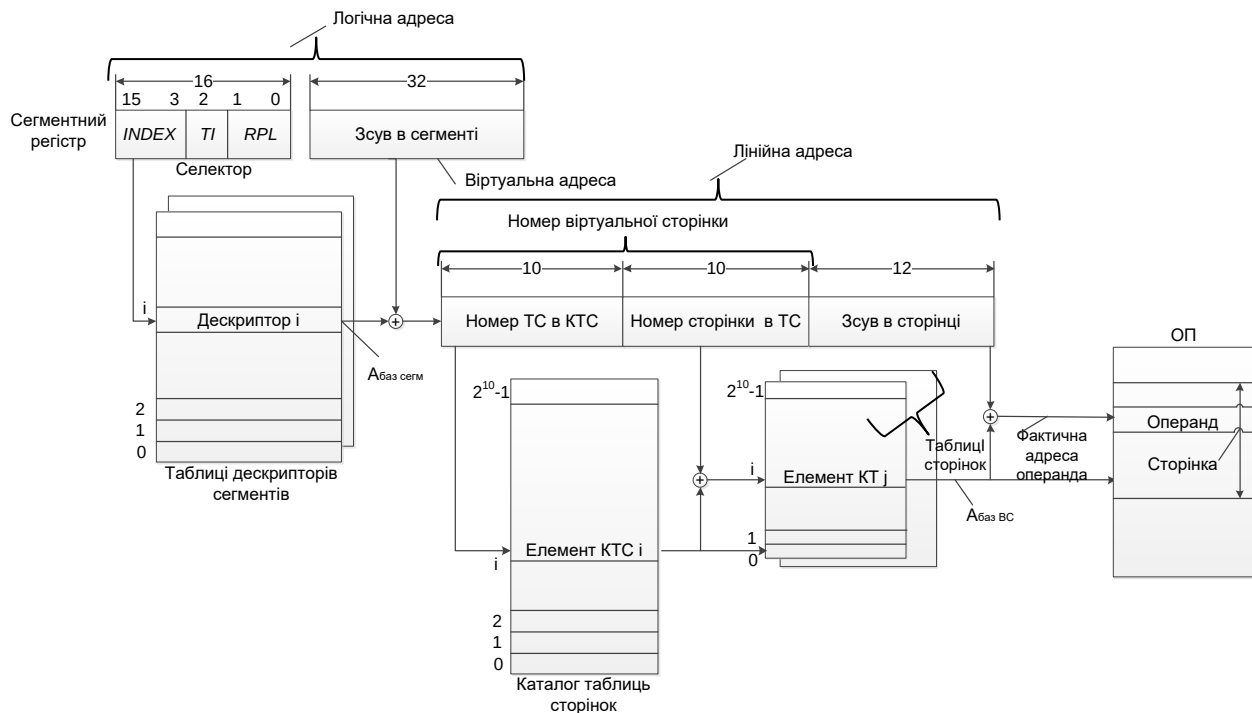


Рис. 11.10 Формування фізичної адреси при сегментно-сторінковій організації пам'яті в 32-розрядному мікропроцесорі

Дескриптор (рис. 11.11) містить відомості про сегмент. В одному з його полів міститься базова адреса сегмента. В інших полях записана додаткова інформація про сегмент: довжина, допустимий рівень прав доступу до даного сегменту з метою захисту інформації, знаходиться в ньому, тип сегмента (сегмент коду, сегмент даних, спеціальний системний сегмент і т.д.) і деякі інші атрибути.

Сума отриманої з *дескриптора* базової адреси сегмента і обчисленого зсуву дає лінійну адресу операнда, який при включеному механізмі *сторінкового перетворення* являє собою номер *віртуальної сторінки* (старші 20 розрядів) і зсув операнда в сторінці (молодші 12 розрядів лінійної адреси відповідно до обсягу сторінки в 4 Кбайт).

При перетворенні номера *віртуальної сторінки* в номер фізичної використовуються наступні системні об'єкти: *каталог таблиць сторінок (KTC)* і *таблиці сторінок (ТС)*. Структури цих таблиць подібні між собою (рис. 11.12).

Базова адреса сегменту (Абаз.сегм.)	Довжина сегменту	Права доступу	Атрибути
32	20	2	10

Рис. 11.11. Структура дескриптора сегмента

Базова адреса ТС (сторінки)	U/S	R/W	P	Атрибути
20	12			

Рис. 11.12. Елемент каталогу таблиць сторінок (таблиці сторінок) Перетворення проводиться в два етапи

Спочатку за розрядами A31-A22 лінійної адреси в *KTC* обирається потрібний елемент. *Каталог таблиць сторінок* завжди присутній в ОП і містить вказівки по розміщенню *таблиці сторінок*, що відносяться до того чи іншого процесу.

Елемент *KTC* містить:

- адресу початку *таблиці сторінок*;
- біт присутності (P) *таблиці сторінок* в оперативній пам'яті;
- біт дозволу читання/запису (R/W);
- біт захисту сторінки (користувач/супервізор (U/S)) і деякі інші атрибути.

ти.

Після отримання з обраного елемента *KTC* початкової адреси *таблиці сторінок* відбувається звернення до *ТС*.

У вибраній *таблиці сторінок* знаходиться елемент, номер якого визначається розрядами A21-A12 лінійної адреси.

Структура елемента *таблиці сторінок* аналогічна структурі елемента *КТС*. Елемент *ТЗ* у відповідному полі містить адресу початку необхідної *фізичної сторінки* та інші атрибути, аналогічні елементу *КТС*.

При $P = 0$ виникає переривання, необхідна сторінка підкачується в ОП, її адреса заноситься до відповідного елемента *ТС*, і команда виконується повторно.

Скорочення втрат часу при використанні сегментно-сторінкової організації пам'яті в персональній ЕОМ.

Перетворення *логічної адреси* у фізичну при сегментно-сторінковій організації пам'яті вимагає, як мінімум, трьох звернень до системних таблиць, розташованих в оперативній пам'яті (таблиці дескрипторів, *КТС* і *ТЗ*). Це може привести до істотного зниження продуктивності комп'ютера.

Механізм скорочення втрат часу на таке перетворення ґрунтується на тому факті, що зміна стану сегментних реєстрів здійснюється відносно рідко, наприклад, при перемиканні ЕОМ на нове завдання, а нове *сторінкове перетворення* потрібно лише при виході програми за межі завантаженої в оперативну пам'ять сторінки.

При сегментному перетворенні адреси після першого прочитання *дескриптора* з таблиці дескрипторів, розташованої в оперативній пам'яті (наприклад, після зміни стану сегментного реєстра при перемиканні на нове завдання), він запам'ятовується в програмно-недоступних ("тіньових") реєстрах мікропроцесора (рис. 11.13).

При наступних зверненнях до даного сегменту використовується *дескриптор* з "тіньового" реєстра без звернення до ОП.

Тому на його виклик потрібно мінімальний час. Так як стан сегментних реєстрів змінюється відносно рідко, то такий підхід призводить до значної економії часу при сегментному перетворенні адреси [10].

При *сторінковому перетворенні* номера *віртуальної сторінки* в номер *фізичної сторінки* використовується *кеш-буфер асоціативної трансляції* (TLB),

що містить фізичні адреси 32-х сторінок, що найбільш активно використовуються (рис. 11.14) і розташований безпосередньо в мікропроцесорі.

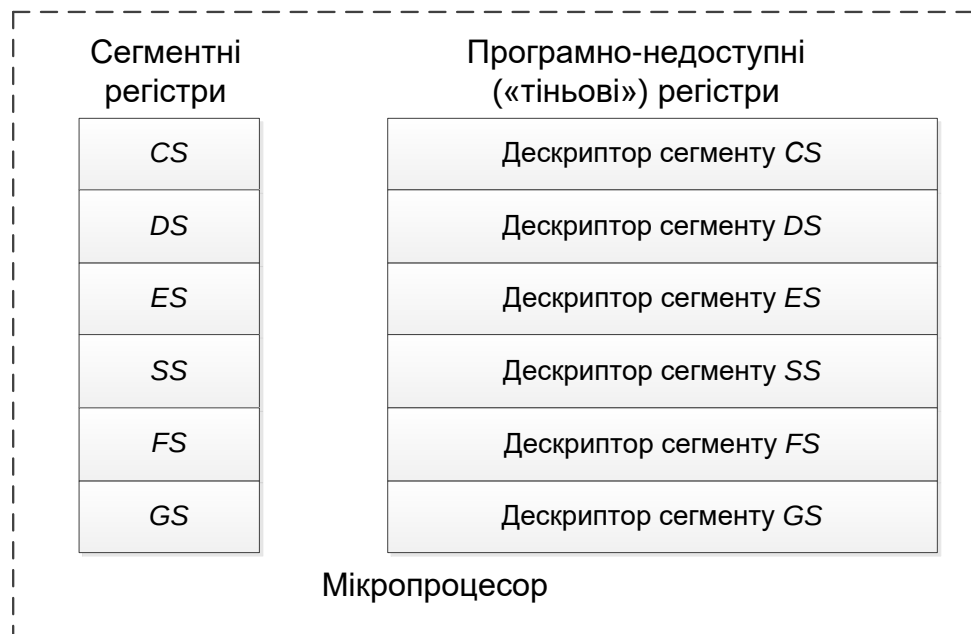


Рис. 11.13. Збереження дескрипторів сегментів в "тіньових" реєстрах мікропроцесора

	Тег (A31-A15)	Фізична адреса початку сторінки	Атрибути сторінки	
Блок 0				Рядок 0
				Рядок 1
				Рядок 2
				Рядок 3
Блок 1				Рядок 0
				Рядок 1
				Рядок 2
				Рядок 3
...	...	...	...	...
Блок 7				Рядок 0
				Рядок 1
				Рядок 2
				Рядок 3

Рис. 11.14. Формат буфера асоціативної трансляції адреси сторінки

Номером *віртуальної сторінки* є старші 20 розрядів лінійної адреси, отриманої при сегментному перетворенні (A31 – A12).

За молодшими розрядами (A14 – A12) цього номера вибирається блок в *буфері асоціативної трансляції*. Вміст поля тегів кожного з чотирьох рядків цього блоку асоціативним чином (одночасно) порівнюється з розрядами (A31 – A15) лінійної адреси.

Якщо значення для одного з рядків обраного блоку збіглися, значить, номер цієї *віртуальної сторінки* вже перетворювався в номер *фізичної сторінки* і результат цього перетворення знаходиться в знайденому рядку TLB.

Якщо порівняння не було успішним, то перетворення номера *віртуальної сторінки* в номер фізичної проходить звичайним чином через звернення до каталогу таблиць сторінок і до таблиці сторінок, а отримане значення заноситься в TLB.

При цьому в поле тегів заносяться старші 17 розрядів лінійної адреси цієї сторінки (A31 – A15).

Якщо немає вільного рядка в блоці, який визначається розрядами A14 – A12 лінійної адреси, то з блоку витісняється рядок, інформація в якій найдовше не використовувалася (механізм LRU).

11.4. Захист пам'яті в мультипрограмних ЕОМ

Організація захисту пам'яті в ЕОМ

При мультипрограмному режимі роботи ЕОМ в її пам'яті одночасно можуть перебувати кілька незалежних програм. Тому необхідні спеціальні заходи щодо запобігання або обмеження звернень однієї програми до областей пам'яті, що використовуються іншими програмами.

Програми можуть також містити помилки, які, якщо цьому не перешкодити, призводять до спотворення інформації, що належить іншим програмам.

Наслідки таких помилок особливо небезпечні, якщо руйнування зазнають програми операційної системи. Іншими словами, треба виключити вплив про-

грами користувача на роботу програм інших користувачів і програм операційної системи. Слід захищати і самі програми від помилок, які можливо в них знаходяться [12].

Таким чином, засоби захисту пам'яті повинні запобігати:

- недозволеній взаємодії користувачів один з одним;
- несанкціонованому доступу користувачів до даних;
- пошкодженню програм і даних через помилки в програмах;
- навмисним спробам зруйнувати цілісність системи;
- використанню інформації в пам'яті не відповідно до її функціонального

го призначення.

Щоб перешкодити руйнуванню одних програм іншими, досить захистити область пам'яті цієї програми від спроб запису в неї з боку інших програм, а в деяких випадках і своєї програми (*захист від запису*), при цьому допускається звернення інших програм до цієї області пам'яті для зчитування даних.

В інших випадках, наприклад, при обмеженнях на доступ до інформації, що зберігається в системі, необхідно забороняти іншим програмам будь-яке звернення до деякої області пам'яті, як на запис, так і на зчитування. Такий *захист від запису і зчитування* допомагає в налагодженні програми, при цьому здійснюється контроль кожного випадку звернення до області пам'яті своєї програми.

Для полегшення налагодження програм бажано виявляти і такі характерні помилки в програмах, як спроби використання даних замість команд або команд замість даних у власній програмі, хоча ці помилки можуть і не руйнувати інформацію (невідповідність функціонального використання інформації). Якщо порушується захист пам'яті, виконання програми припиняється і виробляється запит переривання стосовно порушення захисту пам'яті.

Захист від вторгнення програм в чужі області пам'яті може бути організований різними методами. Але при будь-якому підході реалізація захисту не повинна помітно знижувати продуктивність комп'ютера і вимагати занадто великих апаратурних витрат.

Методи захисту базуються на деяких класичних підходах, які отримали свій розвиток в архітектурі сучасних ЕОМ.

До таких методів можна віднести захист окремих осередків, *метод граничних регістрів, метод ключів захисту*.

Захист окремих елементів пам'яті організовується в ЕОМ, призначених для роботи в системах управління, де необхідно забезпечити можливість налагодження нових програм без порушення функціонування робочих програм, що знаходяться в пам'яті та керують технологічним процесом.

Це може бути досягнуто виділенням в кожному осередку пам'яті спеціального "розряду захисту".

Установка цього розряду в "1" забороняє робити запис в дану комірку, що забезпечує збереження робочих програм.

Недолік такого підходу – велика надмірність у кодуванні інформації через надмірно дрібний рівень об'єкта (комірки), що захищається [11].

У системах з мультипрограмною обробкою доцільно організовувати захист на рівні блоків пам'яті, що виділяються програмам, а не окремим коміркам.

Метод граничних регістрів (рис. 11.15) полягає у введенні двох граничних регістрів, що вказують верхню і нижню межі області пам'яті, куди програма має право доступу.

При кожному зверненні до пам'яті перевіряється, чи знаходиться адреса, що використовується у встановлених межах. При виході за межі звернення до пам'яті не відбувається, а формується запит переривання, що передає управління операційній системі.

Зміст граничних регістрів встановлюється операційною системою при завантаженні програми в пам'ять.

Модифікація цього *методу* полягає в тому, що один регістр використовується для вказівки адреси початку області, що захищається, а інший містить довжину цієї області. *Метод граничних регістрів*, володіє безсумнівною просто-

тою реалізації, має і певні недоліки. Основний з них це те, що цей *метод* підтримує роботу лише з безперервними областями пам'яті.

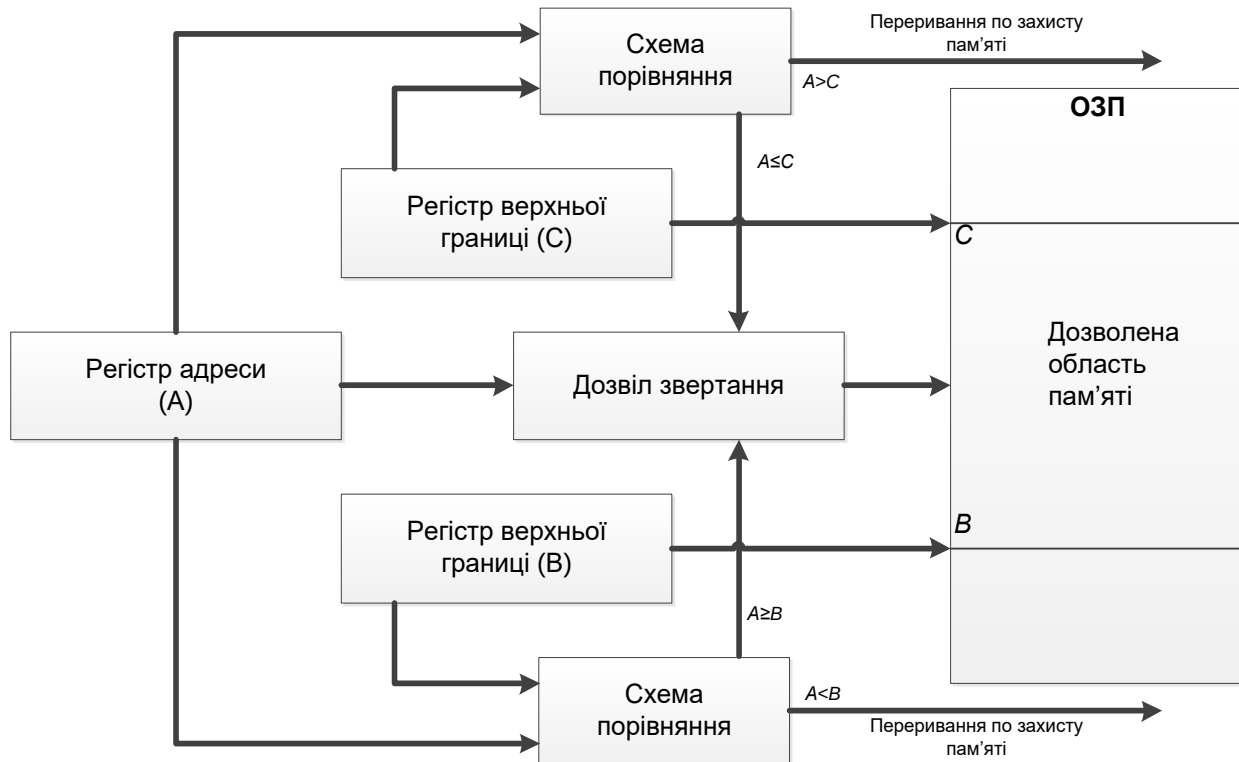


Рис. 11.15. Захист пам'яті методом граничних регістрів

Метод ключів захисту, на відміну від попереднього, дозволяє реалізувати доступ програми до областей пам'яті, організованим у вигляді окремих модулів, які не становлять собою єдиний масив. Пам'ять у логічному відношенні ділиться на однакові блоки, наприклад, сторінки. Кожному блоку пам'яті ставиться у відповідність код, що називають *ключем захисту пам'яті*, а кожній програмі, яка бере участь в мультипрограмній обробці, присвоюється код *ключа програми*.

Доступ програми до даного блоку пам'яті для читання і запису дозволено, якщо ключі збігаються (тобто даний блок пам'яті відноситься до даної програми) або один з них має код 0 (код 0 присвоюється програмами операційної системи і блокам пам'яті, до яких мають доступ усі програми: загальні дані, підпрограми, що спільно використовуються і т. п.). Коди ключів захисту блоків пам'яті і ключів програм встановлюються операційною системою.

У ключі захисту пам'яті передбачається додатковий розряд режиму захисту. Захист діє тільки при спробі запису в блок, якщо в цьому розряді стоїть 0, і при будь-якому зверненні до блоку, якщо стоїть 1. Коди ключів захисту пам'яті зберігаються в спеціальній пам'яті ключів захисту, більш швидкодіючою, ніж оперативна пам'ять. Функціонування цього механізму захисту пам'яті пояснюється схемою на рис. 11.16.

При зверненні до пам'яті група старших розрядів адреси ОЗП, що відповідає номеру блоку, до якого проводиться звернення, використовується як адреса для вибірки з пам'яті ключів захисту коду ключа захисту, привласненого операційною системою даного блоку.

Схема аналізу порівнює ключ захисту блоку пам'яті і ключ програми, що знаходиться в регістрі слова стану програми (ССП), і виробляє сигнал "Звернення дозволено" або сигнал "Переривання по захисту пам'яті".

При цьому враховуються значення режиму звернення до ОЗП (запис або зчитування), зазначених вище тригером режиму обігу ТrPO, і режиму захисту, встановленого в розряді режиму обігу (PPO) ключа захисту пам'яті.



Рис. 11.16. Захист пам'яті методом ключів захисту

Засоби захисту пам'яті в персональній ЕОМ

Захист пам'яті в персональній ЕОМ ділиться на *захист при управлінні пам'яттю* і *захист за привілеями*.

Засоби захисту при управлінні пам'яттю здійснюють перевірку: перевищення ефективною адресою довжини сегмента; прав доступу до сегмента на запис або тільки на читання; функціонального призначення сегмента.

Перший механізм базується на *методі граничних регістрів*. При цьому початкові адреси того чи іншого сегмента програми встановлюються операційною системою.

Для кожного сегмента фіксується його довжина. При спробі звернення за відносною адресою, що перевищує довжину сегмента, виробляється сигнал порушення захисту пам'яті. При перевірці функціонального призначення сегмента визначаються операції, які можна проводити над даними, що знаходяться у ньому. Так, сегмент, який представляє собою стек програми, повинен допускати звернення, як на запис, так і на читання інформації. До сегменту, який містить програму, можна звертатися тільки на виконання. Будь-яке звернення на запис або читання даних з цього сегменту буде сприйнято як помилкове. Тут спостерігається певний відхід від принципів Неймана в побудові ЕОМ, в яких стверджується, що будь-яка інформація, що знаходиться в ЗП, функціонально не поділяється на програму та дані, а її ідентифікація проводиться лише на стадії застосування даної інформації. Очевидно, що такий розвиток викликано багато в чому появою мультипрограмування і необхідністю більш уважного розгляду питань захисту інформації.

Захист за привілеями фіксує більш тонкі помилки, пов'язані, в основному, з розмежуванням прав доступу до тієї чи іншої інформації.

В якійсь мірі *захист за привілеями* можна порівняти з класичним *методом ключів захисту* пам'яті. Різним об'єктам (програмам, сегментам пам'яті, запитам на звернення до пам'яті та до зовнішніх пристроїв), які повинні бути розпізнані процесором, присвоюється ідентифікатор, що називають *рівнем привілеїв*. Процесор постійно контролює, чи має поточна програма достатні привілеї, щоб: виконувати деякі команди; виконувати команди введення-виведення на тому чи іншому зовнішньому пристрої; звертатися до даних інших програм; викликати інші програми.

На апаратному рівні в процесорі розрізняються 4 *рівні привілеїв*. Найбільш привілейованими є програми на *рівні 0*. Число програм, які можуть виконуватися на більш високому *рівні привілеїв*, зменшується від *рівня 3* до *рівня 0*. Програми *рівня 0* діють як ядро операційної системи.

Тому *рівні привілеїв* зазвичай зображаються у вигляді чотирьох *кілець захисту* (рис. 11.17).

Як правило, *розподіл програм по кільцях захисту* має наступний вигляд:

- *рівень 0* – ядро ОС, що забезпечує ініціалізацію роботи, управління доступом до пам'яті, захист і ряд інших життєво важливих функцій, порушення яких повністю виводить з ладу процесор;
- *рівень 1* – основна частина програм ОС (утиліти);
- *рівень 2* – службові програми ОС (драйвери, СУБД, спеціалізовані підсистеми програмування та ін);
- *рівень 3* – прикладні програми користувача.

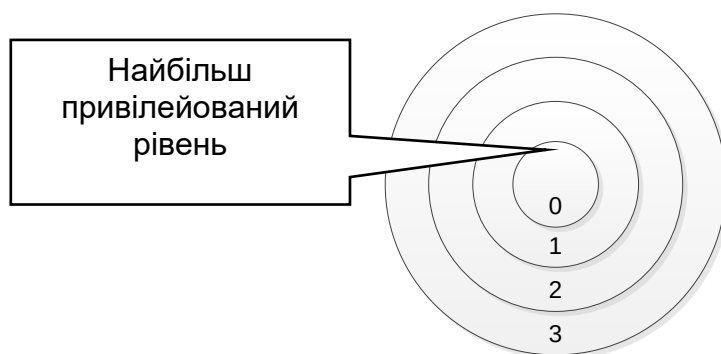


Рис. 11.17. Кільця захисту

Конкретна операційна система не обов'язково повинна підтримувати всі чотири *рівні привілеїв*. Так, ОС UNIX працює з двома *кільцями захисту*: супервізор (*рівень 0*) і користувач (*рівні 1,2,3*).

Операційна система OS/2 підтримує три *рівні*: код ОС працює в *кільці 0*, спеціальні процедури для звернення до пристроїв введення-виведення діють в *кільці 1*, а прикладні програми виконуються в *кільці 3*.

Просту незахищену систему можна цілком реалізувати в одному *кільці 0* (в інших *кільцях* це зробити неможливо, так як деякі команди доступні тільки на цьому *рівні*).

Рівень привілеїв сегмента визначається полем DPL *рівня привілеїв* його дескриптора. *Рівень привілеїв* запиту до сегмента визначається *рівнем привілеїв* RPL, закодованому в селекторі.

Звернення до сегменту дозволяється тільки тоді, коли *рівень привілеїв* сегмента не вище *рівня запиту*. Звернення до програм, які знаходяться на більш високому *рівні привілеїв* (наприклад, до утиліт операційної системи з програм користувача), можливо за допомогою спеціальних апаратних механізмів – шлюзів виклику [12].

При сторінковому перетворенні адреси застосовується простий дворівневий механізм захисту: користувач (*рівень 3*)/супервізор (*рівні 0,1,2*), що вказується в полі U/S відповідної таблиці сторінок.

При сегментно-сторінковому перетворенні адреси спочатку перевіряються привілеї при доступі до сегмента, а потім – при доступі до сторінки. Це дає можливість встановити більш високий ступінь захисту окремих сторінок сегмента.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке пам'ять ЕОМ?
2. Що таке «запам'ятовуючий пристрій»? Назвіть основні параметри та наведіть класифікацію ЗП.
3. Яким чином визначається швидкодія пам'яті?
4. Наведіть класифікацію запам'ятовуючих пристроїв.
5. Назвіть головну перевагу ієрархічної структури пам'яті.
6. Дайте визначення поняттю «оперативна пам'ять». На якому рівні ієрархічної організації пам'яті ЕОМ вона розташована?
7. Назвіть основні функції системи управління пам'яттю.
8. Які стратегії розподілу оперативної пам'яті вам відомі?

9. В чому полягає різниця між статичним та динамічним розподілами пам'яті?
10. У чому полягає принцип віртуальної пам'яті?
11. Що таке каталог таблиць сторінок? Які елементи КТС ви знаєте?
12. У чому полягає основне призначення дескрипторів?
13. За яких умов використовується кеш-буфер асоціативної трансляції?
14. У яких випадках застосовуються засоби захисту пам'яті?
15. Зазначте, на яких класичних підходах захисту базуються методи захисту?
16. Що являє собою захист окремих елементів пам'яті?
17. В чому суть метода граничних регістрів?
18. Що являє собою метод ключів захисту?
19. Що перевіряють засоби захисту при управлінні пам'яттю?
20. Визначте кількість рівнів, на які розподілено програми по кільцях захисту. Дайте коротку характеристику «кільця захисту».

12. ПРОЦЕСОРИ

12.1. Режими адресації й формати команд 16-розрядного процесора

Мікропроцесор Intel-8086 (Д01810ВМ80) має двоадресну систему команд. Її особливістю є відсутність команд, які використовують обидва операнда з оперативної пам'яті. Виняток становлять лише команди пересилання і порівняння ланцюжка байт або слів, які в даному посібнику розглядатися не будуть. Таким чином, у командах допустимі такі поєднання операндів: RR, RS, RI, SI. Тут R позначається операнд, що знаходиться в одному з регістрів реєстрової пам'яті мікропроцесора, S – операнд, що знаходиться в оперативній пам'яті, адреса якого формується по одному з допустимих способів адресації, I - безпосередній операнд, закодований в адресному полі самої команди. *Формат команди* багато в чому визначається способом адресації операнда, що знаходиться в оперативній пам'яті, довжиною використовуваного безпосереднього операнда, а також наявністю та довжиною зсуву, використовуваного при відносних режимах адресації.

Мікропроцесор має всі режими адресації, загальна схема яких була розглянута вище. Звісно, вони мають певні особливості, притаманні даному процесору.

Безпосередня адресація припускає, що операнд займає одне з полів команди і, отже, вибирається з оперативної пам'яті одночасно з нею. Залежно від форматів оброблюваних процесором даних безпосередній операнд може мати довжину 8 або 16 біт, що надалі будемо позначати data8 і data16 відповідно.

Механізми адресації операндів, що знаходяться в реєстровій пам'яті і в оперативній пам'яті, істотно розрізняються. До реєстрової пам'яті допускається лише *пряма регістрова адресація*. При цьому в команді вказується номер регістра, що містить операнд. 16-розрядний операнд може знаходитися в регістрах AX, BX, CX, DX, DI, SI, SP, BP, а 8-розрядний - у регістрах AL, AH, BL, BH, CL, CH, DL, DH.

Адресація оперативної пам'яті має свої особливості, пов'язані з її розбиттям на сегменти і використанням сегментної групи регістрів для вказівки початкової адреси сегмента. 16-розрядна адреса, що отримується в блоці формування адреси операнда на основі вказаного режиму адресації, називається *ефективною адресою* (ЕА). Іноді *ефективна адреса* позначається як ЕА (effective address). 20-розрядна адреса, яка отримується складанням *ефективної адреси* із більшого в 16 разів значення відповідного сегментного регістра, називається *фізичною адресою* (ФА) [7].

Саме *фізична адреса* передається з мікропроцесора по 20-ти адресним лініям, що входять до складу системної шини, в оперативну пам'ять і використовується при зверненні до її комірки на фізичному рівні. При отриманні *ефективної адреси* можуть використовуватися всі основні режими адресації, розглянуті вище, а також деякі їх комбінації.

Пряма адресація припускає, що *ефективна адреса* є частиною команди. Так як ЕА складається з 16 розрядів, то і відповідне поле команди повинно мати таку ж довжину [6].

При реєстровій *непрямій адресації* *ефективна адреса* операнда знаходиться в базовому регістрі ВХ або одному з індексних регістрів DІ або SІ:

$$EA = \left\{ \begin{array}{l} [BX] \\ [DI] \\ [SI] \end{array} \right\}$$

Позначення імені регістра у квадратних дужках вказує на вміст відповідного регістра. Фігурні дужки – символ вибору однієї з кількох можливих альтернатив.

При реєстровій *відносній адресації* *ефективна адреса* дорівнює сумі вмісту базового або індексного регістру і зміщення:

$$EA = \left\{ \begin{array}{l} [BX] \\ [BP] \\ [DI] \\ [SI] \end{array} \right\} + \left\{ \begin{array}{l} disp8 \\ disp16 \end{array} \right\}$$

Позначення тут і далі вказують на disp 8- або disp 16-розрядне зміщення відповідно.

Ефективна адреса при базово-індексній адресації дорівнює сумі вмісту базового і індексного реєстрів, що визначаються командою:

$$EA = \left\{ \begin{matrix} [BX] \\ [BP] \end{matrix} \right\} + \left\{ \begin{matrix} [DI] \\ [SI] \end{matrix} \right\}$$

Найбільш складний механізм відносної базово-індексної адресації. Ефективна адреса в цьому випадку дорівнює сумі 8 - або 16-розрядного зсуву і базово-індексної адреси:

$$EA = \left\{ \begin{matrix} [BX] \\ [BP] \end{matrix} \right\} + \left\{ \begin{matrix} [DI] \\ [SI] \end{matrix} \right\} + \left\{ \begin{matrix} \text{disp8} \\ \text{disp16} \end{matrix} \right\}$$

Формати двооперандних команд наведено на рис. 12.1. Пунктиром показані поля, які в залежності від режиму адресації можуть відображатися в команді.



Рис. 12.1. Формати двооперандних команд мікропроцесора I8086

Поле КІП містить код виконуваної операції. Ознака вказує на довжину операндів. При $w=1$ операція проводиться над словами, а при $w=0$ - над байтами. Ознака d вказує положення приймача результату. Ознака $d=1$, якщо результат записується на місце операнда, закодованого в поле reg , і $d=0$, якщо результат записується за адресою, закодованою полями $(\text{md}, \text{r/m})$.

Другий байт команди, що називають *постбайтом*, визначає операнди, що беруть участь в операції. Поле reg вказує реєстр реєстрової пам'яті згідно табл. 12.1

Регістр реєстрової пам'яті

<i>reg</i>	Регістр	
	<i>w</i> = 1	<i>w</i> = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Поля *md* і *r/m* задають режим адресації іншого операнда згідно табл. 12.2.

У цій таблиці крім визначення режиму адресації оперативної пам'яті вказаний також сегментний регістр, який використовується за умовчанням для отримання *фізичної адреси*.

Використання іншого сегментного регістра можливо введенням спеціального префікса (додаткового байта, який записується перед командою).

У командах, які використовують безпосередній операнд, ознаку *s* разом з ознакою *w* визначає розрядність безпосереднього операнда, що записується в команді, і розрядність виконуваної операції згідно табл. 12.3.

Зміна природного порядку виконання команд програми здійснюється за допомогою команд передач і управління.

До них відносяться команди переходів, *циклів*, виклику підпрограми і повернення з неї, а також деякі інші. Ми розглянемо лише перші дві групи команд.

Класифікація команд переходів у персональній ЕОМ наведено на рис. 12.2 [12].

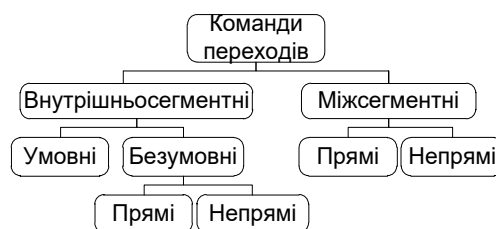


Рис. 12.2. Класифікація команд переходів IBM PC

Таблиця. 12.2.

Класифікація команд переходів у персональній EOM

r / m	md				
	00	01	10	11	
				$w = 1$	$w = 0$
000	$(BX) + (SI)$ (DS)	$(BX) + (SI) + disp8$ (DS)	$(BX) + (SI) + disp16$ (DS)	AX	AL
001	$(BX) + (DI)$ (DS)	$(BX) + (DI) + disp8$ (DS)	$(BX) + (DI) + disp16$ (DS)	CX	CL
010	$(BP) + (SI)$ (SS)	$(BP) + (SI) + disp8$ (SS)	$(BP) + (SI) + disp16$ (SS)	DX	DL
011	$(BP) + (DI)$ (SS)	$(BP) + (DI) + disp8$ (SS)	$(BP) + (DI) + disp16$ (SS)	BX	BL
100	(SI) (DS)	$(SI) + disp8$ (DS)	$(SI) + disp16$ (DS)	SP	AH
101	(DI) (DS)	$(DI) + disp8$ (DS)	$(DI) + disp16$ (DS)	BP	CH
110	$disp16$ (DS)	$(BP) + disp8$ (SS)	$(BP) + disp16$ (SS)	SI	DH
111	(BX) (DS)	$(BX) + disp8$ (DS)	$(BX) + disp16$ (DS)	DI	BH

Таблиця. 12.3.

Команди, які використовують безпосередній операнд

W	S	Операція	Безпосередній операнд
0	0	8-розрядна	8-розрядний
0	1	не використовується	
1	0	16-розрядна	16-розрядний
1	1		8-розрядний, розширюваний знайомий до 16-ти розрядів при виконанні операції

Фізична адреса виконуваної команди визначається вмістом покажчика команд IP і сегментного регістра команд CS. Команди, що змінюють значення обох цих регістрів, називаються командами *міжсегментних переходів*, а ті ко-

манди, що змінюють тільки значення ІР-командами *внутрішньосегментних переходів*.

Команди *безумовних переходів* роблять модифікацію регістру ІР або регістрів ІР і СS без попереднього аналізу яких-небудь умов.

Існує п'ять команд *безумовних переходів* [15]. Усі вони мають однакову мнемоніку JMP і містять один операнд. Конкретний *формат команди* визначається відповідним префіксом і наведений у загальній таблиці машинного представлення команд.

Примітка: у стовпці "Схема операції" АС означає регістр-акумулятор, у якості якого використовується регістр АХ при $w = 1$ і регістр АL при $w = 0$.

При *безумовному* прямому внутрішньосегментному переході нове значення покажчика команд ІР дорівнює сумі зсуву, закодованого у відповідному полі команди, і потокового значення ІР, у якості якого використовується адреса команди, записаної слідом за командою переходу. Команди прямих міжсегментних переходів містять у собі крім нового значення ІР і нове значення сегментного регістра СS.

Команди непрямих переходів (внутрішньосегментних і міжсегментних) передають управління на команду, адреса якої визначається вмістом регістра або коміркою оперативної пам'яті, на які вказує закодований у команді переходу *постбайт*.

Команди *умовних переходів* є тільки внутрішньосегментними. За своїм *форматом* і способом формування нового значення ІР вони повністю аналогічні команді внутрішньосегментного прямого переходу з 8-розрядним зсувом.

Відмінність їх полягає в тому, що в командах *умовного переходу* механізм формування нового значення ІР включається лише при виконанні певних умов, а саме, при певному стані регістра прапорів.

При невиконанні умови, що перевіряється, в ІР, залишається його поточне значення, тобто адреса команди, наступної за командою *умовного переходу*.

Команди *циклів* ідентичні за форматом і дуже близькі за діями, що виконуються командами умовних переходів.

Однак у порівнянні з останніми вони мають ряд особливостей, що дозволяють ефективно використовувати їх при програмуванні циклічних ділянок алгоритмів.

Один з найбільш поширених видів циклічної ділянки програми наведено на рис. 12.3.



Рис. 12.3. Структура рахункового циклу з пост перевіркою

Команди циклів призначені для спрощення дій декременту (зменшення на 1) лічильника циклу, перевірки умови виходу з циклу і переходу.

Деякі *команди циклу* реалізують вихід з циклу не тільки за значенням лічильника, але і при виконанні деяких інших умов.

Опис *команд циклу* зведено в табл. 12.4. За винятком команди JCXZ, яка не змінює значення регістра CX, при виконанні *команд циклів* проводяться наступні дії: $CX = (CX) - 1$.

Потім, якщо умова, що перевіряється, виконана, то $IP = (IP) + disp8$ з розширенням зсуву знаком до 16 розрядів, в іншому випадку IP не змінюється, і програма продовжує виконання в звичайному порядку [2].

Табл.12.4.

Команди циклів

Назва	Мнемоніка	Альтернативна мнемоніка	КІП	Умова, що перевіряється
Зациклити	LOOP		11100010	$(CX) = 0$
Зациклити поки нуль (рівно)	LOOPZ	LOOPE	11100001	$(ZF = 1) \& ((CX) = 0)$
Зациклити поки не нуль (нерівно)	LOOPNZ	LOOPNE	11100000	$(ZF = 0) \& ((CX) = 0)$
Перейти по (CX)	JCXZ		11100011	$(CX) = 0$

12.2 Кодування команд

З метою кращого розуміння архітектури ЕОМ розглянемо машинне представлення команд різних форматів, що використовують різні режими адресації операндів [16].

Розглянемо це на прикладі операції додавання. Так як у системі команд ЕОМ, що базуються на мікропроцесорах фірми Intel, результат операції записується на місце першого операнда, то дана операція буде мати вигляд:

$$a = a + b$$

Для наочного представлення команди будемо користуватися її *символічним записом*, наближеним до запису на мові Асемблер. Загальний формат асемблерної команди має наступний вигляд:

[Мітка:] Мнемоніка_операції Операнд, Операнд [; Коментар]

Мітка – це ідентифікатор, що присвоюється адресі першого байта команди. Наявність мітки в команді необов'язкова. При відсутності мітки двокрапка не повинна бути.

У всіх командах необхідна наявність мнемоніки, що позначає виконувану команду.

Наявність і кількість (один або два) операндів залежить від команди. У випадку двох операндів вони розділяються комою, при цьому першим зазначається операнд-приймач, а іншим – операнд-джерело. Приклади позначення операндів при різних режимах адресації будуть розглянуті нижче.

Поле коментарю призначено для пояснення програми і може містити будь-які комбінації символів. При відсутності коментаря крапка з комою може не ставитися.

Кодування лінійних команд

Приклад 1.

Обидва операнда знаходяться в регістрах загального призначення: $(AX) = a$; $(CX) = b$. Для звернення до операндів використовується пряма регістрова адресація.

Символічний запис команди:

ADD AX, CX

Згідно табл. 12.2. машинне представлення цієї команди має вигляд:

000000dw mdreg r/m

За умовою операнди займають повнорозрядні регістри довжиною 1 слово, отже, необхідно встановити $w=1$.

Оскільки обидва операнди розташовуються в регістрах загального призначення, то кожен з них можна закодувати в полі reg. Тому команда може мати два різних представлення в машинному коді. При цьому, якщо в поле reg закодовано номер регістра AX, то біт приймача результату $d=1$.

Якщо в полі reg закодовано номер регістра CX, то біт приймача результату $d=0$ [12].

Тут і далі в записі команд b означає двійкове представлення, d – 16-е.

Після виконання команди в AX буде записана сума вмісту регістрів AX і CX, а покажчик команди IP збільшиться на довжину виконаної команди (2 байти) і буде вказувати на перший байт наступної команди.

Тут і далі представлення інформації будемо давати в 16-му виді, якщо інший не обговорено.

Якщо перед початком виконання команди (AX) = 0C34, (CX) = 1020, (IP) = 0012, то після її виконання (AX) = 1C54, (CX) = 1020, (IP) = 0014.



або



Кодування команд переходів

Кодування команд переходів і циклів має певну специфіку. Розглянемо машинне представлення цих команд докладніше.

При *безумовному внутрішньосегментному прямому переході* нове значення IP дорівнює сумі 8- або 16-розрядного зсуву й поточного значення IP. У якості поточного значення IP використовується адреса команди, записаної слідом за командою переходу.

Схему виконання операції наведено на рис. 12.4., де передбачається, що перед обчисленням адреси переходу вміст IP уже вказує на команду, наступну за командою переходу.

Команда, що має 8-розрядний зсув, називається командою короткого переходу й має в символічному записі після мнемоніки команди префікс short. Зсув записується в додатковому коді, який перед додаванням з поточним значенням IP розширюється знаком до 16 розрядів.

Таким чином, діапазон адрес переходів для команди короткого переходу становить -128...+127 байтів щодо поточного значення IP.

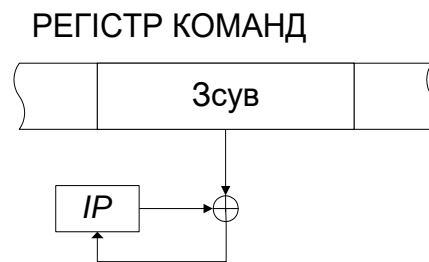


Рис. 12.4.Схема внутрішньосегментного прямого переходу

Приклад 1.

Команда `JMP short L` здійснює передачу керування команді, позначеної міткою L.

Нехай ця команда переходу записана за адресою 010A. Тоді якщо мітці L відповідає, наприклад, адреса 011A, то зсув у команді переходу буде дорівнювати:

$$011A - (010A + 2) = 011A - 010C = 011A + FEF4 = 0E$$

Тут операція віднімання замінена додаванням з використанням додаткового коду негативного числа. Перенос за межі розрядної сітки в операціях, пов'язаних з обчисленням зсуву, ігнорується.

Машинне представлення команди наступне [11]:

$$\begin{array}{cc} \text{коп} & \text{disp} \\ 11101011 & 00001110 \end{array} \quad b = EB0Eh$$

Зверніть увагу на те, що в якості поточного значення IP узята адреса команди переходу, збільшена на 2, тому що довжина самої команди переходу дорівнює 2 байтам.

Якщо команда, позначена міткою L, розташовується за адресою 00C1, то зсув буде дорівнювати [11]:

$$00C1 - 010C = 00C1 + FEF4 = FFB5$$

Отриманий зсув має довжину 2 байта, що неприпустимо для даного *формату команди*. Але тому що старший байт являє собою знакове розширення молодшого байта (FFB5h = 11111111 10110101b), то цей зсув можна закодувати в 1 байті, і команда буде мати машинне представлення: EBB5h.

Якщо мітці L відповідає адреса 0224, то необхідна величина зсуву, рівна 0224-010C=0118, не може бути записана в 8-розрядному форматі. Отже, за допомогою команди короткого переходу здійснити перехід на зазначену адресу неможливо.

Оцінка впливу структури програми на час її виконання

Час виконання програми складається із часу виконання кожної команди програми. *Час виконання команди* можна визначити, якщо помножити число тактів синхронізації, необхідних для виконання команди, на тривалість такту. Цей час можна виразити у вигляді суми базового часу виконання, який залежить від типу команди, і часу обчислення ефективної адреси, якщо операнд розташовується в пам'яті. При визначенні базового часу передбачається, що команда, яка виконується, вже обрана з пам'яті й перебуває в черзі команд.

А якщо ні, то потрібно врахувати тривалість додаткових тактів синхронізації, необхідних для вибірки команди.

Час обчислення ефективної адреси (ЕА) залежить від *режиму адресації*. Щоб визначити *час виконання команди*, слід урахувати вирівнювання операнда, тобто його розташування в оперативній пам'яті. Звертання до однобайтного операнду не вимагає додаткових тактів синхронізації. Час звертання до слова пам'яті залежить від його адреси. Якщо слово має непарну адресу, то його передача з оперативної пам'яті займає 2 цикли шини, що тривають по 4 такти синхронізації кожний. Отже, кожне звертання до слова з непарною адресою вимагає чотирьох додаткових тактів синхронізації.

Якщо при обчисленні фізичної адреси проводиться заміна сегментного регістру (замість заданого за замовчуванням використовується інший, визначений префіксом заміни), то *час виконання команди* збільшується на 2 такти.

Базовий час виконання деяких команд залежить також від значення операндів.

Так, час виконання команди множення, реалізованої за алгоритмом множення додаткових кодів із пропуском такту складання, визначається кількістю пар сусідніх незбіжних розрядів (01 або 10), тому що при комбінаціях 00 або 11 такт складання з нульовим доданком відсутній.

Тому, наприклад, для 8-розрядних операндів максимальний час множення буде при значенні множника 01010101, а мінімальний - при ненульовому множителю 10000000 (нагадаємо, що числа в персональній ЕОМ представляються в додатковому коді, тому зазначений код відповідає числу -128) [11]. Тобто в першому випадку команда множення буде виконуватися на 7 тактів додавання довше.

Для команд умовного переходу є два часи: менший відповідає випадку, коли умова не виконується й перехід не проводиться, а більший відповідає реалізації переходу. У другому випадку враховується необхідність нового заповнення черги команд і вибірки наступної команди. Це ж відноситься й до команд циклів.

12.3 Архітектура процесора. Взаємодія основних вузлів і пристроїв персонального комп'ютера при автоматичному виконанні команди. Архітектура 32-розрядного мікропроцесора.

Взаємодія основних вузлів і пристроїв персонального комп'ютера при автоматичному виконанні команди

Основні етапи автоматичного виконання програми в комп'ютері із класичною трьохадресною архітектурою вже були розглянуті.

Розглянемо особливості цього процесу на прикладі функціонування персонального комп'ютера, що використовує 16 розрядний мікропроцесор типу I8086.

Структуру такого комп'ютера наведено на рис. 12.5 [14]. У складі ЕОМ крім традиційних пристроїв комп'ютера із класичною архітектурою (оперативний запам'ятовуючий пристрій, арифметико-логічний пристрій і основні схеми пристрою керування) виділимо наступні блоки:

- регістрова пам'ять;
- блок формування адреси операнда (БФАО);
- двадцятирозрядні суматори для одержання *фізичних адрес* даних (Σ ФАД) і *фізичних адрес* команд (Σ ФАК).

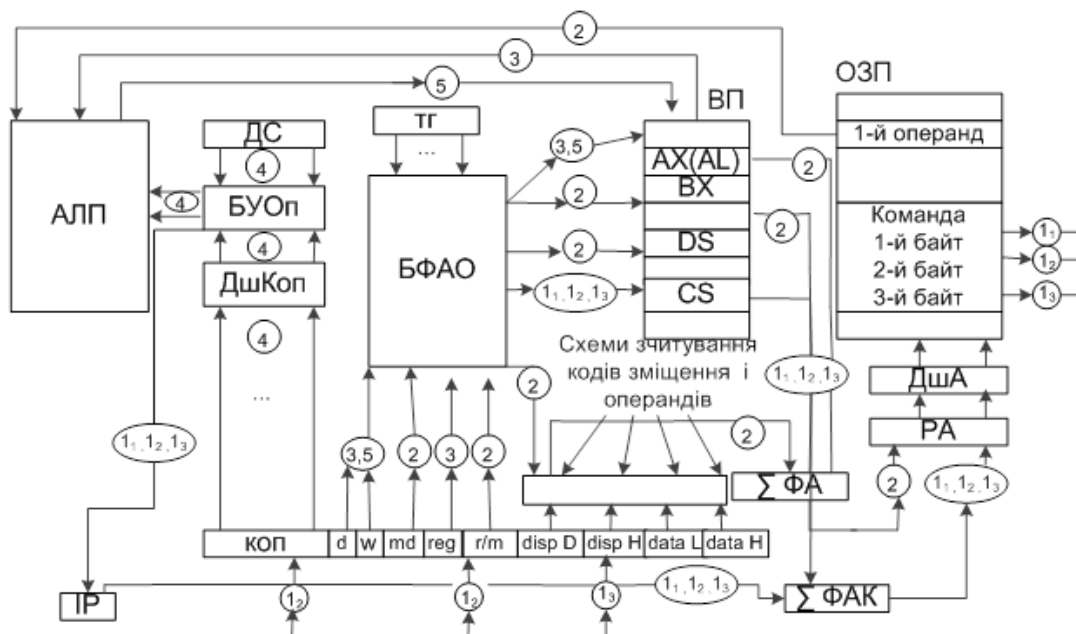


Рис. 12.5. Структурна схема базової моделі персональної ЕОМ

На входи *регістрової пам'яті* із БФАО надходять номери регістрів, до яких відбувається звертання.

На входи обраних регістрів надходять із АЛП результати виконання операції й значення *сегментних регістрів*, установлюваних операційною системою ЕОМ.

У регістрах зберігаються складові *ефективних адрес* даних, вихідні й проміжні дані, що брали участь у виконанні операції, старші 16 розрядів базових адрес *сегментів*. На блок формування адреси операнда надходять:

- сигнали від тактового генератора мікропроцесора;
- коди полів іншого байта команди, що виконується, що перебуває в РК;

ці коди визначають режими адресації першого (поля md і r/m) і другого (поле reg) операндів;

- коди двох молодших розрядів першого байта команди (d і w), які визначають, відповідно, операнд, на місце якого записується результат операції, і розрядність операндів. БФАО виробляє наступні вихідні сигнали:

- коди номера реєстрів, що обираються;
- сигнали зчитування кодів з обраних реєстрів;
- сигнали зчитування зсувів ($disp\ L$ і $disp\ H$);
- сигнали зчитування безпосередніх операндів ($data\ L$ і $data\ H$).

Суматори *фізичних адрес* використовуються для одержання адреси звертання до оперативної пам'яті з обліком її сегментної організації. Одним з доданків виступає початкова адреса *сегмента*, яка формується шляхом множення на 16 значення відповідного *сегментного реєстру*. Другий доданок - це зсув відносно початку *сегмента*.

Для *сегмента* коду таким зсувом є значення покажчика команд IP , а для *сегмента* даних – сформована блоком формування адреси операнда *ефективна адреса*.

Суть *етапів виконання команди* залишається без зміни в порівнянні із класичною ЕОМ:

- перший - вибір коду команди;
- другий і третій – вибір операндів;
- четвертий – виконання операції в АЛП;
- п'ятий – запис результату в оперативну або *реєстрову пам'ять*;
- шостий – формування адреси наступної команди, що виконується.

Але зміст цих *етапів* змінився [15].

Розглянемо виконання вищевказаних *етапів* на прикладі наступної команди:

$ADD\ AL, [BX+disp8]$

Припустимо, що її перший байт перебуває в гнізді ОЗП з адресою $i + 24 * [CS]$, тобто $[IP] = i$.

Перший етап. Код IP , тобто $[IP] = i$, передається на суматор $\Sigma ФАК$. На цей же суматор надходить код *сегментного регістру* команд $[CS]$ із РП, помножений на 16.

На виході $\Sigma ФАК$ сформується код *фізичної адреси* ОЗП, за яким знаходиться перший байт команди. Код з виходу $\Sigma ФАК$ надходить на регістр адреси ОЗП. З ОЗП вибирається перший байт команди й надсилається в регістр команд (для деякого спрощення припускаємо, що обмін інформацією між мікропроцесором і ОЗП відбувається байтами). І в завершенні цього *етапу* до IP додається 1.

Усі зазначені взаємодії відзначимо на схемі знаком 11. Ця послідовність дій буде повторена ще два рази для вибору другого й третього байтів команди, що виконується. Це відзначене на схемі знаками 12 і 13.

Другий етап. На початку другого *етапу* розшифровується код операції команди, що виконується. Якщо команда, що виконується, не порушує природнього порядку програми, що виконується, то здійснюється формування адреси першого операнда й вибірка цього операнда із РП або ОЗП ЕОМ.

У даній команді для першого операнда використовується регістровий відносний режим адресації. Відповідно, *ефективна адреса* визначається так: $EA = [BX] + disp8$.

У цьому випадку коди полів md і r/m другого байта з регістру команд надходять у БФАО й у такий спосіб комутують устаткування БФАО, що на його виході з'являються сигнали, що забезпечують зчитування:

- коду регістру BX ;
- коду $disp L$;
- коду *сегментного регістру* DS .

Усі зазначені коди надходять на суматор *фізичної адреси* даних $\Sigma\text{ФАД}$. При цьому забезпечується передача значення DS із зсувом на 4 розряди вліво (множення на 16). Сформований на $\Sigma\text{ФАД}$ код надходить на $РА\text{ ОЗП}$.

Відбувається вибірка байта даних, який направляється в АЛП. Виконання другого *етапу* завершено. Усі зазначені взаємодії пристроїв відзначимо на схемі цифрою 2.

Третій етап. Вибір другого операнда. У цьому випадку БФАО під впливом сигналу з розряду w і поля reg регістру команд видає сигнал звертання до регістру AL , код якого подається в АЛП. Усі взаємодії на цьому *етапі* відзначимо цифрою 3.

Четвертий етап. Виконання операції додавання в АЛП. Тут блок керування операціями видає ті сигнали в АЛП, які необхідні для виконання операції. Лінії взаємодії відзначимо цифрою 4.

П'ятий етап. Код виконаної операції з АЛП направляється до регістру AL ($d = 1$) *регістрової пам'яті*. Взаємодія відзначається цифрою 5.

Команда виконана. В IP перебуває основна складова адреси наступної команди програми: $(IP) = i + 3$. Тут *шостий етап* як окремий (автономний) *етап* виключений. Формування основної складової адреси наступної команди, що виконується, (покажчика команд) було реалізовано на першому *етапі*. Значення *сегментного регістру* команд в арифметичних командах не змінюється.

ЕОМ готова до виконання наступних команд програми.

Архітектура 32-розрядного мікропроцесора

В 1985 році фірма *Intel* випустила 32-розрядний мікропроцесор, що став засновником сімейства *IA-32*.

Розвиток цього сімейства пройшов ряд етапів, серед яких можна виділити наступні: реалізація блоку обробки чисел із плаваючою комою безпосередньо на кристалі МП (мікропроцесор *I486*), введення *MMX-технології* обробки даних з фіксованою крапкою за принципом *SIMD – single instruction multi data* (один

потік команд - безліч потоків даних) у мікропроцесорі *PentiumMMX* і розвиток цієї технології на числа із плаваючою комою (*SSE -streaming Simdextention*), що з'явилося вперше в МП *Pentium III* [4],[9]. Однак основні риси цієї архітектури аж до теперішнього часу залишаються незмінними [6].

Архітектура 32-розрядного мікропроцесора (рис. 12.6) суттєво відрізняється від архітектури 16-розрядного. Деякі із цих відмінностей чисто кількісні, інші носять принциповий характер.

Головна зовнішня відмінність – збільшення розрядності шини даних і шини адреси до 32 біт. Це, у свою чергу, пов'язане зі змінами в розрядності внутрішніх елементів мікропроцесора й у механізмі виконання деяких процесів, наприклад, формування *фізичної адреси*.

Регістри блоку обробки чисел з фіксованою крапкою стали 32-розрядними. До кожного з них можна звертатися як до одного подвійного слова (32 розряди). До молодших 16 розрядів цих регістрів можна звертатися так само, як і в 16-розрядному мікропроцесорі.

У блоці *сегментних регістрів* відбулися як кількісні, так і якісні зміни. До чотирьох регістрів CS, DS, SS і ES, які використовуються у реальному режимі додані ще два: FS і GS.

Хоча розрядність регістрів цього блоку залишилася колишньою (кожний по 16 біт), у формуванні *фізичної адреси* оперативної пам'яті вони використовуються по-іншому. При роботі мікропроцесора в так званому захищеному режимі вони призначаються для пошуку дескриптора (описувача) *сегмента* у відповідних системних таблицях, а вже в дескрипторі зберігається базова адреса й атрибути *сегмента*. Формування адреси в цьому випадку виконує блок сегментації диспетчера пам'яті [6].

Якщо крім *сегментів* пам'ять розбита ще й на *сторінки*, то остаточне обчислення *фізичних адрес* виконує блок керування *сторінками*.

Починаючи з мікропроцесора *I486*, до складу кристала мікропроцесора входить блок обробки чисел із плаваючою комою, що включає в себе вісім 80-розрядних регістрів для вистави знаків, мантис і порядків таких чисел.

На кристалі мікропроцесора розташовується також внутрішня кеш-пам'ять, яка являє собою особливим чином організовану швидкодіючу буферну пам'ять, призначену для зберігання інформації (команд і даних), що найбільш часто використовується.

У різних моделях мікропроцесорів обсяг кеш-пам'яті становить від 8 Кбайт до 512 Кбайт.

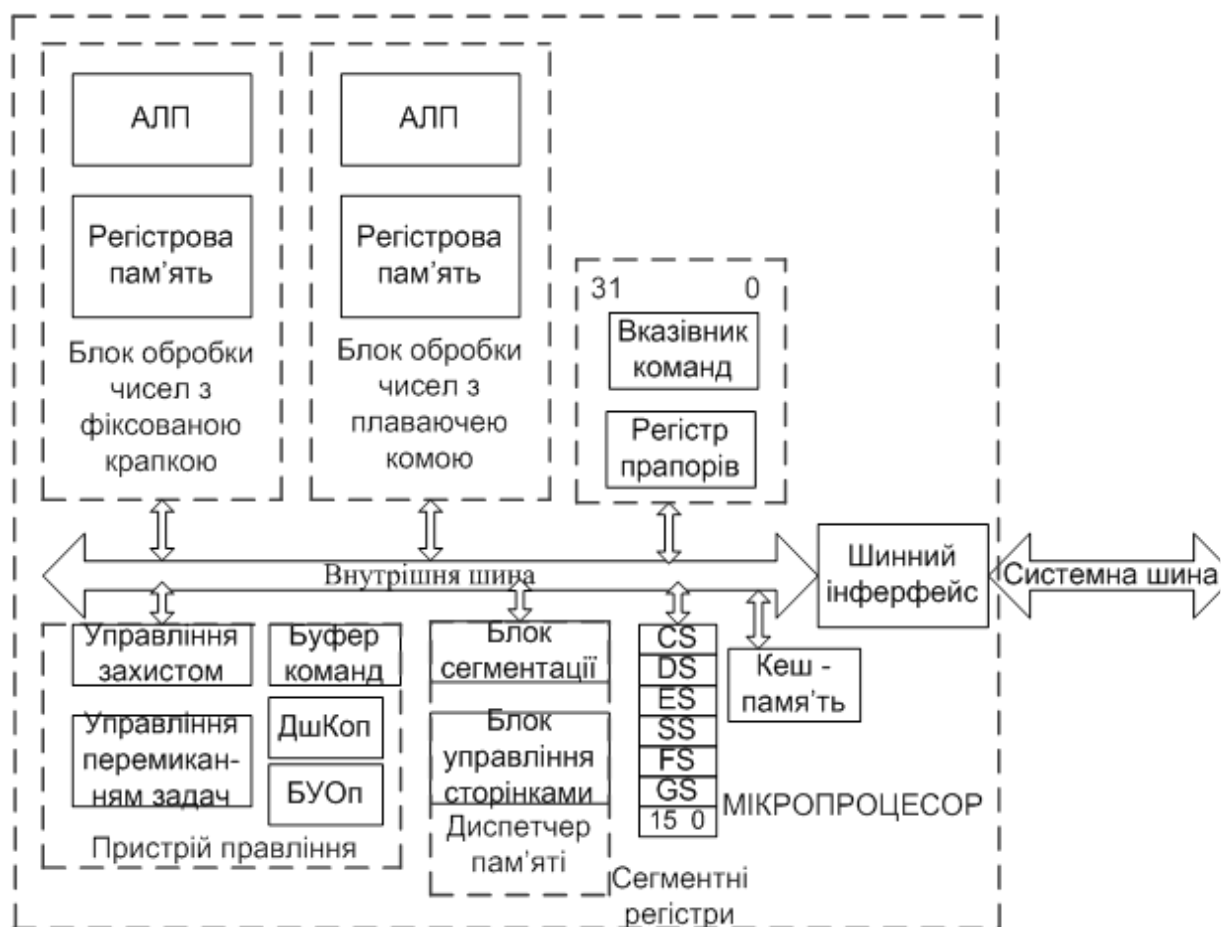


Рис. 12.6. Структура 32-розрядного мікропроцесора

Мікропроцесор на апаратному рівні підтримує *мультипрограмний режим роботи* ЕОМ, тобто можливість мати в пам'яті одночасно трохи готових до виконання програм, запуск яких здійснюється операційною системою відповідно

до алгоритмів її функціонування або залежно від особливих ситуацій, що складаються в роботі зовнішніх пристроїв.

Із цією можливістю нерозривно зв'язані засоби *захисту пам'яті*, які забезпечують контроль над недозволеними взаємодіями між окремими програмами. Вони містять у собі *захист* при керуванні пам'яттю й *захист* за привілеями [17].

Головні особливості розширеного формату команди - можливість використовувати кожен з регістрів загального призначення в кожному з режимів адресації, а також додавання ще одного режиму адресації - відносного базового індексного з масштабуванням. При цьому *ефективна адреса* формується таким способом:

$$EA = (base) + (index) \cdot scale + disp,$$

де (base) – значення базового регістру; (index) – значення індексного регістру; scale – величина масштабного множника (scale = 1,2,3,4); disp – значення зсуву, закодованого в самій команді.

Відзначимо, що в 32-розрядній архітектурі *ефективну адресу* звичайно називають зсувом (offset), у той же час відрізняючи його від зсуву, що кодується в самій команді (displacement).

Оцінка продуктивності ідеального конвеєра

Виконання кожної команди складається з ряду послідовних етапів (кроків, стадій), суть яких не міняється від команди до команди. З метою збільшення швидкодії процесора й максимального використання всіх його можливостей у сучасних мікропроцесорах використовується *конвеєрний принцип обробки інформації*. Цей принцип має на увазі, що в кожний момент часу процесор працює над різними стадіями виконання декількох команд, причому на виконання кожної стадії виділяються окремі апаратні ресурси. По черговому тактовому імпульсу кожна команда в *конвеєрі* просувається на наступну стадію обробки, виконана команда залишає *конвеєр*, а нова надходить у нього.

У різних процесорах кількість і суть етапів різняться. Розглянемо принципи *конвеєрної* обробки інформації на прикладі п'ятиступінчатого *конвеєра*, у якому виконання команди складається з наступних етапів:

1. IF(*Instruction Fetch*) – зчитування команди в процесор;
2. ID(*Instruction Decoding*) – декодування команди;
3. OR (*Operand Reading*) – зчитування операндів;
4. EX (*Executing*) – виконання команди;
5. WB (*Write Back*) – запис результату.

Виконання команд у такому *конвеєрі* наведено в табл. 12.5.

Так як в кожному *такті* можуть виконуватися різні стадії обробки команд, то тривалість *такту* вибирається виходячи з максимального часу виконання всіх стадій.

Крім того, слід враховувати, що для передачі команди з однієї стадії на іншу потрібен певний додатковий час (Δt), пов'язаний із записом проміжних результатів обробки в буферні регістри.

Таблица 12.5.

Принципи конвеєрної обробки інформації на прикладі
п'ятиступінчатого конвеєра

Команда	Такт								
	1	2	3	4	5	6	7	8	9
i	IF	ID	OR	EX	WB				
i+1		IF	ID	OR	EX	WB			
i+2			IF	ID	OR	EX	WB		
i+3				IF	ID	OR	EX	WB	
i+4					IF	ID	OR	EX	WB

Нехай для виконання окремих стадій обробки потрібні наступні витрати часу (у деяких умовних одиницях):

$$T_{IF} = 20, T_{ID} = 15, T_{OR} = 20, T_{EX} = 25, T_{WB} = 20.$$

Тоді, припускаючи, що додаткові витрати часу становлять $dt = 5$ одиниць, одержимо час *такту*:

$$T = \max\{TIF, TID, TOR, TEX, TWB\} + \Delta t = 30$$

Оцінімо час виконання однієї команди й деякої групи команд при послідовній і конвеєрній обробці.

При послідовній обробці час виконання N команд складе:

$$T_{\text{посл}} = N * (TIF + TID + TOR + TEX + TWB) = 100N$$

Аналіз табл. 12.5. показує, що при конвеєрній обробці після того, як результат виконання першої команди отриманий, результат чергової команди з'являється в наступному *такті* роботи процесора. Отже,

$$T_{\text{конв}} = 5T + (N - 1) * T$$

Очевидно, що при досить тривалій роботі *конвеєра* його швидкодія буде суттєво перевищувати швидкодію, що досягається при послідовній обробці команд. Це збільшення буде тим більше, чим менше тривалість *такту конвеєра* й чим більше кількість виконаних команд.

Скорочення тривалості *такту* досягається, зокрема, розбивкою виконання команди на велику кількість етапів, кожний з яких містить у собі відносно прості операції й тому може виконуватися за короткий проміжок часу. Так, якщо в мікропроцесорі *Pentium* довжина *конвеєра* становила 5 ступенів (при максимальній тактовій частоті 200 МГц), то в *Pentium-4* – вже 20 ступенів (при максимальній тактовій частоті на сьогоднішній день 3,4 ГГц).

Конфлікти в конвеєрі й способи мінімізації їх впливу на продуктивність процесора

Значна перевага конвеєрної обробки перед послідовною має місце в *ідеальному конвеєрі*, у якому відсутні конфлікти й усі команди виконуються один за одним без перезавантаження *конвеєра*.

Наявність конфліктів знижує реальну продуктивність *конвеєра* в порівнянні з ідеальним випадком [15].

Конфлікти - це такі ситуації в конвеєрній обробці, які перешкоджають виконанню чергової команди в призначеному для неї *такті*. Конфлікти діляться на три групи:

- структурні,
- за керуванням,
- за даними.

Структурні конфлікти виникають у тому випадку, коли апаратні засоби процесора не можуть підтримувати всі можливі комбінації команд у режимі одночасного виконання зі сполученням.

Причини *структурних конфліктів*.

1. Не повністю конвеєрна структура процесора, при якій деякі ступені окремих команд виконуються більше одного *такту*.

Нехай етап виконання команди $i+1$ займає 3 *такти*. Тоді діаграма роботи конвеєра буде мати вигляд, представлений у табл. 12.6.

При цьому в роботі конвеєра виникають так звані "міхури" (обробка команд $i+2$ і наступних за нею, починаючи з *такту* 6), які знижують продуктивність процесора.

Таблиця 12.6.

Діаграма роботи конвеєра

Команда	Такт								
	1	2	3	4	5	6	7	8	9
i	IF	ID	OR	EX	WB				
$i+1$		IF	ID	OR	EX	EX	EX	WB	
$i+2$			IF	ID	OR	O	O	EX	WB
$i+3$				IF	ID	OR	O	O	EX
$i+4$					IF	ID	OR	O	O

Цю ситуацію можна було б ліквідувати двома способами. Перший припускає збільшення часу *такту* до такої величини, яка дозволила б усі етапи будь-якої команди виконувати за один *такт*. Однак при цьому суттєво знижується ефект конвеєрної обробки, тому що всі етапи всіх команд будуть виконуватися значно довше, у той час як звичайно декількох *тактів* вимагає виконання лише окремих етапів дуже невеликої кількості команд.

Другий спосіб припускає використання таких апаратних розв'язків, які дозволили б значно знизити витрати часу на виконання даного етапу (наприклад, використовувати матричні схеми множення).

Але це призведе до ускладнення схеми процесора й неможливості реалізації на цій ВІС інших, функціонально більш важливих, вузлів [12]. Так як представлена в табл. 12.6 ситуація виникає при реалізації команд, що відносно рідко зустрічаються в програмі, то звичайно розроблювачі процесорів шукають компроміс між збільшенням тривалості *такту* й ускладненням того або іншого пристрою процесора.

2. Недостатнє дублювання деяких ресурсів.

Одним з типових прикладів служить конфлікт через доступ до запам'ятовувальних пристроїв. З табл. 12.5 видно, що у випадку, коли операнди й команди перебувають в одному запам'ятовувальному пристрої, починаючи з *такту* 3, роботу *конвеєра* необхідно постійно припиняти, оскільки різні команди в тому самому *такті* звертаються до пам'яті на зчитування команди, вибірку операнда, запис результату.

Боротьба з конфліктами такого роду проводиться шляхом збільшення кількості однотипних функціональних пристроїв, які можуть одночасно виконувати ті самі або схожі функції. Наприклад, у сучасних мікропроцесорах звичайно розділяють кеш-пам'ять для зберігання команд і кеш-пам'ять даних, а також використовують багатопортову схему доступу до регістрової пам'яті, при якій до регістрів можна одночасно звертатися по одному каналу для запису, а по іншим - для зчитування інформації. Конфлікти через виконавчі пристрої звичайно згладжуються введенням до складу мікропроцесора додаткових блоків.

Так, у мікропроцесорі Pentium-4 передбачено 4 АЛП для обробки цілочисельних даних. Процесори, що мають у своєму складі більше одного *конвеєра*, називаються *суперскалярними*.

Недоліком суперскалярних мікропроцесорів є необхідність синхронного просування команд у кожному з *конвеєрів*. У табл. 12.7 представлена послідов-

ність виконання команд у мікропроцесорі, що має два *конвеєри*, за умови, що команді ДО1 потрібно 3 *такти* на етапі EX.

При цьому команди будуть завершуватися в послідовності

ДО 2- ДО 4- ДО 1- ДО 6-...

Отже, для забезпечення правильної роботи суперскалярного мікропроцесора при виникненні затору в одному з *конвеєрів* повинні припиняти свою роботу й інші.

А якщо ні, то може порушитися вихідний порядок завершення команд програми. Але такі припинення суттєво знижують швидкість процесора. Рішення цієї проблеми полягає в тому, щоб дати можливість виконуватися командам в одному *конвеєрі* незалежно від ситуації в інших *конвеєрах*. Це приводить до *неупорядкованого виконання команд*. При цьому команди, які стоять у програмі пізніше, можуть завершитися раніше команд, що стоять спереду.

Апаратні засоби мікропроцесора повинні гарантувати, що результати виконаних команд будуть записані в приймач у тому порядку, у якому команди записані в програмі. Для цього в мікропроцесорі результати етапу виконання команди звичайно зберігаються в спеціальному буфері відновлення послідовності команд. Запис результату чергової команди із цього буфера в приймач результату проводиться лише після того, як виконані всі попередні команди й записані їхні результати.

Конфлікти за керуванням виникають при конвеєризації команд переходів і інших команд, що змінюють значення лічильника команд.

Суть конфліктів цієї групи найбільше зручно проілюструвати на прикладі команд умовного переходу. Нехай у програмі, представленої в табл. 12.5, команда $i+1$ є командою умовного переходу, що формує адресу наступної команди залежно від результату виконання команди i . Команда i завершить своє виконання в *такті* 5. У той же час команда умовного переходу вже в *такті* 3 повинна прочитати необхідні їй ознаки, щоб правильно сформулювати адресу наступної команди.

Якщо конвеєр має більшу глибину (наприклад, 20 ступенів), то проміжок часу між формуванням ознаки результату й *тактом*, де він аналізується, може бути ще більшим. В інженерних завданнях приблизно кожна шоста команда є командою умовного переходу, тому припинення конвеєра при виконанні команд переходів до визначення дійсного напрямку переходу суттєво позначається на продуктивності процесора.

Таблиця 12.7.

Команди умовного переходу

Етап	Такт													
	1		2		3		4		5		6		7	
IF	K1	K2	K3	K4	K5	K6	K7	K8	K7	K9	K7	K10	K11	K12
ID			K1	K2	K3	K4	K5	K6	K5	K8	K5	K9	K7	K10
OR					K1	K2	K3	K4	K3	K6	K3	K8	K5	K9
EX							K1	K2	K1	K4	K1	K6	K3	K8
WB										K2		K4	K1	K6

Найбільш ефективним методом зниження втрат від *конфліктів по керуванню* служить *передбачення переходів*.

Суть даного методу полягає в тому, що при виконанні команди умовного переходу спеціальний блок мікропроцесора визначає найбільш імовірний напрямок переходу, не чекаючи формування ознак, на підставі аналізу яких цей перехід реалізується. Процесор починає вибирати з пам'яті й виконувати команди по передвіщеним напрямках програми (так зване *виконання по припущенню*, або "спекулятивне" виконання).

Однак так як напрямок переходу може бути передбачено невірно, то одержувані результати з метою забезпечення можливості їх анулювання не записуються до пам'яті або регістру (тобто для них не виконується етап WB), а накопичуються в спеціальному буфері результатів [11].

Якщо після формування аналізованих ознак виявилось, що напрямок переходу обраний вірно, усі отримані результати переписуються з буфера за місцем призначення, а виконання програми триває у звичайному порядку. Якщо напрямок переходу передбачено невірно, то буфер результатів очищується.

Також очищується й *конвеєр*, що містить команди, що перебувають на різних етапах обробки, що впливають за командою умовного переходу. При цьому анулюються результати всіх уже виконаних етапів цих команд. *Конвеєр* починає завантажуватися з першої команди іншої гілки програми. Так як конвеєрна обробка ефективна при великому числі послідовно виконаних команд, то перезавантаження конвеєра приводить до значних втрат продуктивності. Тому питанням ефективного передбачення напрямку розгалуження розроблювачі всіх мікропроцесорів приділяють велику увагу.

Методи передбачення переходів діляться на статичні й динамічні. При використанні статичних методів до виконання програми для кожної команди умовного переходу вказується напрямок найбільш імовірного розгалуження. Ця вказівка робиться або програмістом за допомогою спеціальних засобів, наявних у деяких мовах програмування, по досвіду виконання аналогічних програм або результатам тестового виконання програми, або програмою-компілятором за закладеними у ній алгоритмами.

Методи динамічного прогнозування враховують напрямки переходів, що реалізовувалися цієї командою при виконанні програми.

Наприклад, підраховується кількість переходів, виконаних раніше по тому або іншому напрямку, і на підставі цього визначається напрямок переходу при наступнім виконанні даної команди.

У сучасних мікропроцесорах імовірність правильного передбачення напрямку переходів досягає 90-95 % [6].

Конфлікти за даними виникають у випадках, коли виконання однієї команди залежить від результату виконання попередньої команди.

Під час обговорення цих конфліктів будемо припускати, що команда *i* передує команді *j*.

КОНТРОЛЬНІ ПИТАННЯ

1. Зазначте, чим визначається формат команди?
2. Назвіть відомі вам режими адресації. В чому полягає суть кожного з них?
3. Дайте визначення поняттям «ефективна адреса» та «фізична адреса».
4. Наведіть класифікацію команд переходів.
5. Дайте коротку характеристику кожній з команд переходів.
6. Для чого потрібні команди близького переходу?
7. Які особливості розміщення нових значень IP і CS при міжсегментному непрямому переході?
8. У чому полягає відмінність внутрішньосегментного непрямого переходу від міжсегментного непрямого в символічному записі команди?
9. Що таке час виконання програми? Як можна його визначити?
10. Назвіть основні блоки, з яких складається ЕОМ.
11. Дайте визначення поняття «дизасемблер». Для чого вони використовуються?
12. У якому випадку час виконання команди збільшується на 2 такти?
13. Які вихідні сигнали виробляє блок формування адреси операнда (БФАО)?
14. Що являє собою конвеєрний принцип обробки інформації?
15. Назвіть етапи, з яких складається виконання команд у п'ятиступінчатому конвеєрі.
16. Дайте визначення поняттю «конфлікт» в контексті конвеєрної обробки інформації та назвіть їх основні типи. Який режим роботи підтримує мікропроцесор на апаратному рівні?
17. Зазначте шість етапів виконання команди та наведіть коротку характеристику кожного з них.
18. Наведіть класифікацію типів конфліктів.
19. Назвіть основні причини структурних конфліктів.

20. За яких обставин виникають конфлікти за керуванням. У чому полягає суть таких конфліктів?

13. УНІВЕРСАЛЬНІ ПРОЦЕСОРИ

13.1. Регістрова структура універсального мікропроцесора

Регістрова структура універсального мікропроцесора

В універсальному 32-розрядному мікропроцесорі виділяють наступні *групи регістрів*:

- основні функціональні регістри;
- регістри процесора із плаваючою крапкою;
- системні регістри;
- регістри налагодження й тестування.

Перші дві групи регістрів використовуються прикладними програмами, останні дві групи - системними програмами, що мають найвищий рівень привілеїв.

Розглянемо кожну із цих груп докладніше.

Основні функціональні регістри

До складу регістрів цієї групи входять:

- регістри загального призначення;
- регістр показчика команд;
- регістр прапорів;
- сегментні регістри.

Склад і структуру *регістрів загального призначення* наведено на рис. 13.1.

Блок складається з восьми 32-розрядних регістрів. До кожного з них можна звертатися як до одного подвійного слова (32 розряди).

Відзначимо, що поняття "слово" у даній архітектурі не ідентично розрядності мікропроцесора. Історично склалося так, що під *словом* розуміється одиниця інформації довжиною 2 байта, або 16 двійкових розрядів. До молодших 16 розрядів регістрів загального призначення можна звертатися так само, як і в 16-розрядному мікропроцесорі (AX, BX...SP). Чотири 16-розрядних регістру AX, BX, CX, DX допускають звертання окремо до свого старшого й молодшого

байта. Тим самим регістри дозволяють на програмному рівні працювати або з вісьмома 32-розрядними, або з вісьмома 16-розрядними, або з вісьмома 8-розрядними регістрами [12].

31	16	15	0
	(AH)	AX	(AL)
	(BH)	BX	(BL)
	(CH)	CX	(CL)
	(DH)	DX	(DL)
		SI	
		DI	
		BP	
		SP	
← 32 →			

Рис. 13.1. Регістри загального призначення

Усі ці регістри використовуються для зберігання проміжних результатів обчислень і складових частин адреси при різних режимах адресації операндів, розташованих у пам'яті.

Крім того, ряд регістрів цього блоку мають своє, властиве тільки їм призначення:

- EAX/AX/AL – регістр-акумулятор, використовується для скорочення довжини команди при роботі з безпосередніми операндами;
- AX/AL – приймач (джерело) даних у командах введення (виведення) даних з (в) зовнішнього пристрою;
- DX – визначає адреси ЗП в командах введення (виведення) даних;
- ECX – використовується в якості лічильника циклів у командах циклів;
- BP, SP – використовуються при роботі зі стеком;
- ESI, EDI (DI, SI) – визначають положення рядків у пам'яті в командах обробки рядків.

Регістр показчика команд і регістр прапорів мають довжину 32 розряди.

Молодше слово кожного із цих регістрів (розряди 0-15) функціонально відповідає аналогічним розрядам в 16-розрядному мікропроцесорі (рис. 13.2).

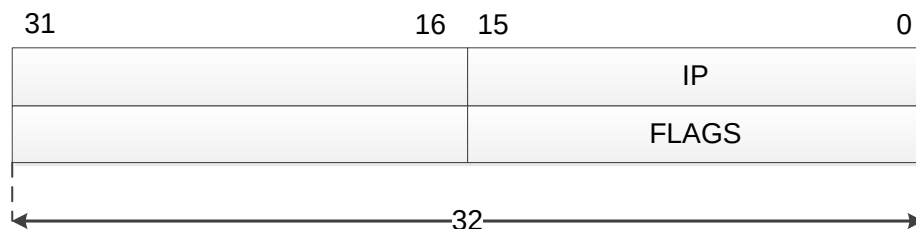


Рис. 13.2. Регістр показчика команд і регістр прапорів

Регістр показчика команд EIP зберігає зсув адреси команд відносно початку сегмента коду (сегмента команд).

Регістр прапорів EFLAGS містить ознаки результату виконаної команди, а також розряди, що управляють роботою мікропроцесора: обробкою маскованих переривань, послідовністю завдань, що викликаються, введенням - виведенням - і послідовністю інших дій. Із цих прапорів розглянемо тільки найбільш значимі й цікаві з огляду подальшого вивчення роботи мікропроцесора.

До *бітів стану* регістру прапорів відносяться:

- ZF – ознака нуля результату (ZF = 1, якщо всі розряди результату дорівнюють 0);
- SF – знак результату (SF = 1, якщо старший розряд результату дорівнює 1, тобто якщо результат негативний);
- OF – ознака переповнення (OF = 1, якщо при виконанні арифметичних операцій над числами зі знаком відбувається переповнення розрядної сітки);
- CF – прапор переносу (CF = 1, якщо виконання операції додавання приводить до переносу за межі розрядної сітки), установлюється також у деяких інших операціях;
- PF – ознака парності (доповнює до непарного числа одиниць молодший байт результату);

- AF – прапор підлоги переносу (використовується при операціях над двійково-десятковими числами);

- DF – установлюється користувачем і визначає порядок обробки рядків символів у відповідних командах: декремент (при DF = 1) або інкремент (при DF = 0) змісту індексних реєстрів ESI, EDI (SI, DI) після обробки одного символу.

До складу *прапорів керування* входять:

- IF – *прапор переривань* (при IF = 1 дозволяється обробка маскованих апаратних переривань);

- TF – прапор пастки, або трасування (при TF = 1 після виконання кожної команди виникає переривання, яке використовується відладчиками);

- NT – біт вкладеного завдання (показує, що дане завдання було викликано з іншої програми, аналогічно підпрограмі, і повернення із цього завдання повинно проводитися по механізму перемикавання завдань);

- IOPL – 2-розрядне поле *рівня привілеїв введення/виведення* (визначає рівень привілеїв програм, яким дозволено виконання операції введення-виведення);

- VM – режим віртуального мікропроцесора i8086 (при роботі мікропроцесора в захищеному режимі установка VM = 1 викликає перемикання в режим віртуального мікропроцесора i8086; у цьому випадку мікропроцесор функціонує як швидкий МП i8086, але реалізує механізми захисту пам'яті, сторінкової адресації й ряд інших додаткових можливостей; біт VM може бути встановлений тільки в захищеному режимі).

Блок *сегментних реєстрів* складається із шести 16-розрядних реєстрів, які вказують на різні сегменти, розташовані в пам'яті комп'ютера [11]:

- CS (Code Segment) – сегмент коду програми;
- DS (Data Segment) – сегмент даних;
- SS (Stack Segment) – сегмент стеку;
- ES, FS, GS – додаткові сегменти даних.

При роботі мікропроцесора в реальному режимі в сегментному реєстрі містяться старші 16 розрядів 20-розрядної базової адреси сегмента. Фізична адреса початку сегмента виходить множенням цієї величини на 16:

$$A_{\text{баз.сегм}} = (\text{сегментний реєстр}) \times 16$$

20-розрядна адреса, що отримується, дозволяє адресувати пам'ять ємністю 220 байт = 1 Мбайт. При цьому сегменти мають постійну довжину 216 байт. Розроблювачі перших персональних комп'ютерів вважали, що оперативна пам'ять, більша, ніж 1 Мбайт, ніколи не буде потрібна користувачеві, тому вся архітектура будувалася виходячи саме із цього положення.

При переході до 32-розрядної архітектури стало необхідним забезпечити можливість адресації пам'яті ємністю до 232 байт. Крім того, введення захищеного режиму роботи мікропроцесора вимагало зберігання великої кількості додаткової інформації про сегмент: його довжину, яка стала змінною, рівні привілеїв, його типи і т.д. Просте збільшення розрядності сегментних реєстрів до 32 біт не забезпечило б можливості зберігання всієї цієї інформації. Тому всі дані про сегмент стали розміщатися в спеціальних структурах - *дескрипторах (описувачах) сегментів*, які зберігаються в *таблицях дескрипторів*, розташованих у пам'яті, а сегментні реєстри, зберігши свою первісну довжину в 16 розрядів, містять так званий селектор (показчик), який використовується для того, щоб знайти потрібний *дескриптор* у цих таблицях.

Реєстри процесора із плаваючою крапкою

До цієї групи реєстрів відносяться (рис. 13.3): реєстри даних; реєстри тегів; реєстр стану; показники команд і даних FPU; реєстр керування FPU.

Блок реєстрів даних. Доступний або як стек (його вершина TOP визначена в реєстрі станів FPU), або як набір пронумерованих реєстрів. Старший розряд 80-розрядного реєстру даних кодує знак мантиси числа із плаваючою крапкою, що зберігається в ньому.

Наступне поле відведене під кодування порядку. Порядок представлений у вигляді так званого машинного, або зміщеного, порядку ($P_{\text{зс}}$) без знака:

$$П_{зс} = П + \Delta$$

де $П$ – дійсний порядок числа, а Δ величина визначається таким способом: $\Delta = | -П_{\text{макс}} |$. При цьому найбільший по модулю негативний дійсний порядок перетвориться в нульовий зміщений, а всі інші дійсні порядки перетворяться в позитивні.

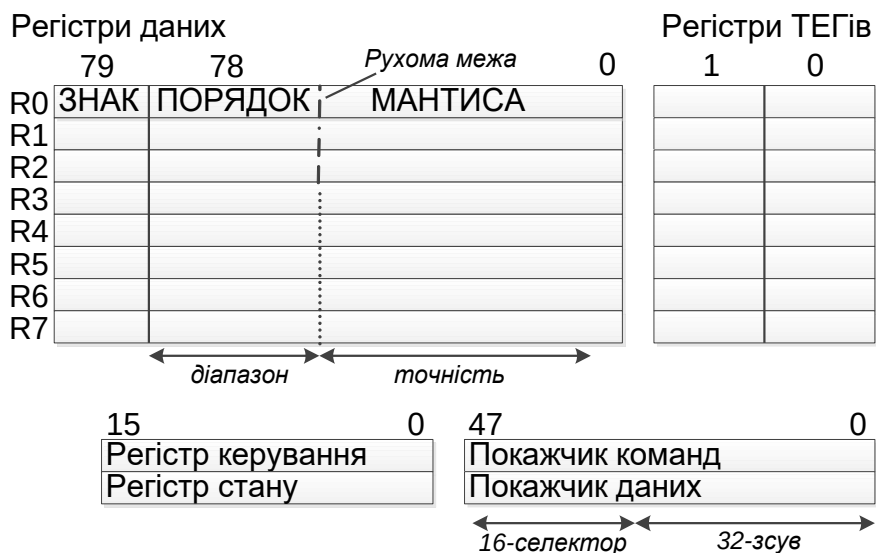


Рис. 13.3. Структура регістрів процесора із плаваючою крапкою

Тим самим спрощуються операції обробки чисел із плаваючою крапкою. В останньому полі регістру даних записується мантиса числа. Кількість розрядів, що виділяються під поле порядку й поле мантиси, визначається регістром керування FPU.

Даний блок може також використовуватися для зберігання двійково-десятикових чисел.

Вони представляються в упакованій формі й містять 18 тетрад, кожна з яких відповідає одному десятиковому розряду.

Для представлення знака такого числа використовується старший розряд старшого байта (біт 79), в інших розрядах цього байта встановлюються нулі. Крім цього мікропроцесор може обробляти символічні дані, дані типу «ря-

док» і типу «показчик». Мікропроцесор може обробляти числа наступних типів (табл. 13.1):

Таблиця 13.1.

Типи чисел 32-розрядного мікропроцесора

Тип	Розмір, байт	Діапазон	Обробка
Цілі без знака	1	0...255	АЛП ФТ
	2	0...65535	
	4	0...4,3*10 ⁹	
Цілі зі знаком	1	-128...+127	АЛП ФТ
	2	-32768...+32767	
	4	2,1*10 ⁹ ...+2,1*10 ⁹	
	8	9,1*10 ¹⁸ ...+9*10 ¹⁸	FPU
Із плаваючою крапкою	4 (1+8+23)мантиса	±3,37*10 ³⁵	FPU
	8 (1+11+52)	±1,67*10 ³⁰⁸³⁰⁸	
	10 (1+15+64)	±1,1*10 ⁴⁹³²	
Двійково-десяткові числа	1 розпакований	0...9	АЛП ФТ
	1 упакований	0...99	АЛП ФТ
	10 упакованих	0..9...9(18 цифр)	FPU

Регістр тегів. Визначає вміст регістру даних з метою оптимізації обробки:

- 00 – достовірне значення;
- 01 – нуль (нульове значення);
- 10 – не-числа (наприклад, нескінченність);
- 11 – порожньо (зміст регістрів не визначений).

Операції із плаваючою крапкою вимагають досить багато часу.

Використання тегів дозволяє в певних випадках скоротити час виконання команди. Наприклад, якщо відомо, що один із співмножників дорівнює нулю, то добутку можна привласнити нульове значення без виконання будь-яких дій.

Регістр стану містить показчик вершини блоку даних, що працює в режимі стека (TOP), ознаки результату й помилок, що виникають при виконанні

операції в FPU, а також прапор переповнення й антипереповнення стека регістрів даних.

Регістр керування управляє наближенням (до найближчого значення, униз, нагору, до нуля), точністю (довжина мантиси 24, 53 або 64 біта), а також містить маску ознак помилок, що фіксуються у регістрі стану.

Показчики команд і даних містять адресу команди, що викликала помилку, і адресу використаного операнда. Ці регістри мають 48-розрядний формат: 16 розрядів містять селектор відповідного сегмента, а інші 32 розряди – зсув у ньому.

Системні регістри

Системні регістри керують функціонуванням мікропроцесора в цілому й режимами роботи окремих його блоків. Ці регістри доступні тільки в захищеному режимі для програм, що мають максимальний рівень привілеїв. Вони включають до свого складу 2 групи регістрів:

- регістри керування (CR0...CR4);
- регістри системних адрес і системних сегментів.

Регістр керування CR0 містить біти, що визначають режим роботи процесора:

- PE – дозвіл захисту: установка PE = 1 переводить мікропроцесор у захищений режим;
- PG – вмикання сторінкової адресації пам'яті (при PG = 1 сторінковий механізм включений);
- CD, NW- керування режимами роботи внутрішньої кеш-пам'яті (CD = 1 – заборона заповнення кеш-пам'яті; NW = 1 – заборона наскрізного запису).

Ряд бітів (MP, EM, TS, NE) керують режимами роботи FPU.

Регістр CR1 був зарезервований для наступного розвитку. Однак починаючи із МП Pentium у мікропроцесорах з'явився регістр керування CR4, а регістр CR1 так і залишився зарезервованим.

Регістр CR2 містить лінійну адресу, яка викликала сторінкову помилку, наприклад, відсутність сторінки в оперативній пам'яті або недостатній рівень привілеїв.

У регістрі CR3 перебуває базова адреса каталогу таблиці сторінок (старші 20 розрядів), а також біти PCD і PWT кеш-пам'яті, що управляють роботою, при сторінковій адресації (при PCD = 1 завантаження змісту сторінки в кеш-пам'ять заборонена; при PWT = 1 реалізується режим наскрізного запису, а при PWT = 0 - зворотного запису).

Регістр CR4 містить біти, що забезпечують розширення функціональних можливостей мікропроцесора, починаючи з Pentium. Зокрема, він містить наступні керуючі розряди:

- VME, PVI – керують роботою віртуальних переривань;
- PAE – забезпечує розширення фізичної адреси до 36 розрядів (при PAE = 1); PGE – визначає деякі сторінки (часто використовувані або використовувані декількома процесорами) як глобальні (при PGE = 1);
- PSE – розширює розмір сторінок, що адресуються, до 4 Мбайт (при PSE = 1), при PSE = 0 зберігає розмір сторінки 4 Кбайт.

Регістри системних адрес і системних сегментів наведено на рис. 13.4.

До їх числа входять GDTR – *регістр глобальної таблиці дескрипторів* і IDTR – *регістр таблиці дескрипторів переривань*. У цих регістрах визначаються базова адреса й розмір відповідної таблиці.

До цієї групи відносяться також LDTR – *регістр локальної таблиці дескрипторів* і TR – *регістр завдань*. Регістри LDTR і TR являють собою селектори, які вказують на стан дескрипторів, що описують відповідно сегмент, що містить локальну таблицю дескрипторів, і сегмент стану завдання (Task State Segment – TSS) [16].

Використання цих регістрів надалі буде розглянуто більш докладно.

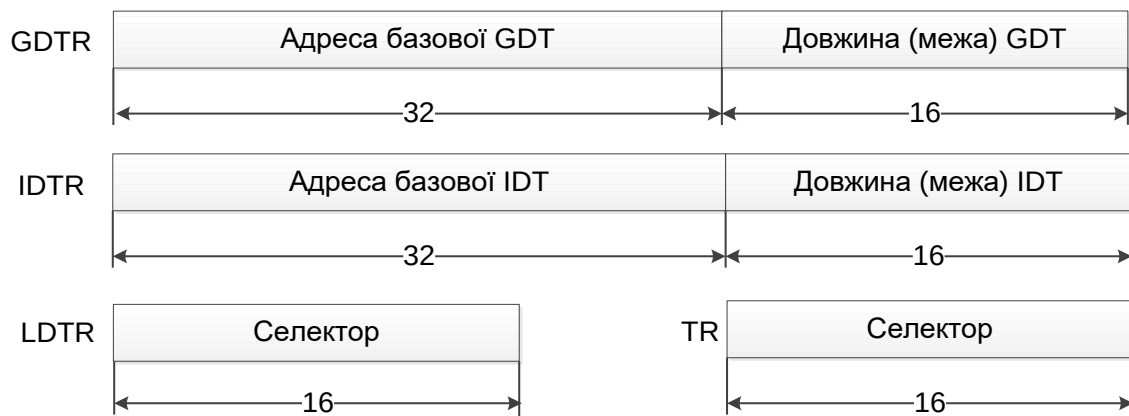


Рис. 13.4. Структура регістрів системних адрес і системних сегментів

Регістри налагодження й тестування

32-розрядні *регістри налагодження* (DR0...DR7) мають наступне призначення:

- DR0...DR3 – містять лінійні адреси 4 контрольних точок зупинок при налагодженні;
- DR4 і DR5 зарезервовані;
- DR6 - регістр стану: показує поточний стан МП при зупинці в цих точках;
- DR7 – регістр керування: задає умови зупинки в контрольних точках.

Регістри DR4 і DR5 не використовуються.

Регістри тестування (TR3...TR7) використовуються при тестуванні кеш-пам'яті й буферу асоціативної трансляції адрес сторінок (TLB).

З розвитку архітектури мікропроцесора їх кількість розширилася й доповнилася новим змістом. Зокрема, за допомогою регістру TR12 можна заборонити передбачення й трасування розгалужень, паралельне виконання інструкцій і виконати деякі інші дії.

Короткі підсумки. У розділі розглянуті призначення й склад регістрової структури універсального мікропроцесора, багато в чому визначальні архітектурні особливості мікропроцесора.

Логічний адресний простір

Для адресації операндів у фізичному адресному просторі програми використовують логічну адресацію. Процесор автоматично трансліює логічні адреси у фізичні, що потім видаються на системну шину.

Архітектура комп'ютера розрізняє фізичний адресний простір (ФАП) і логічний адресний простір (ЛАП). *Фізичний адресний простір* являє собою простий одномірний масив байтів, доступ до якого реалізується апаратурою пам'яті за адресою, що присутня на шині адреси мікропроцесорної системи. *Логічний адресний простір* організується самим програмістом виходячи з конкретних потреб. Трансляцію логічних адрес у фізичні здійснює блок керування пам'яттю MMU [17].

В архітектурі сучасних мікропроцесорів ЛАП представляється у вигляді набору елементарних структур: байтів, сегментів і сторінок. У мікропроцесорах використовуються наступні варіанти організації *логічного адресного простору*:

- *плоский (лінійний) ЛАП*: складається з масиву байтів, що не має певної структури; трансляція адреси не потрібна, тому що логічна адреса збігається з фізичною;
- *сегментований ЛАП*: складається із сегментів – безперервних областей пам'яті, що містять у загальному випадку змінне число байтів; логічна адреса містить 2 частини: ідентифікатор сегмента й зсув усередині сегмента; трансляцію адреси проводить блок сегментації MMU;
- *сторінковий ЛАП*: складається зі сторінок – безперервних областей пам'яті, кожна з яких містить фіксоване число байтів. Логічна адреса складається з номера (ідентифікатора) сторінки й зсуву усередині сторінки; трансляція логічної адреси у фізичну проводиться блоком сторінкового перетворення MMU;
- *сегментно-сторінковий ЛАП*: складається із *сегментів*, які, у свою чергу, складаються зі сторінок; логічна адреса складається з ідентифікатора сегмента й зсуву усередині сегмента. Блок сегментного перетворення MMU здій-

снює трансляцію логічної адреси в номер сторінки й зсув у ній, які потім транслюються у фізичну адресу блоком сторінкового перетворення MMU.

Таким чином, основою одержання фізичної адреси пам'яті служить логічна адреса. У якійсь мірі логічний адресний простір, з яким має справу програміст, можна порівняти із структурою книги, де аналогом сегмента виступає розповідь, сторінка книги відповідає сторінці ЛАП, а шукана інформація – це деяке слово. При цьому якщо пам'ять організована як лінійна, то номер шуканого слова задається в явному вигляді й просто відраховується від початку книги. При сегментному представленні пам'яті шукане слово визначається його номером.

Сторінкове представлення пам'яті припускає завдання інформації про слово у вигляді номера сторінки в книзі й номера слова на зазначеній сторінці. При сегментно-сторінковому представленні логічна адреса слова задається номером слова в певній розповіді. У цьому випадку за змістом книги визначається номер сторінки, з якої починається зазначена розповідь. Потім, знаючи кількість слів на сторінці й положення слова в розповіді, можна розрахувати сторінку книги й положення шуканого слова на цій сторінці.

Формування фізичної адреси в універсальному мікропроцесорі при різних режимах роботи

Мікропроцесор здатний працювати у двох режимах: реальному й захищеному.

При роботі в *реальному режимі* можливості процесора обмежені: ємність пам'яті, що адресується, становить 1 Мбайт, відсутня сторінкова організація пам'яті, *сегменти* мають фіксовану довжину 216 байт.

Цей режим звичайно використовується на початковому етапі завантаження комп'ютера для переходу в *захищений режим*.

У *реальному режимі* сегментні регістри процесора містять старші 16 біт фізичної адреси початку *сегмента*. Зміщений на 4 розряди ліворуч *селектор* дає 20-розрядну базову адресу сегмента.

Фізична адреса виходить шляхом додавання цієї адреси з 16-розрядним значенням зсуву в сегменті, сформованого за заданим режимом адресації для операнда або, що вилучається з регістру EIP для команди (рис.13.5). За отриманою адресою відбувається вибірка інформації з пам'яті [2].

Найбільш повно можливості мікропроцесора з адресації пам'яті реалізуються при роботі в *захищеному режимі*. Обсяг пам'яті, що адресується, збільшується до 4 Гбайт, з'являється можливість сторінкового режиму адресації. *Сегменти* можуть мати змінну довжину від 1 байта до 4 Гбайт. Загальна схема формування фізичної адреси мікропроцесором, що працює у *захищеному режимі*, наведено на рис. 13.6. Як ми вже відзначали, основою формування фізичної адреси є логічна адреса. Вона складається із двох частин: *селектора* й *зсуву в сегменті*.

Селектор міститься в сегментному регістрі мікропроцесора й дозволяє знайти опис сегмента (*дескриптор*) у спеціальній таблиці дескрипторів. *Дескриптори* сегментів зберігаються в спеціальних системних об'єктах глобальній (GDT) і локальній (LDT) таблицях дескрипторів. *Дескриптор* відіграє дуже важливу роль у функціонуванні мікропроцесора, від формування фізичної адреси при різній організації адресного простору й до організації мультипрограмного режиму роботи. Тому розглянемо його структуру більш докладно.

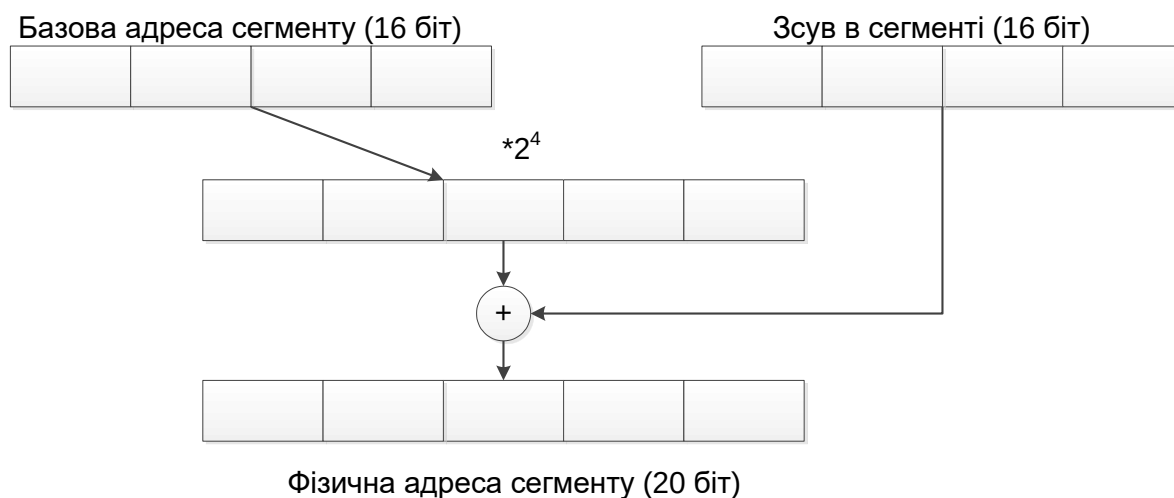


Рис. 13.5. Схема одержання фізичної адреси

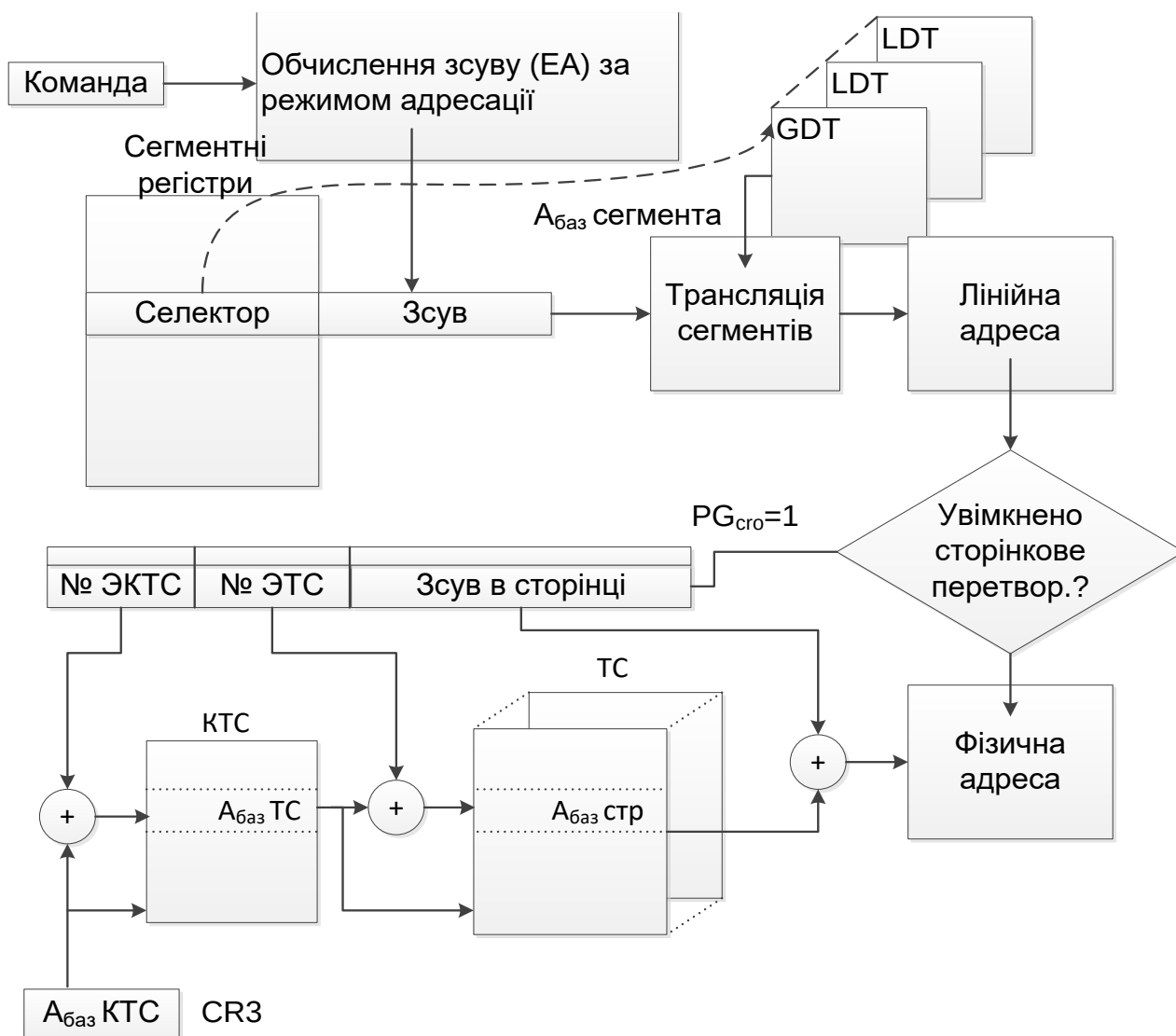


Рис. 13.6. Формування фізичної адреси при сегментно-сторінковій організації пам'яті

Сегменти мікропроцесора, що працює в захищеному режимі, характеризуються більшою кількістю параметрів.

Тому в універсальних 32-розрядних мікропроцесорах інформація про сегмент зберігається в спеціальній 8-байтній структурі даних, що називається *дескриптором*, а за сегментними реєстрами закріплена основна функція – визначення місця розташування дескриптора.

Ми будемо розглядати саме структуру, а не формат дескриптора, тому що при переході від мікропроцесора i286 до 32-розрядного МП розташування окремих полів дескриптора втратило свою стрункість і частково стало мати ви-

гляд «заплаток», поставлених з метою механічного збільшення розрядності цих полів. 32-розрядне поле базової адреси дозволяє визначити початкову адресу сегмента в будь-якій точці адресного простору в 2³² байти (4 Гбайт).

Поле межі (limit) вказує довжину сегмента (точніше, довжину сегмента мінус 1: якщо в цьому полі записано 0, то це означає, що сегмент має довжину 1) в одиницях, що адресуються, тобто максимальний розмір сегмента дорівнює 220 елементів. Величина елемента визначається одним з атрибутів дескриптора бітом G (Granularity - гранулярність, або дрібність):

Структуру дескриптора сегмента наведено на рис. 13.7. Таким чином, сегмент може мати розмір з точністю до 1 байта в діапазоні від 1 байта до 1 Мбайт (при G = 0). При обсязі сторінки в 2¹² = 4 Кбайт можна задати обсяг сегмента до 4 Гбайт (при G = 1):

$$V_{\text{сегм. макс}} = 2^{20}_{\text{стр}} \times 2^{12}_{\text{байт}} = 2^{32} \text{ байт}$$

$$G = \begin{cases} 0 - \text{довжина в байтах} \\ 1 - \text{довжина в сторінках} \end{cases}$$



Рис. 13.7. Структура дескриптора сегмента

Тому що в архітектурі IA-32 сегмент може починатися в довільній точці адресного простору й мати довільну довжину, сегменти в пам'яті можуть частково або повністю перекриватися.

Біт розмірності (Default size) визначає довжину адрес і операндів, використовуваних у команді за замовчуванням:

$$G = \begin{cases} 0 - 16 & \text{розрядів} \\ 1 - & 32 \text{ розряди} \end{cases}$$

Звичайно, цей біт призначений не для звичайного користувача, а для системного програміста, що застосовує його, наприклад, для оцінки сегментів для збору «сміття» або сегментів, базові адреси яких не можна модифікувати. Цей біт доступний тільки програмам, що працюють на вищому рівні привілеїв. Мікропроцесор у своїй роботі його не змінює й не використовує.

Байт доступу визначає основні правила поводження із сегментом.

Біт присутності *P* (Present) показує можливість доступу до сегмента. Операційна система (ОС) відзначає сегмент, переданий з оперативної в зовнішню пам'ять, як тимчасово відсутній, установлюючи в його дескрипторі $P = 0$. При $P = 1$ сегмент перебуває у фізичній пам'яті.

Коли вибирається дескриптор з $P = 0$ (сегмент відсутній в ОЗП), поля базової адреси й межі ігноруються. Це природно: наприклад, як може йти мова про базову адресу сегмента, якщо самого сегмента взагалі немає в оперативній пам'яті? У цій ситуації процесор відкидає всі наступні спроби використовувати *дескриптор* у командах, і адресний простір, який визначається *дескриптором*, як би «зникає» [1].

Виникає особливий випадок не присутності сегмента. При цьому операційна система копіює сегмент, що запитується, із диска до пам'яті (при цьому, можливо, видаляючи інший сегмент), завантажує в *дескриптор* базову адресу сегменту, установлює $P = 1$ і здійснює рестарт тієї команди, яка звернулася до відсутнього сегменту ОЗП.

Дворозрядне поле *DPL* (Descriptor Privilege Level) вказує один із чотирьох можливих (від 0 до 3) *рівнів привілеїв дескриптора*, що визначає можливість доступу до сегмента з боку тих або інших програм (рівень 0 відповідає найвищому рівню привілеїв).

Біт звернення A (Accessed) встановлюється в «1» при будь-якому звертанні до сегмента. Використовується операційною системою для того, щоб відслідковувати сегменти, до яких довше всього не було звертань.

Нехай, наприклад, 1 раз у секунду операційна система в дескрипторах усіх сегментів скидає біт A. Якщо по закінченню деякого часу необхідно завантажити в оперативну пам'ять новий сегмент, місця для якого недостатньо, операційна система визначає «кандидатів» на те, щоб очистити частину оперативної пам'яті, серед тих сегментів, у дескрипторах яких біт A до цього моменту не був установлений в «1», тобто до яких не було звертань за останній час.

4	3	2	1	0	
1	0	C	R	A	Сегмент коду
1	0	ED	W	A	Сегмент даних
0		Т и п			Системний об'єкт

Рис. 13.8. Формат поля типу байта доступу

Поле типу в байті доступу визначає призначення й особливості використання сегмента. Якщо біт S (System – біт 4 байта доступу) дорівнює 1, то даний дескриптор описує реальний сегмент пам'яті. Якщо S = 0, то цей дескриптор описує спеціальний системний об'єкт, який може й не бути сегментом пам'яті, наприклад, шлюз виклику, що використовується при перемиканні завдань, або дескриптор локальної таблиці дескрипторів LDT. Призначення бітів <3...0> байта доступу визначається типом сегмента (рис. 13.8).

У сегменті коду: біт підпорядкування, або узгодження, C (Conforming) визначає додаткові правила звернення, які забезпечують захист сегментів програм. При C = 1 даний сегмент є підлеглим сегментом коду. У цьому випадку він навмисно втрачає захист по привілеям.

Такий спосіб зручний для організації, наприклад, підпрограм, які повинні бути доступні всім завданням, що виконуються в системі. При C = 0 - це звичайний сегмент коду; біт зчитування R (Readable) встановлює, чи можна звер-

татися до сегмента тільки на виконання або на виконання й зчитування, наприклад, констант як даних за допомогою префікса заміни сегмента. При $R = 0$ допускається тільки вибірка із сегмента команд для їхнього виконання. При $R = 1$ дозволене також читання даних із сегмента.

Запис у сегмент коду заборонено. При будь-якій спробі запису виникає програмне переривання.

У сегменті даних:

- ED (Expand Down) – біт напрямку розширення. При $ED = 1$ цей сегмент є сегментом стека й зсув у сегменті повинен бути більше розміру сегмента. При $ED = 0$ – це сегмент власно даних (зсув повинний бути менше або дорівнювати розміру сегмента);

- біт дозволу запису W (Writeable). При $W = 1$ дозволена зміна сегмента. При $W = 0$ запис у сегмент заборонено, при спробі запису в сегмент виникає програмне переривання.

У випадку обігу за операндом *зсув у сегменті* формується мікропроцесором за режимом адресації операнда, заданого в команді. Зсув у сегменті коду вилучається з *регістру – покажчика команд* EIP.

Сума витягнутої з дескриптора початкової адреси сегмента й сформованого зсуву в сегменті дає *лінійну адресу* (ЛА).

Якщо в мікропроцесорі використовується тільки сегментне представлення адресного простору, то отримана лінійна адреса є також і фізичною [2].

Якщо крім сегментного використовується й сторінковий механізм організації пам'яті, то *лінійна адреса* представляється у вигляді двох полів: старші розряди містять номер віртуальної сторінки, а молодші зсув у сторінці. Перетворення номера віртуальної сторінки в номер фізичної проводиться за допомогою спеціальних системних таблиць: *каталогу таблиць сторінок* (КТС) і *таблиць сторінок* (ТС). Положення каталогу таблиць сторінок у пам'яті визначається системним регістром CR3. Фізична адреса обчислюється як сума отриманої з таблиці сторінок адреси фізичної сторінки й зсуву в сторінці, отриманого з лінійної адреси.

Розглянемо тепер усі етапи перетворення логічної адреси у фізичний більш докладно.

Структура коду команди й формування зсуву в сегменті

Як відзначалося вище, зсув у сегменті коду команд вилучається з регістру ЕІР і тому не вимагає додаткових пояснень.

Механізм же формування зсуву в сегменті даних проводиться на основі режиму адресації операнда й вимагає окремого вивчення.

Розглянемо спочатку структуру коду команди універсального 32-розрядного мікропроцесора. Команди в архітектурі ІА-32 мають велику різноманітність форматів, які залежать від типу операції, режимів адресації операндів, довжини використовуваних безпосередніх операндів і зсувів і ряду інших факторів. Вони мають довжину від 1 до 15 байт. Усе це суттєво ускладнює їхнє декодування в МП із даною архітектурою. На рис.13.9. наведено формат двооперандної команди загального виду.

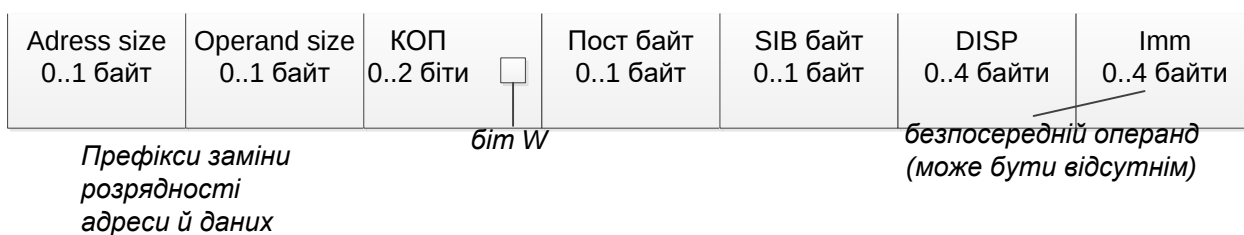


Рис. 13.9. Формат команди 32-розрядного мікропроцесора

Команда може починатися з декількох необов'язкових байтів (префіксів), які визначають особливості виконання команди.

Префікси розмірів довжини адреси й операндів дозволяють змінити їхні значення, установлені за замовчуванням *бітом розмірності D* у дескрипторі сегмента.

Для операндів разом з бітом W, що міститься в коді команди, префікс розміру дозволяє визначити операнд довжиною 8, 16 або 32 розрядів. Префікс розміру адреси визначає 16- або 32-розрядний зсув у сегменті (табл. 13.2).

У коді команди можуть використовуватися також додаткові байти для префікса заміни сегментного регістру, встановленого за замовчуванням, префікса повторення операції або префікса що запобігає переривання, операції переміщення даних [4].

Таблиця 13.2.

Використання префіксів перевизначення розмірів адрес і операндів

Біт розмірності D у дескрипторі сегмента	0	0	0	0	1	1	1	1
Префікс розмірності операнда*	-	-	+	+	-	-	+	+
Префікс розмірності адреси*	-	+	-	+	-	+	-	+
Розрядність операнда (біт)**	16/8	16/8	32/8	32/8	32/8	32/8	16/8	16/8
Розрядність адреси (біт)	16	32	16	32	32	16	32	16

Примітки: * + – префікс присутній; - – префікс відсутній; ** w = 1/0

Поле КІП містить код виконуваної команди, а також біт W розмірності використовуваних операндів. Для команд, що застосовують безпосередній операнд, код операції може також займати також частину постбайту. Постбайт (рис. 13.10) визначає місце розташування операндів.

Основна частина команд мікропроцесора з архітектурою IA-32 дозволяє працювати тільки з одним операндом, що перебуває в оперативній пам'яті. Його режим адресації кодується полями md і r/m постбайту. Другий операнд або вилучається з *регістрів загального призначення* мікропроцесора (його номер вказується в полі reg постбайту), або кодується в поле Imm самої команди (безпосередній операнд).

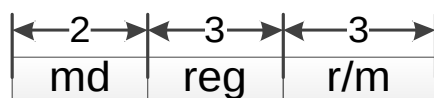


Рис. 13.10. Формат постбайту

Байт масштабованого індексу бази (SIB) служить для представлення складних структур пам'яті. На його наявність вказує код 100 у полі r/m постбайта. SIB-байт має наступну структуру (рис. 13.11):

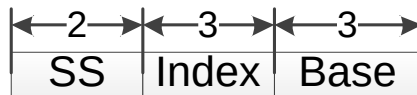


Рис. 13.11. Формат SIB-байта

Тут SS - поле масштабу, Index задає номер одного з РОН, використовуюваного в якості індексного регістру (регістр ESP не може бути регістром індексу). Поле Base у комбінації з полем mod постбайта вказує регістр бази й зсув для індексованих операндів.

Застосування SIB-байта дозволяє формувати зсув у сегменті, іноді його називають ефективною адресою (EA), для операндів таким способом:

$$(\text{зсув в сегменті}) = [base] + [index]^{ss} + disp,$$

де $[base]$ – значення базового регістру, $[index]$ – значення індексного регістру, ss – величина масштабного множника, $disp$ – значення зсуву, закодованого в самій команді.

У якості бази або індексу може бути використаний будь-який *регістр загального призначення* мікропроцесора. Величина індексу може бути помножена на масштабний коефіцієнт (1, 2, 4 або 8), що дає можливість посилання на елемент масиву або запис відповідної довжини.

Зсув кодується як величина зі знаком у додатковому коді. Його довжина визначається значенням біта **D** у *дескрипторі* сегмента, бітом w у першому байті команди й наявністю або відсутністю префікса розрядності адреси згідно табл. 13.2.

Цей механізм відбиває основні вдосконалення в способах адресації операндів для 32-розрядної архітектури IA-32 у порівнянні з архітектурою x86. Різні комбінації доданків у виразі дають наступні способи адресації пам'яті:

- пряма (тільки зсув);
- непряма (тільки база);

- базова відносна (база + зсув);
- індексна (індекс із масштабом);
- індексна із зсувом (індекс із масштабом + зсув);
- базова індексна (база + індекс із масштабом);
- відносна базова індексна (база + індекс із масштабом + зсув).

Головні особливості формату команд МП із архітектурою ІА-32 у порівнянні з 16-розрядним мікропроцесором:

- можливість використання кожного з *регістрів загального призначення* в кожному з режимів адресації;
- можливість використання 32-розрядних безпосередніх операндів і зсувів при відносних режимах адресації поряд з тими, що були раніше 8- і 16-розрядними;
- додавання ще одного режиму адресації - відносного базового індексного з масштабуванням.

Сегментна організація пам'яті в захищеному режимі

В основі сегментної моделі пам'яті лежить поділ її на незалежні адресні простори змінної довжини – *сегменти*. Для програми адресний простір розділений на блоки суміжних адрес, що називаються сегментами, а програма може звертатися тільки до даних, що перебувають у цих сегментах. У середині сегментів застосовується лінійна адресація, тобто програма може звертатися до байта 0, байта 1 і т.д.

Така адресація здійснюється відносно початку сегмента, і фізична адреса, асоційована, наприклад, із програмною адресою 0, по суті, схована від програміста. Цей підхід до керування пам'яттю опирається на той факт, що програми звичайно логічно розділяються на області (сегменти) коду, даних і стека. При цьому спрощується ізоляція програм одна від одної в мультипрограмному режимі роботи. Кожний *сегмент* має своє цільове призначення. Кожне завдання має безпосередній доступ до трьох основних сегментів: коду, даних і стека, обумовлених сегментними регістрами CS, DS SS відповідно, і до трьох додат-

кових сегментів даних, обумовлених сегментними регістрами ES, FS, GS. Опис цих сегментів утримується в їхніх *дескрипторах*.

Будь-яка програма, незалежно від рівня її привілеїв, не може звертатися до сегмента доти, поки він не описаний за допомогою дескриптора, а сам дескриптор не поміщений у таблицю дескрипторів [12].

Дескриптори зберігаються або в *глобальній таблиці дескрипторів (Global Descriptor Table - GDT)*, або в локальних таблицях дескрипторів (*Local Descriptor Table - LDT*). В *GDT* містяться дескриптори сегментів, які доступні всім активним завданням, наявним у системі на даний момент. *GDT* може містити будь-які дескриптори сегментів, за винятком дескрипторів переривань і пасток. Звичайно *GDT* включає дескриптори сегментів кодів і даних операційної системи, сегментів стану завдань і дескриптори сегментів, що містять локальні таблиці дескрипторів. Мікропроцесорна система має єдину глобальну таблицю дескрипторів.

Локальна таблиця дескрипторів LDT використовується для зберігання дескрипторів, доступних тільки даному завданню. Їхня кількість визначається кількістю активних завдань у системі [12].

З погляду розташування в пам'яті, локальні таблиці дескрипторів являють собою звичайні сегменти. Вони можуть накладатися один на одного, частково перетинатися. Це приводить до того, що окремі сегменти, описані дескрипторами у своїх *LDT*, можуть розділятися декількома завданнями (рис. 13.12).

Для знаходження *дескриптора* в таблиці дескрипторів використовується *селектор*, який міститься в одному із сегментних регістрів. *Селектор* являє собою 16-разрядне слово, яке розбито на 3 поля (рис. 13.13):

- *TI* (Table Indicator – *індикатор таблиці*) показує, до якої таблиці відбувається звертання: $TI = 0$ – дескриптор перебуває в глобальній таблиці дескрипторів *GDT*, $TI = 1$ – у локальній таблиці *LDT*;
- *Index: поле індексу* – номер дескриптора у відповідній таблиці дескрипторів;

– RPL (Request privilege level – *рівень привілеїв запиту*). При звертанні зрівнюється з полем DPL у байті доступу дескриптора.

Звертання дозволяється, якщо рівень привілеїв запиту не нижче, ніж рівень привілеїв дескриптора.

Максимальна кількість дескрипторів, що перебувають у таблиці дескрипторів, визначається довжиною поля Index селектора й дорівнює 213. Так як кожний дескриптор має довжину 8 байт, максимальний обсяг будь-якої таблиці дескрипторів становить 216 байт. Кожна з таблиць дескрипторів має регістр (GDTR для глобальної таблиці й LDTR для локальної), що визначає її положення в пам'яті.

Регістр GDTR містить 48 розрядів, з яких 32 задають базову адресу *глобальної таблиці дескрипторів*, а 16 вказують її обсяг у байтах (межу таблиці).

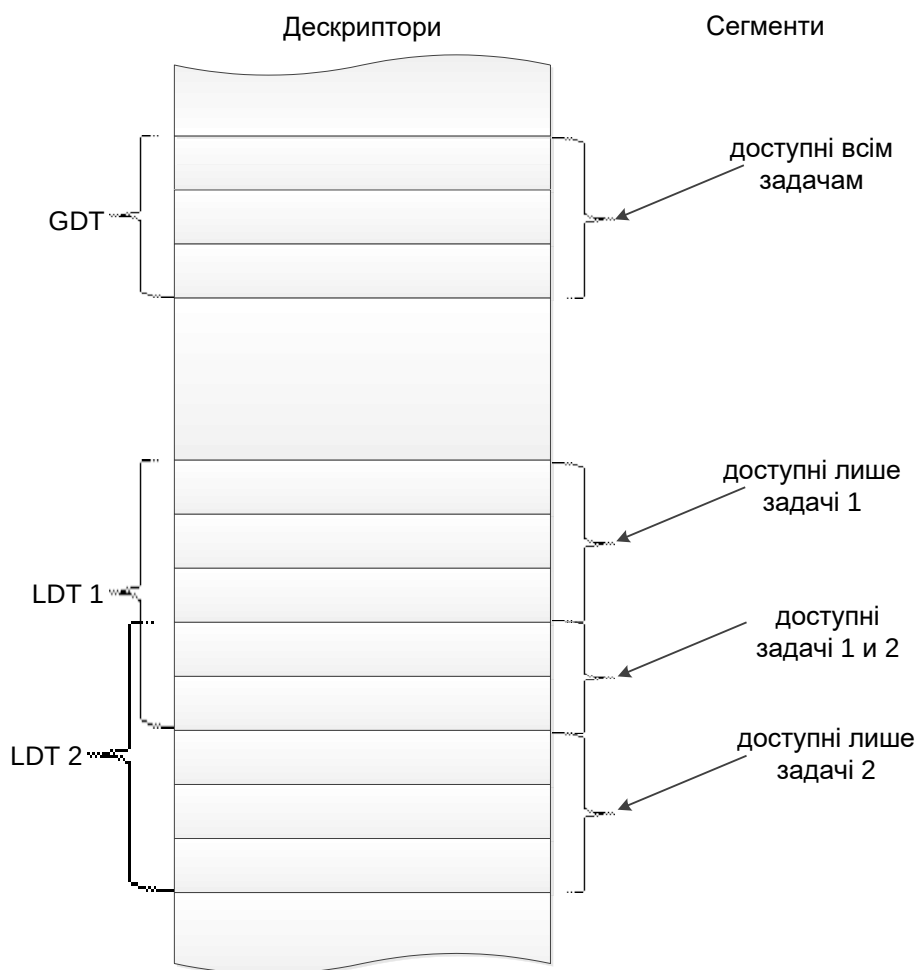


Рис. 13.12. Опис сегментів у таблицях дескрипторів

Для визначення положення *дескриптора* відносно початку таблиці його номер (поле Index селектора) помножується на 8, тобто реально зсувається на три розряди ліворуч, тому що довжина дескриптора становить 8 байт.

15	...	3	2	1	0
Index			TI	RPL	

Рис. 13.13. Формат селектора

Якщо *селектор* звертається до *дескриптора*, що міститься в таблиці GDT (при TI = 0 у селекторі), то отриманий зсув порівнюється з таблицею, що зберігається в GDTR між таблиці. Якщо порушення межі немає, то зсув додається до базової адреси, що міститься в GDTR, у результаті чого утворюється фізична адреса обраного дескриптора (рис. 13.14).

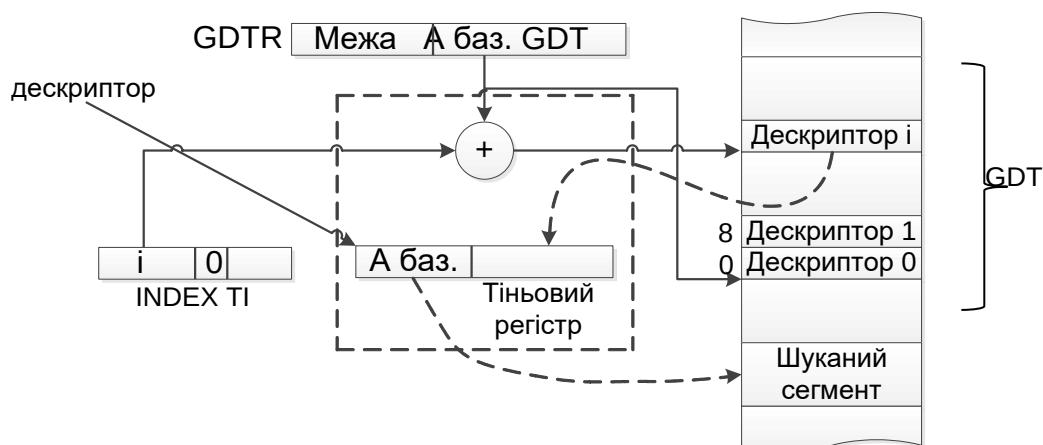


Рис. 13.14. Одержання дескриптора, що міститься в глобальній таблиці дескрипторів GDT

Нульовий дескриптор в GDT є порожнім, не використовуваним. Селектор з нульовим значенням розрядів 2...15 називається нуль-індикатором. Він забезпечує звертання до нульового дескриптора GDT.

Так як цей дескриптор не використовується, то при звертанні до нього відбувається переривання. Одне з можливих застосувань порожніх селекторів

полягає в наступному. Перед ініціюванням завдання операційна система може завантажити в регістри DS і ES порожні селектори.

Якщо надалі не ініціалізувати ці регістри, то адресація пам'яті через них викличе особливий випадок (переривання). Завантаження в LDTR порожнього селектора, для якого поле $\text{Index} = 0$ допустимо.

Така операція повідомляє процесор про те, що в завданні не буде використовуватися локальна дескрипторна таблиця. Це характерно для невеликих однокористувальницьких систем.

Для звертання до *локальної таблиці дескрипторів* призначений 16-розрядний регістр LDTR. Він містить *селектор*, що визначає розміщення в GDT дескриптора використовуваної локальної таблиці дескрипторів.

Така структура спрощує роботу з таблицями LDT. Завдяки опису LDT за допомогою селектора ці таблиці перетворюються у звичайні сегменти пам'яті й, зокрема, можуть розміщатися в будь-яких областях пам'яті, брати участь у свопінгу і т.п. У середині процесора з регістром LDTR асоціюється так званий «тіньовий регістр», у якому і зберігається дескриптор LDT поточного завдання. Це прискорює надалі звертання до локальної таблиці дескрипторів поточного завдання.

При перемиканні з одного завдання на інше для заміни використовуваної LDT досить завантажити в регістр LDTR селектор нової LDT, а процесор уже автоматично завантажить у тіньовий регістр дескриптор нової LDT при першому звертанні до нього [2].

Якщо в селекторі *індикатор таблиці* $TI = 1$, то дескриптор сегмента вибирається з *локальної таблиці дескрипторів*. Процес визначення адреси сегмента в цьому випадку наведено на рис. 13.15.

Він більш складний у порівнянні з одержанням дескриптора із глобальної таблиці дескрипторів і проходить наступні етапи [2]:

1. Аналізуємо, до якої із двох можливих таблиць дескрипторів відбувається звертання. Якщо в селекторі $TI = 1$, то звертання відбувається до локальної таблиці дескрипторів.

2. Знаходимо дескриптор локальної таблиці дескрипторів у глобальній таблиці дескрипторів.

3. Зчитуємо дескриптор LDT в «тіньовий» регістр регістру LDTR мікропроцесора. Після зчитування дескриптора в «тіньовий» регістр подальше звертання до сегмента аналогічно звертання до GDT, де замість GDTR використовуються поля базової адреси й межі з дескриптора LDTR, що перебуває в «тіньовому» регістрі.

4. За адресою локальної таблиці дескрипторів, що перебуває в її дескрипторі, і номером дескриптора з оброблюваного селектора знаходимо дескриптор сегмента в локальній таблиці дескрипторів.

5. Записуємо дескриптор сегмента в «тіньовий» регістр сегментного регістру мікропроцесора для прискорення наступних звертань до потрібного сегмента.

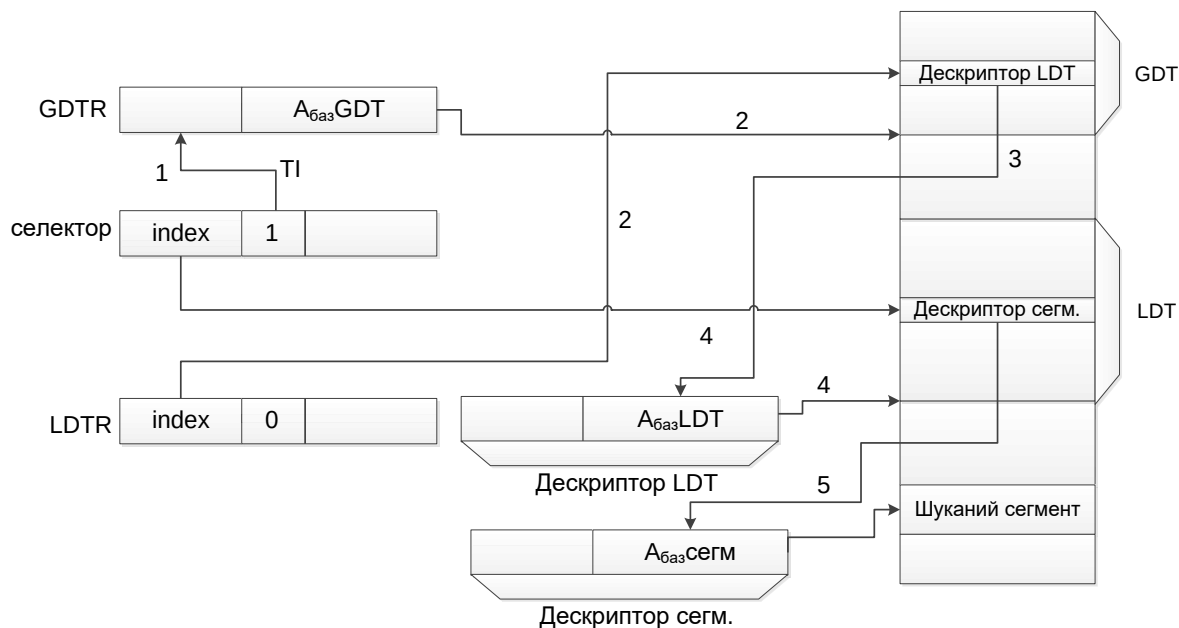


Рис. 13.15. Одержання дескриптора, що міститься в локальній таблиці дескрипторів LDT

Таким чином, при звертанні до сегмента через таблицю LDT з'являється додатковий рівень вкладеності, що знижує швидкодію мікропроцесора. «Тіньові» регістри мікропроцесора частково забезпечують розв'язок цієї проблеми.

Поле адреси дескриптора, отриманого з локальної або глобальної таблиці дескрипторів, визначає початок шуканого сегмента.

При підсумовуванні отриманої базової адреси сегмента й зсуву в сегменті виходить лінійна адреса шуканої комірки пам'яті.

У випадку якщо режим сторінкової адресації виключений (у реєстрі CR0 біт PG = 0), отримана лінійна адреса дорівнює фізичній адресі шуканого операнда або команди.

Розглянемо докладніше процес одержання адреси операнда на прикладі команди

MOV · EAX, [ECX + ESI + 20h] .

У цій команді немає спеціальних вказівок про використання сегмента, тому вона звертається до поточного сегмента даних, селектор якого за замовчуванням перебуває в сегментному реєстрі DS.

Нехай (DS) = 00000000000011.0.Xxb.

Формування фізичної адреси операнда включає наступні дії (для сегментованого ЛАП):

1. Утворити ефективну адресу (обчислити зсув у сегменті):

$$EA = (ECX) + (ESI) + 20h.$$

2. Вибрати 3-й дескриптор (Index = 3) з GDT (TI = 0).

Для цього:

– вважати базову адресу глобальної таблиці дескрипторів ($A_{\text{базGDT}}$) з GDTR;

– обчислити $A_{\text{базGDT}} + (\text{index}) \times 8$;

– звернутися за отриманою адресою до пам'яті й зчитати потрібний дескриптор.

3. Одержати лінійну адресу: $ЛА = EA + A_{\text{баздескр.3}}$, де $A_{\text{баздескр.3}}$ – базова адреса сегмента з зчитаного дескриптора з номером 3.

4. Так як при сегментній організації адресного простору *лінійна адреса* дорівнює фізичній, слід звернутися до пам'яті по сформованій адресі й передати подвійне *слово* в EAX.

При $TI = 1$ треба було б ще одне звертання до пам'яті для зчитування дескриптора LDT з GDT.

Щоб скоротити число звертань до пам'яті (а такий процес повинен проходити й при зчитуванні коду кожної команди), у мікропроцесорах з архітектурою IA-32 застосовується так зване кешування дескрипторів.

Кешування опирається на той факт, що звертання до пам'яті відбуваються набагато частіше, ніж зміна використовуваних *сегментів* і перемикання завдань.

Тому з кожним регістром, що містять *селектори* тих або інших сегментів (сегментні регістри, а також регістри *локальної таблиці дескрипторів* LDTR і регістр завдань TR), асоціюються «тіньові», або кеш-регістри [3].

При першому зчитуванні *дескриптора*, обумовленого даним *селектором*, процесор автоматично зчитує (кешує) потрібний дескриптор у відповідний «тіньовий» регістр.

Оскільки тепер дескриптор перебуває усередині МП, для одержання лінійної адреси пам'яті буде потрібно тільки сформувавши ефективну адресу й просумувати його з базовою адресою сегмента з потрібного «тіньового» регістру.

Так як програма звичайно рідко модифікує регістри із *селекторами*, у *захисному режимі* вона буде виконуватися приблизно з такою ж швидкістю, як і в реальному режимі.

Крім локальної й глобальної таблиць дескрипторів у мікропроцесорній системі використовується також *дескрипторна таблиця переривань* (IDT). Вона містить дескриптори спеціальних системних об'єктів, які визначають точки входу в процедури обробки переривань.

IDT служить заміною таблиці векторів переривань 16-розрядного мікропроцесора. Звертання до неї відбувається тільки апаратними засобами МП при

виникненні апаратних переривань або особливих випадків при виконанні програми.

Програми самостійно не можуть звернутися до IDT, тому що єдиний біт *індикатору таблиці в селекторі* сегмента ідентифікує тільки GDT або LDT.

До переводу процесора в *захищений режим* необхідно створити таблиці GDT і IDT і відповідно ініціалізувати регістри GDTR і IDTR. Таблиці GDT і IDT визначаються при завантаженні у відповідні регістри GDTR і IDTR базової адреси й межі.

Ця дія здійснюється тільки один раз у ході підготовки до переходу в *захищений режим*, і надалі вміст GDTR і IDTR не змінюється. Це значить, що місцезнаходження таблиць GDT і IDT у відомому змісті фіксоване, і вони не можуть брати участь у свопінгу [4].

Сторінкова організація пам'яті

Сторінкова організація пам'яті застосовується тільки в *захищеному режимі*, якщо в регістрі керування CR0 біт PG = 1.

Основне застосування сторінкового перетворення адреси пов'язане з реалізацією віртуальної пам'яті, яка дозволяє програмісту використовувати більший простір пам'яті, ніж фізична основна пам'ять.

Принцип *віртуальної пам'яті* припускає, що користувач при підготовці своєї програми має справу не з фізичною ОП, що дійсно працює в складі комп'ютера, що має деяку фіксовану ємність, а з віртуальною (що здається) однорівневою пам'яттю, обсяг якої дорівнює всьому адресному простору, обумовленому розміром адресної шини ($L_{ша}$) комп'ютера:

$$V_{virt} \gg V_{fiz}, V_{virt} = 2^{L_{ша}},$$

Для 32-розрядного мікропроцесора:

$$V_{virt} = 2^{32} = 4\text{Гбайт}$$

Програміст має у своєму розпорядженні адресний простір, обмежений лише розрядністю адресної шини, незалежно від реального обсягу оперативної

пам'яті комп'ютера й обсягів пам'яті, які використовуються іншими програмами, паралельно оброблюваними в мультипрограмній ЕОМ.

Віртуальна пам'ять, забезпечуючи можливість програмісту звертатися до дуже великого обсягу безперервного адресного простору, надаваного в його монопольне розпорядження, має звичайні властивості: побайтову адресацію, час доступу, порівняне з часом доступу до оперативної пам'яті.

На всіх етапах підготовки програм, включаючи завантаження до пам'яті, програма представляється у віртуальних адресах, і лише при виконанні машинної команди віртуальні адреси перетворюються у фізичні.

Для кожної програми, що виконується в мультипрограмному режимі, створюється своя віртуальна пам'ять. Кожна програма використовує ті самі віртуальні адреси від нульового до максимально великого в даній архітектурі.

Для перетворення віртуальних адрес у фізичні фізична й віртуальна пам'ять розбиваються на блоки фіксованої довжини, що називаються *сторінками*. Обсяги віртуальної й фізичної сторінок збігаються. Сторінки віртуальної й фізичної пам'яті нумеруються. Відсутні у фізичній пам'яті сторінки звичайно зберігаються в зовнішній пам'яті. Фіксований розмір усіх сторінок дозволяє завантажити будь-яку потрібну віртуальну сторінку в будь-яку фізичну [18].

Як відзначалося вище, при сторінковому представленні пам'яті віртуальна (логічна) адреса являє собою номер віртуальної сторінки й зсув усередині цієї сторінки. У свою чергу, фізична адреса - це номер фізичної сторінки й зсув у ній.

Правила перекладу номерів віртуальних сторінок у номери фізичних сторінок звичайно задаються у вигляді таблиці сторінкового перетворення. Такі таблиці формуються системою керування пам'яттю й модифікуються щораз при перерозподілі пам'яті. Операційна система постійно відслідковує стан віртуальних сторінок тієї або іншої програми й визначає, чи перебуває вона в оперативній пам'яті, і якщо перебуває, то в якому конкретно місці. Прикладні програми не торкаються процесу сторінкового перетворення адреси й можуть використовувати весь адресний простір. Процесор автоматично формує особливий випа-

док не присутності, коли програма звертається до сторінки, відсутньої у фізичній пам'яті.

При обробці цього особливого випадку ОС завантажує потрібну сторінку із зовнішньої пам'яті, при необхідності відправляючи деяку іншу сторінку на диск (процес свопінга).

Переклад віртуальних адрес у фізичні наведено на рис. 13.16.

Розглянемо приклад перетворення адреси віртуальної сторінки на адресу фізичної сторінки. Нехай комп'ютер використовує адресний простір, що припускає розбивку на сторінки обсягом $V_{\text{стр}} = 1\text{I}$, і має оперативну пам'ять $V_{\text{ОЗУ}} = 3$ сторінки.

Нехай на комп'ютері одночасно виконуються чотири програми, що мають наступну кількість сторінок: $V_A = 2$, $V_B = 1$, $V_C = 3$, $V_D = 2$. Перемикання між програмами відбувається через час кванта $t_k = 1$. Час виконання кожної сторінки будь-якої програми становить $t = 2t_k$. Вважаємо, що сторінки програм завантажуються в оперативну пам'ять по мірі необхідності й по можливості у вільні області ОЗП.

Якщо вся пам'ять зайнята, то нова сторінка заміщає ту, до якої довше всього не було звертань.

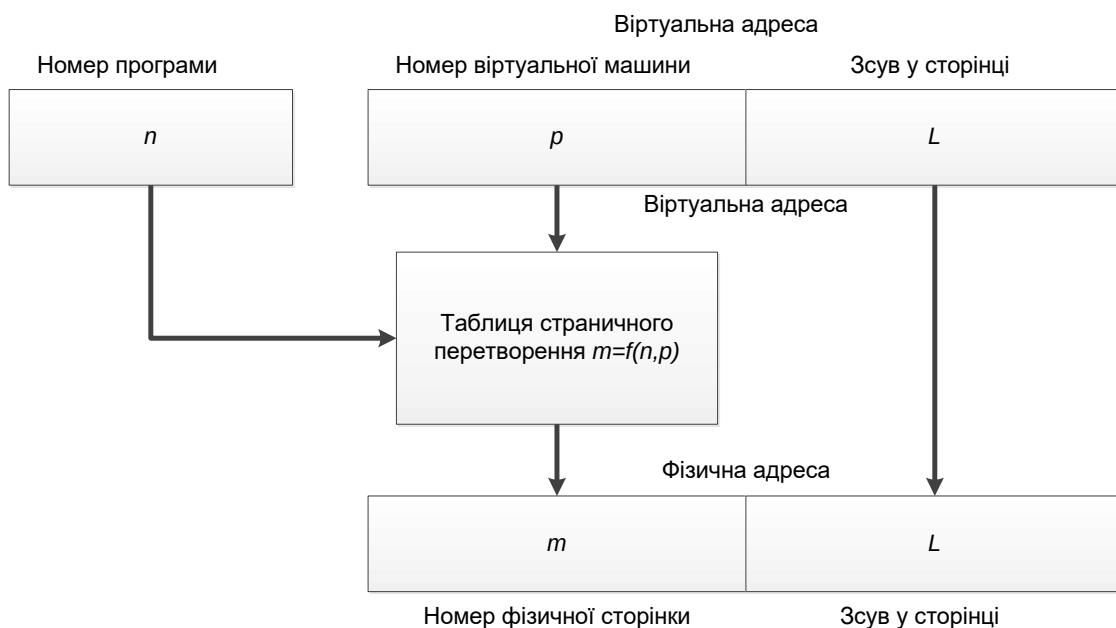


Рис. 13.16. Принцип перетворення віртуальної сторінкової адреси у фізичну

За таких умов таблиця завантаження оперативної пам'яті й таблиці сторінкового перетворення для кожної програми будуть мати вигляд, представлений у табл. 13.3.

Таблиця 13.3.

Таблиця завантаження оперативної пам'яті. Таблиці сторінкового перетворення для кожної програми

Сторінка	Такти															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Динамічний розподіл оперативної пам'яті															
ОЗП 0	A 0	A 0	A 0	D 0	D 0	D 0	C 0	C 0	C 0	C 1	C 1	C 1	C 1	C 1	C 1	C 1
1		B 0	B 0	B 0	A 0	A 0	A 0	D 0	D 0	D 0	D 1	D 1	D 1	D 1	D 1	D 1
2			C 0	C 0	C 0	B 0	B 0	B 0	A 1	A 1	A 1	A 1	A 1	A 1	C 2	C 2
Таблиця сторінкового перетворення для програми А																
A 0	0	0	-	-	1	1	1	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	2	2	2	2	2	2	-	-
Таблиця сторінкового перетворення для програми В																
B 0	-	1	1	1	-	2	2	2	-	-	-	-	-	-	-	-
Таблиця сторінкового перетворення для програми С																
3 0	-	-	2	2	2	-	0	0	0	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2
Таблиця сторінкового перетворення для програми D																
D 0	-	-	-	0	0	0	-	1	1	1	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	1	1	1	1	1	1

У таблиці розподілу оперативної пам'яті виділені номери активних у даному такті сторінок. У таблицях сторінкового перетворення прочерками відзначені ситуації, коли дана віртуальна сторінка відсутня в оперативній пам'яті [2].

Якщо кожна сторінка має обсяг 1000 комірок, що адресуються, то, наприклад, у такті 9 звертання по віртуальній адресі 1100 програми А (віртуальна

сторінка 1, зсув у сторінці дорівнює 100) приведе до звертання по фізичній адресі 2100 (фізична сторінка 2, зсув у фізичній сторінці такий ж, як і у віртуальній, тобто 100).

Розглянемо тепер застосування цих загальних принципів сторінкового перетворення адреси в мікропроцесорі з архітектурою IA-32 при обсязі сторінки в 4 Кбайт.

Основою сторінкового перетворення служить 32-розрядна *лінійна адреса*, отримана на етапі сегментного перетворення *логічної адреси*. Сторінкове перетворення виконується при значенні біта PG = 1 у керуючому регістрі CR0.

У цьому випадку старші 20 розрядів *лінійної адреси* фактично являють собою номер віртуальної сторінки.

Однак при прямому одноступінчастому перетворенні цього номера в номер фізичної сторінки необхідна таблиця з 220 елементів довжиною 4 байта кожний (20-розрядний номер сторінки плюс деяка додаткова інформація), тобто 4 Мбайт.

У мультипрограмному середовищі така таблиця може знадобитися для кожного завдання. Ця таблиця повинна постійно зберігатися в оперативній пам'яті, щоб суттєво не збільшувати час формування фізичної адреси.

Для цих цілей буде потрібно постійне резервування істотної частини обсяг ОЗП, що на етапі появи перших ЕОМ на основі МП із архітектурою IA-32 було практично неможливо.

Замість цього мікропроцесор використовує двоступінчасте сторінкове перетворення адреси. Коренева сторінка, що називається *каталогом таблиць сторінок* (КТС), містить 1024 32-розрядних *елемента каталогу таблиць сторінок* (ЕКТС - PDE page directory entry). Кожний з них адресує підлеглу *таблицю сторінок* (ТС), тобто всього допускається до 1024 підлеглих таблиць сторінок.

Кожна з таблиць сторінок містить 1024 32-розрядних *елемента таблиці сторінок* (ЕТС - PTE page table entry), кожний з яких і адресує фізичну сторінку. Таким чином, загальна кількість фізичних сторінок, що адресується, дорів-

нює 220, тобто весь віртуальний адресний простір (4 Кбайт * 220 елементів = 232 байт).

Кожна таблиця займає $1024 = 4$ Кбайт, тобто дорівнює 1 сторінці. Загальний обсяг таблиць, використовуваних для сторінкового перетворення, не зменшився, а навіть трохи зріс за рахунок використання каталогу таблиць сторінок. Однак, по-перше, практично завжди в системі цей розмір можна суттєво зменшити за рахунок того, що деякі *лінійні адреси* ніколи не будуть сформовані (а цю інформацію дають *таблиці дескрипторів сегментів*), і для них не потрібно створювати таблицю сторінок.

А по-друге, в оперативній пам'яті повинні постійно перебувати лише *каталог таблиць сторінок* і *таблиця сторінок* виконуваної в цей час програми. Інші *таблиці сторінок* можуть тимчасово зберігатися в зовнішній пам'яті [4].

Таким чином, перетворення лінійної адреси у фізичну має вигляд, який наведено на рис. 13.17.

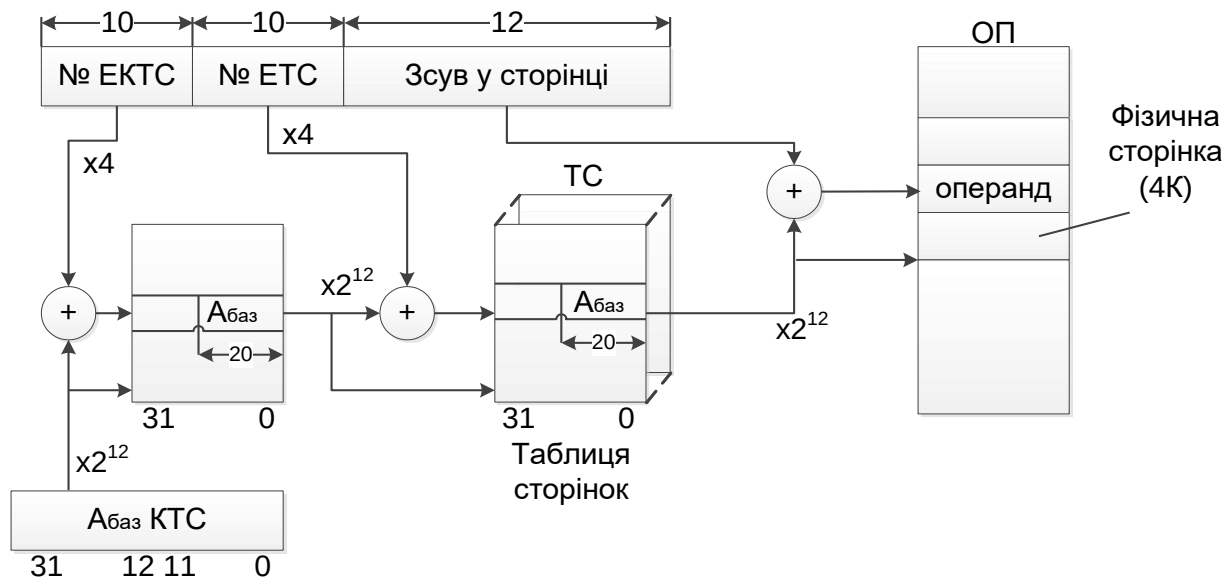


Рис. 13.17. Сторінкове перетворення лінійної адреси у фізичний

Старші 20 розрядів *лінійної адреси* розбиваються на два 10-розрядних поля: поле *номера елемента каталогу таблиць сторінок* і поле *номера елемента таблиці сторінок*.

Так як і каталог таблиць сторінок, і кожна таблиця сторінок займають рівно 1 сторінку й вирівняні по межі сторінки, то молодші 12 розрядів їх базової адреси дорівнюють нулю, і для визначення їх фізичної адреси достатньо 20-розрядного поля.

Для каталогу таблиць сторінок його 20-розрядна адреса перебуває в реєстрі керування CR3. КТС постійно перебуває в пам'яті й не бере участь у свопінгу [2].

Старші 20 розрядів фізичної адреси таблиці сторінок вилучається з ЕТС. Структури елемента КТС і елемента ТС схожі (рис. 13.18).

31...12	11 10 9	8 7	6	5	4	3	2	1	0
A _{баз}	Резерв ОС	0 0	D	A	PCD	PWT	U/S	R/W	P

Рис. 13.18. Структура елементів каталогу таблиць сторінок і таблиці сторінок

Старші 20 розрядів елемента дають базова адреса таблиці сторінок (в ЕКТС) або фізичної сторінки (в ЕТС). Біти P, A, R/W і U/S мають певну подібність із аналогічними атрибутами дескриптора сегмента, інші біти мають специфічне призначення.

Біт присутності P показує, чи відображається адреса сторінкового кадру (таблиці сторінок або сторінки пам'яті) на сторінку у фізичній пам'яті. При P = 1 сторінка присутня в ОЗП. При P = 0 сторінки в пам'яті немає, і звертання до цієї сторінки викликає переривання типу «сторінкове порушення».

Біт доступу A встановлюється мікропроцесором у стан A = 1 при звертанні до даного сторінкового кадру для запису або читання інформації.

Біт модифікації D (Dirty – "брудний") встановлюється процесором рівним 1 в елементі ЕТС при записі на дану сторінку. Для елементів каталогу таблиць сторінок значення біта D є невизначеним. При завантаженні сторінки на згадку операційна система скидає біт D. Якщо при необхідності вивантаження

сторінки в зовнішню пам'ять виявляється, що для неї $D = 0$, це означає, що до сторінки в пам'яті не було звертань до запису, у зовнішній пам'яті є її точна копія, і реально передавати сторінку з пам'яті на диск не потрібно. Тим самим заощаджується час при свопінгу.

Біт читання-запису R/W і біт U/S (user/supervisor – користувач/супервізор) визначають права доступу до таблиці сторінок або до сторінки для програм з різними рівнями привілеїв. Для сторінок існує тільки 2 рівня привілеїв: рівень супервізора (U/S = 0), відповідний до значення DPL сегмента 0, 1, 2, і рівень користувача (U/S = 1), що відповідає DPL = 3. Якщо до сторінки здійснюється запит з рівнем привілеїв 3 (програми користувача), то при значенні U/S = 0 йому забороняється доступ до відповідної до таблиці або сторінки. Якщо U/S = 1, то при значенні R/W = 0 дозволяється тільки читання таблиці або сторінки, а при R/W = 1 - і читання, і запис.

При запитах з більшими привілеями (системні програмні рівні 0, 1, 2) допускається запис і читання таблиць і сторінок при будь-яких значеннях U/S, R/W (табл. 13.4).

Таблиця 13.4.

Припустимі дії із сторінками на різних рівнях привілеїв

U/S	R/W	Припустимо для рівня 3	Припустимо для рівнів 0, 1, 2
0	X	Нічого	Читання/запис
1	0	Читання	Читання/запис
1	1	Читання/запис	Читання/запис

Біти PWT і PCD використовуються для керування роботою кеш-пам'яті при сторінковій адресації. Біт PCD – заборона кешування сторінки. При PCD = 1 кешування заборонене. Біт PWT – біт зворотного запису сторінок. Визначає метод відновлення зовнішньої кеш-пам'яті (кеш 2-го рівня). При PWT = 1 – від-

новлення проводиться методом наскрізного запису (як для внутрішнього кешу), при $PWT = 0$ - методом зворотного запису.

Біти 9...11 в ЕКТС і ЕТС зарезервовані за операційною системою. Процесор ніколи не використовує й не змінює ці біти. Розроблювачі ОС можуть залучити ці біти для зберігання інформації про «старіння» сторінок, щоб визначати сторінки, що підлягають заміні із зовнішньої пам'яті, і для інших цілей.

Старші 10 розрядів *лінійної адреси* разом із змістом регістру керування CR3 визначають необхідний *елемент каталогу таблиць сторінок*. Наступні 10 розрядів лінійної адреси містять номер елемента в обраній таблиці сторінок.

Тому що й ЕКТС, і ЕТС мають довжину 4 байта, для одержання зсуву початку елемента відносно початку відповідної таблиці необхідно його номер помножити на 4.

Останні 12 розрядів *лінійної адреси* містять зсув у сторінці. Таким чином, сума зсуву в сторінці й базовій адресі сторінки, витягнутої з ЕТС, дає фізичну адресу шуканого байта [2].

Буфер асоціативної трансляції сторінкової адреси

Сторінкова організація пам'яті вимагає двох додаткових звертань до пам'яті: для зчитування ЕКТС і ЕТС. Щоб не погіршувати продуктивність процесора, у схемі керування сторінковим перетворенням адреси вбудований *буфер асоціативної трансляції сторінкової адреси* (Translation Lookaside Buffer – TLB).

Коли програма формує *лінійну адресу*, яка відображена на елементі РТЕ, що перебуває в TLB, перетворення виконується без додаткових звертань до пам'яті.

TLB являє собою пам'ять із асоціативною вибіркою, яка містить 20-розрядні базові адреси 32 сторінок. Кожна з базових адрес має свою ознаку (тег), у якості якої використовуються старші розряди *лінійної адреси*.

Програми не можуть керувати кешуванням елементів РТЕ. Диспетчер пам'яті MMU кешує кожний використовуваний елемент РТЕ до заповнення буфера. При заповненому TLB процесор може знайти сторінкову інформацію для

128 Кбайт фізичної пам'яті (32 сторінки по 4 Кбайт). При такій ємності TLB частка кеш-влучень становить у середньому 98 %.

TLB складається з модуля основної пам'яті, блоку LRU, використовуваного при її відновленні, і логіки обслуговування (рис. 13.19).

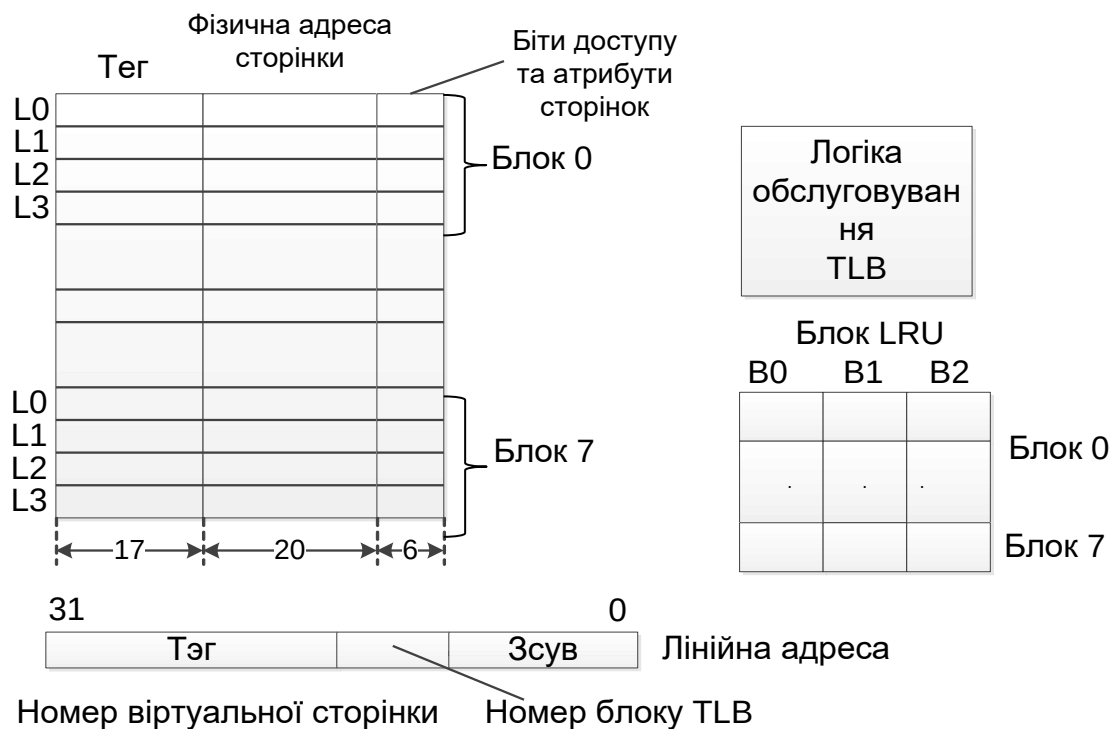


Рис. 13.19. Структура буфера TLB асоціативної трансляції сторінкової адреси

Основний модуль пам'яті містить 8 блоків, кожен з них містить інформацію про 4 сторінки, для яких раніше проводилося перетворення сторінкової адреси. Таким чином, основний модуль містить 32 рядка, що дозволяють безпосередньо, без звертання до КТС і таблиці сторінок, визначити базову фізичну адресу для однієї з 32 сторінок, параметри якої завантажені в TLB.

Кожний рядок TLB містить інформацію, необхідну для її вибору (тег, біти, що визначають доступ до сторінки), і інформацію про обрану сторінку (базова адреса, атрибути). Її структуру наведено на рис. 13.20.



Рис. 13.20. Формат рядка модуля основної пам'яті TLB

Поля базової адреси фізичної сторінки й атрибути D, U/S, R/W, PWT, PCD аналогічні відповідним до полів в ETC. Поле тега містить старші розряди лінійної адреси, для якої номер віртуальної сторінки перетворювався в номер фізичної сторінки. Біт V визначає вірогідність збереженої в даному рядку інформації ($V = 0$ – незаповнений рядок, $V = 1$ – достовірна інформація) [5].

Після формування *лінійної адреси* 3 молодших розряди поля номера віртуальної сторінки (біти 14...12 лінійної адреси) визначають номер одного з 8 блоків TLB. Старші 17 розрядів (біти 31...15) ЛА порівнюються одночасно асоціативним чином з 17 бітами тегів, що містяться в кожному з 4 рядків обраного блоку, з урахуванням біта ймовірності V кожного рядка.

Якщо для деякого рядка порівняння пройшло успішно, виходить, цей рядок містить інформацію із шуканої фізичної сторінки. Тобто для неї перетворення віртуальної адреси у фізичну вже виконувалося, і результат цього перетворення міститься в знайденому рядку. Зазначена у знайденому рядку базова адреса забезпечує вибірку потрібної фізичної сторінки.

Якщо збігу не було, то базова адреса потрібної сторінки відсутня в TLB, і перетворення сторінкової адреси проводиться звичайним шляхом зі зверненням до КТС і ТС. Отримана з таблиці сторінок інформація про фізичну сторінку разом з 17 старшими розрядами *лінійної адреси* (тег) заноситься в рядок одного із блоків TLB, номер якого задається бітами 14...12 лінійної адреси. Вибір одного з 4 рядків адресованого блоку, у який заноситься новий зміст (тег, базова адреса й ін.), визначається прийнятим механізмом заміщення [12].

Для TLB прийнятий механізм заміщення найбільше довго не використовуваного рядка (least recently used – LRU). При цьому вибір рядка, що заміщається, у блоці визначається бітами B0, B1, B2 (біти LRU), які зберігаються в додатковому модулі пам'яті TLB. Цей модуль містить 8 рядків по 3 розряду кожний. Рядок модуля відповідає одному із блоків основної пам'яті TLB. Логіка обслуговування TLB перевизначає біти рядка LRU по мірі звертання до відповідних рядків блоку TLB.

У будь-який момент комбінація розрядів $V_0 \dots V_2$ вказує, до якого з рядків даного блоку довше всього не було звертань. Саме цей рядок і заміщається при необхідності запису нового рядка, якщо всі 4 рядка блоку вже заповнені. Заповнювання рядка визначається значенням біта його ймовірності V .

При ініціалізації всі біти V_0, V_1, V_2 для всіх блоків скидаються в «0». У ході роботи біти V_0, V_1, V_2 приймають значення відповідно до алгоритму LRU у такий спосіб. Якщо останнє звертання у поточному блоці проводилося до рядків L_0 або L_1 , встановлюється $V_0 = 1$.

Якщо ж воно здійснювалося до L_3 або L_4 , то $V_0 = 0$. При перевірці пари рядків $L_0:L_1$ у випадку останнього звертання до L_0 встановлюється $V_1 = 1$, а якщо ні, то $V_1 = 0$. При перевірці пари рядків $L_2:L_3$ у випадку останнього звертання до L_2 встановлюється $V_2 = 1$, а якщо ні, то $V_2 = 0$ (табл. 13.5).

При пошуку підлягаючого заміні рядка в блоці, якщо всі 4 рядки достовірні, спочатку перевіряється, до якої з пар рядків $L_0:L_1$ або $L_2:L_3$ проводилося останнє звертання. Потім проводиться аналіз усередині пари, відібраної на попередньому етапі. Таким чином, заміна рядка в блоці TLB у випадку, коли вони всі достовірні, проводиться відповідно до табл. 13.6.

Таблиця 13.5.

Порядок зміни біт у рядку LRU

Біт LRU	Останнє звертання
$V_0 = 1$	L_0 або L_1
$V_0 = 0$	L_2 або L_3
$V_1 = 1$	L_0
$V_1 = 0$	L_1
$V_2 = 1$	L_2
$V_2 = 0$	L_3

В принципі, для того щоб задати номер одного з 4 рядків, досить 2-розрядного коду. Але при цьому механізм, що визначає рядок, до якого довше всього не було звертань, може виявитися досить складним. Тут же шляхом до-

давання всього лише одного біта одержуємо прозорий механізм, як по встановленню цих розрядів, так і по визначенню рядків, що підлягають заміні. Зберігання елементів таблиць сторінок в TLB несе у собі небезпеку, пов'язану з модифікацією ЕТС «на льоту», тобто в ході виконання програми. Припустимо, що в програмі передбачається вносити зміни в каталог таблиць сторінок або в таблиці сторінок.

Таблиця 13.6.

Порядок заміни рядків у блоці TLB

B0	B1	B2	Рядок, що замінюється
0	0	X	L0
0	1	X	L1
1	X	0	L2
1	X	1	L3

Після підготовки необхідних таблиць і дозволу сторінкового перетворення процесор буде завантажувати ЕТС в TLB до заповнення. Потім по мірі необхідності процесор замінює старі елементи новими.

Нехай тепер програма змінила деякий ЕТС, що зберігається в пам'яті. Якщо змінений елемент вже перебуває в TLB, процесор буде користуватися його старим значенням, тому що він не може довідатися про те, що цей елемент у пам'яті був модифікований.

Для подолання цієї колізії програмісту необхідно після всієї підготовки до зміни ЕТС відразу ж перезавантажити регістр CR3, наприклад, за допомогою такої пари команд:

MOV EAX, CR3

MOV CR3, EAX

При будь-якому перезавантаженні регістру CR3 всі біти ймовірності в TLB скидаються ($V = 0$), і МП буде змушено проводити нові перетворення сторінкової адреси для всіх сторінок, завантажуючи при цьому в TLB і модифікований ЕТС.

Аналогічні операції виконуються при відсутності сторінки в оперативній пам'яті ($P = 0$), але вже операційною системою.

Сторінкове порушення при відсутності сторінки в оперативній пам'яті викликає наступні дії:

1. ОС копіює запитану сторінку з диска в оперативну пам'ять;
2. ОС завантажує адреси сторінкового кадру в елемент таблиці сторінок або каталогу таблиць сторінок і встановлює $P = 1$. При цьому можуть бути встановлені й інші біти, наприклад, R/W;
3. Тому що в TLB може залишатися копія старого ЕКТС або ЕТС, ОС очищає його;
4. Здійснюється рестарт команди, що викликала сторінкове порушення.

Розширення обсягів оброблюваної інформації, особливо мультимедійної, разом зі збільшенням функціональних можливостей мікропроцесорів і мікропроцесорних систем у цілому, призвело до істотної зміни в параметрах сторінок. Сучасні мікропроцесори, зберігаючи головну особливість сторінкової організації, сталість обсягу сторінки, забезпечують можливість використання широкого діапазону їх розмірів.

Так, у мікропроцесорі Itanium на апаратному рівні підтримуються сторінки ємністю 4/8/16/64/256 Кбайт, 1/4/16/64/256 Мбайт.

Короткі підсумки. У розділі розглянуто фізичне й логічне представлення пам'яті. Показаний загальний механізм формування фізичної адреси при сегментно-сторінковому представленні адресного простору.

Розглянутий порядок формування адреси блоком сегментації й блоком сторінкового перетворення мікропроцесора.

Показані способи, що забезпечують скорочення часу такого перетворення.

13.2. Основні напрямки розвитку архітектури універсальних мікропроцесорів

Розвиток мікропроцесорної техніки в області універсальних мікропроцесорів прямує шляхом постійного підвищення їх продуктивності. Традиційними напрямками такого розвитку є підвищення тактової частоти роботи МП і збільшення кількості одночасно виконуваних команд за рахунок збільшення числа конвеєрів (виконавчих пристроїв) у МП.

Однак обидва ці напрямки слід визнати екстенсивними, що мають природні обмеження.

Підвищення тактової частоти, яке в основному забезпечується шляхом збільшення кількості щаблів у конвеєрі, приводить до більших втрат часу при необхідності перезавантаження конвеєра внаслідок конфліктів по керуванню або при перемиканні на нове завдання.

Таке збільшення має також і фізичні обмеження, зв'язані зі схмотехнікою кристала ВІС. Обмеження визначаються також впливом накладних витрат при передачі частково обробленої команди на наступний щабель конвеєра (значення t при визначенні тривалості такту).

На більших частотах ці витрати стають порівнянними із тривалістю обробки на черговому етапі. Багато в чому цей напрямок вичерпав себе в мікропроцесорі Pentium 4, що працює на частотах, близьких до 4 ГГц [3].

Підвищення продуктивності за рахунок збільшення числа конвеєрів у мікропроцесорі можна оцінити збільшенням числа команд, виконуваних програмами за такт (IPC - Instructions Per Cycle).

Так, для МП Alpha 21264 цей показник дорівнює 6, стільки ж мікрооперацій за такт може видати Pentium 4. Але це граничні значення, а реальні програмні коди, зокрема, через різні взаємозалежності, дають набагато нижче значення IPC.

Подальше збільшення числа виконавчих пристроїв веде до ускладнення розташованого в ВІС пристрою керування, що розподіляє команди по конвеє-

рах, а також до складних взаємозалежностей між даними. До того ж реальні коди програм не дозволяють забезпечити ефективне завантаження всіх наявних у МП виконавчих пристроїв, що приводить до їхніх простоїв. Слід зазначити також, що ріст продуктивності мікропроцесора не є прямо пропорційним росту кількості конвеєрів, а звичайно суттєво нижчим.

У цей час для підвищення продуктивності мікропроцесорів використовується ряд нових підходів, основними з яких є:

- *CMP (Chip Multi Processing)* – створення на одному кристалі системи з декількох мікропроцесорів (багатоядерність);
- *SMT (Simultaneous Multithreading)* – багатонитьова архітектура;
- *EPIC (Explicitly Parallel Instruction Computing)* – обчислення з явним паралелізмом у командах.

Розглянемо ці напрямки докладніше.

1. *Напрямок CMP* забезпечується збільшеними технологічними можливостями, які дозволяють створити на одному кристалі кілька мікропроцесорів і організувати їхню роботу із принципу мультипроцесорних систем.

Виробники чипів уже не женуться за частотою, змістивши акцент на багатоядерну архітектуру, яка дозволяє нарощувати продуктивність, зберігаючи в прийнятних границях енергоспоживання й тепловиділення.

Багатоядерні процесори добре пристосовані для вимогливих мультимедійних завдань, таких як обробка відеозаписів, роботи з більшими базами даних, одночасне виконання декількох ресурсномістких завдань, наприклад, комп'ютерної гри, запису DVD і завантаження файлів з Інтернету.

При такому підході завдання підвищення продуктивності роботи окремих додатків вимагає розпаралелювання останніх, тобто проблема переміщається з апаратного на програмний рівень. На даний момент складності полягають у тому, що більша частина існуючого програмного забезпечення створювалася без розрахунків на використання в багатоядерних і багатопроцесорних конфігураціях. Інакше кажучи, прогрес в області апаратних засобів на якийсь час випередив прогрес в області програмного забезпечення.

Розвиток мікропроцесорної техніки в цьому напрямку йде дуже швидкими темпами. Так, компанія Tiler в 2007 році почала поставки спеціалізованих процесорів Tile64, що нараховують 64 ядра.

2. *Напрямок SMT* у розвитку архітектури мікропроцесорів базується на тому, що одне завдання не в змозі повністю завантажити всі зростаючі ресурси мікропроцесора. Тому на одному процесорі здійснюється запуск декількох завдань одночасно, при цьому розпаралелювання програм здійснюється апаратними засобами МП.

Це дозволяє більш рівномірно завантажити ресурси процесора. Паралельно в різних пристроях МП можуть виконуватися команди з різних завдань. Так, мікропроцесор Alpha 21264 підтримує виконання до 4 завдань одночасно. За підтримки SMT на 4 нитки кожний процесор з погляду операційної системи виглядає як 4 логічних процесора. Дослідження показали, що SMT дозволяє збільшити продуктивність даного процесора до двох разів, а додаткові схеми керування займають усього близько 10 % площі кристала.

Деякі мікропроцесори для максимального підвищення своєї продуктивності використовують обидва вищезгаданих підходи. Так, компанія Sun Microsystems представила новий процесор Ultrasparc T2.

Новий чип оснащено вісьмома ядрами, кожне з яких може обробляти вісім потоків інструкцій. Таким чином, він одночасно здатний оперувати з 64 потоками. Тактова частота Ultrasparc T2 становить від 900 МГц до 1,4 ГГц.

3. *Напрямок EPIC* фактично використовує відому технологію VLIW (*Very Large Instruction Word*) – дуже довгого командного слова.

Розпаралелювання алгоритму між виконавчими модулями проводиться компілятором на етапі створення машинного коду, коли команди поєднуються у зв'язування й не конкурують між собою за ресурси мікропроцесора. При цьому спрощується блок керування на кристалі.

Особливості архітектури EPIC:

1. Явний паралелізм у машинному коді. Пошук залежностей між командами проводить не процесор, а компілятор.

2. Велика кількість реєстрів.

3. Масштабованість архітектури до великої кількості функціональних пристроїв (АЛП, FPU, MMX, SSE і т.п.).

4. Застосування предикатів. Предикатний підхід виходить із передумови, що збільшені потужності мікропроцесорів дозволяють запускати паралельно команди з різних розгалужень умовного розгалуження замість того, щоб очікувати формування дійсних ознак для вибору правильного напрямку або покладатися на блок передбачення переходів, ризикуючи прийти до необхідності перезавантаження досить довгих конвеєрів у випадку невдалого передбачення. При цьому кожна команда забезпечується спеціальним полем умови (предикатом) (рис. 6.3.1). З визначенням дійсних ознак розгалуження ті команди, предикати яких вказували, що вони обрані з іншої галузі, знімаються з обробки в конвеєрі. Результати команд не записуються до приймача до визначення правильності напрямку переходу.

Відзначимо *основні переваги цього підходу*:

1. Спрощується архітектура процесора. Замість логіки розпаралелювання на EPIC-процесорі можна розмістити більше реєстрів, функціональних пристроїв і т.п.

2. Процесор не витрачає час на аналіз потоку команд.

3. Можливості процесора по аналізу програми під час виконання обмежені порівняно невеликою ділянкою програми, тоді як компілятор здатний зробити аналіз усієї програми.

4. Якщо деяка програма повинна запускатися багаторазово (а саме так і буває в переважній більшості випадків), вигідніше ропаралелити її один раз при компіляції, а не витрачати на цей час щораз, коли вона виконується на процесорі.

Однак архітектурі EPIC властивий і ряд *недоліків* [2]:

1. Компілятор робить статичний аналіз програми, раз і назавжди плануючи обчислення. Однак навіть при невеликих змінах вихідних даних шлях виконання програми суттєво змінюється.

2. Значно ускладнюються компілятори, отже, збільшуються час компіляції програми й число помилок у самих компіляторах. Якщо перший фактор, враховуючи високу швидкодію сучасних комп'ютерів, не дуже важливий, то на другий слід звернути певну увагу. Дослідження показують, що до моменту встановлення навіть відповідального програмного забезпечення в ньому міститься приблизно 1 помилка на 10 000 рядків вихідного коду. Отже, програма з 500 тис. рядків буде містити близько 50 помилок, як би добре не працювали тестувальники. І ці помилки можуть виявитися найбільш несподіваними.

3. Продуктивність мікропроцесора багато в чому визначається якістю компілятора. Правда, тут необхідно чітко визначити, що розуміється під продуктивністю, адже кількість операцій, виконуваних мікропроцесором в одиницю часу, від компілятора не залежить. Але ця тема буде особливо розглянута далі.

4. Збільшується складність налагодження, тому що налагоджує не вихідна програма, а оптимізований паралельний код.

Програмісту важко визначити місце й причину появи помилки, тому що в процесі трансляції вихідної програми її окремі команди будуть переставлені компілятором для забезпечення оптимальності роботи мікропроцесора. Типовим представником архітектури EPIC є мікропроцесор Itanium фірми Intel.

Короткі підсумки. У розділі розглянуті класичні напрямки підвищення продуктивності мікропроцесорів, показані їх обмеження. Представлені сучасні шляхи розвитку універсальних мікропроцесорів.

КОНТРОЛЬНІ ПИТАННЯ

1. Які групи регістрів універсального 32-розрядного мікропроцесора вам відомі?
2. Наведіть класифікацію основних функціональних регістрів.
3. Назвіть шість 16-розрядних регістрів, з яких складається блок сегментних регістрів.
4. Наведіть класифікацію регістрів процесора з плаваючою крапкою.

5. Дайте визначення системним регістрам. Які групи регістрів входять до їх складу.
6. Що являють собою фізичний та логічний адресні простори? Які варіанти організації логічного адресного простору використовуються у мікропроцесорах?
7. Наведіть алгоритм формування фізичної адреси в універсальному мікропроцесорі при різних режимах роботи.
8. Які поля дескриптора вам відомі? Дайте визначення поняттям: поле межі, біт розмірності, байт доступу, біт присутності, біт звернення.
9. Розкрийте механізм формування зсуву в сегменті даних.
10. Що представляє собою локальна таблиця дескрипторів? Для чого вона використовується?
11. Для чого використовується селектор в таблиці дескрипторів? Які поля містить селектор?
12. Для чого призначений буфер асоціативної трансляції сторінкової адреси?
13. Які дії викликає сторінкове порушення при відсутності сторінки в оперативній пам'яті?
14. Чим відрізняється мікроархітектура Netburst від своїх попередників?
15. Для чого використовується блок передбачення розгалужень? З чого він складається?
16. Що вам відомо про технологію MMX?
17. Назвіть основні риси технології MMX.
18. Сформулюйте призначення команд ADDSUBPS, HADDPS, SSE2.
19. Які підходи використовуються для підвищення продуктивності мікропроцесорів?
20. Назвіть особливості EPIC-архітектури.

14. ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ

14.1. Паралельні структури обчислювальних систем

Два рівні розпаралелювання

Розвиток обчислювальної техніки характеризується тим, що на кожному етапі нових розробок вимоги до продуктивності значно перевищують можливості елементної бази.

Це обумовлене завданнями складних систем управління в реальному часі, централізованим розв'язком завдань у мережах, імітаційним моделюванням складних процесів (наприклад, у ядерній фізиці), оперативним плануванням і керуванням і розв'язком інших завдань дослідження операцій, що долають "прокляття розмірності". Такі завдання вимагають концентрації обчислювальних потужностей, постійно підтримуючи високу актуальність проблеми створення супер-ЕОМ [8].

Вже давно стало ясно, що тільки структурними методами можна зрівняти можливості обчислювальних засобів і необхідні швидкості розв'язку завдань на них. Під структурними розуміють методи розпаралелювання робіт. До розпаралелювання прибігають при проектуванні окремих пристроїв ЕОМ – пристроїв управління, буферів команд, каналів звертання до пам'яті й модулів пам'яті, багатofункціональних арифметико-логічних пристроїв (АЛП), повсюдно застосовуваних конвеєрів і т.д. Але до розпаралелювання однак прибігають і в проектуванні спільної роботи багатьох процесорів при паралельній або розподіленій обробці інформації, додаючи до обігу термін "обчислювальна система (ОС)".

Технічний прогрес, безсумнівно, позначається на рості частоти роботи елементної (елементно-конструкторської) бази, на підвищенні ступеня інтеграції, але завдяки цьому з'являються все нові завдання, що вимагають ще більш значного росту продуктивності обчислювальних засобів. Це можна вважати законом, що призводить до нових хитрощів при сполученні роботи пристроїв ОС,

при збільшенні їх кількості в системі, при збільшенні їх ефективності в процесі розв'язку завдань.

Під *ефективністю роботи пристрою* в складі ОС розуміють ступінь його участі в загальній роботі ОС при розв'язку конкретного завдання – коефіцієнт завантаження пристрою. Розпаралелювання робіт виправдане, якщо приводить до істотного росту усередненого по всіх пристроях коефіцієнта завантаження устаткування при розв'язку завдань. Це безпосередньо позначається на часі розв'язку. Сьогодні говорять не про спеціальний клас завдань, а про завдання, що орієнтують ОС на універсальність, що обумовлена сучасними областями застосування.

Важливим революціонізуючим моментом став перехід на мікропроцесорну елементно-конструкторську базу, що обумовило побудову мультимікропроцесорних ОС.

З'явилась представлення про два основних рівня, на яких у ОС застосовуються практичні методи розпаралелювання:

- на рівні програм, процесів, процедур (перший рівень розпаралелювання);
- на рівні команд і операцій (другий рівень розпаралелювання).

Ці рівні обумовили рівні структуризації ОС на шляху перетворення її в супер-ЕОМ. Сучасним практичним втіленням першого рівня структуризації є однорідні багатопроцесорні ОС на загальній (розподіленій) оперативній пам'яті. Вони одержали назву симетричних ОС через забезпечення "рівноправності" складових модулів.

Остаточне визнання симетричних ОС поклало кінець пошуку "екзотичних" архітектур, ефективних лише при розв'язку певних класів завдань. Універсальність симетричних ОС, можливість реалізації на них будь-яких обчислювальних процесів з високою ефективністю обладнання ілюструються багатьма програмами й аналізуються нижче.

Рівень команд і операцій найбільше яскраво представлено багатofункціональними АЛП і їх узагальненням – вирішальними полями, що представляють

розподілений обчислювальний ресурс багатьох процесорів. Деякі сучасні проекти ОС у різному ступені припускають такий ресурс. Отут основна проблема – повне завантаження окремих виконавчих пристроїв (ВП) у процесі виконання програми.

Розрізняють два способи реалізації такого завантаження: динамічний і статичний.

Динамічне завантаження здійснюється апаратурою в процесі виконання програми. Вона використовує спрощені алгоритми розпаралелювання.

Статичне завантаження передбачається при трансляції програми. Транслятор оптимізує використання обладнання, також вирішуючи завдання розпаралелювання. Це виражається у формуванні «довгих» командних слів, що задають роботи пристроям АЛП в кожному машинному такті.

Основна складність розпаралелювання полягає в дотриманні часткової впорядкованості робіт, що розподіляються. Тому розв'язок завдань синхронізації паралельного обчислювального процесу супроводжує всі зусилля по організації спільної роботи пристроїв. Це позначається й при розв'язку всіх завдань ефективного використання розширеної багаторівневої пам'яті ОС [5].

Вітчизняний досвід створення сімейства БОК (багатопроекторних обчислювальних комплексів) "Ельбрус", моделі якого відносяться до симетричних ОС, і наступне проектування дозволили проаналізувати, розробити й застосувати ряд суттєво нових, важливих і перспективних розв'язків у розпаралелюванні як самого обчислювального процесу, так і роботи окремих пристроїв. Розробка пронизана такими розв'язками, вони лягли в основу проектування розвитку сімейства, є основою узагальнень і подальшого дослідження можливості застосування.

Класифікація паралельних ОС

Потоки команд і потоки даних

Загальноприйнята вдала класифікація ОС, яку запропонував в 1966 р. М.Флінн (США). Основним визначальним архітектурним параметром він вибрав взаємодію *потіку команд та потоку даних* (операндів і результатів) [5].

В ЕОМ класичної архітектури ведеться послідовна обробка команд і даних. Команди надходять одна за іншою (за винятком крапок розгалуження програми), і для них з ОЗП або регістрів так само послідовно надходять операнди.

Одній команді (операції) відповідає один необхідний їй набір операндів (як правило, два для бінарних операцій). Цей тип архітектури – "*один потік команд – один потік даних*", *ОКОД (SISD - "Single instruction, Single data")* умовно зображено на рис. 14.1.

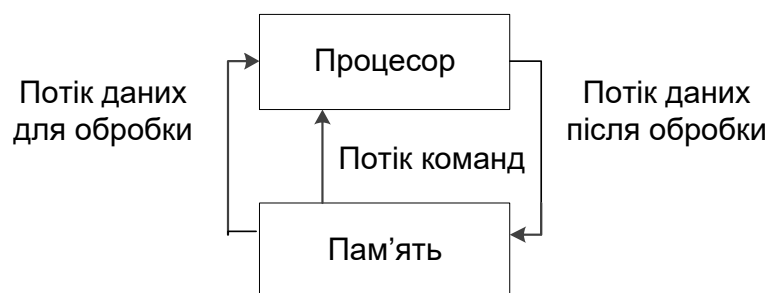


Рис. 14.1. ОС типу ОКОД (SISD)

Тип *ОКМД* – "*один потік команд – багато потоків даних*" (*SIMD - "Single instruction – Multiple Data"*) охоплює ОС, у яких однією командою обробляється набір даних, множина даних, вектор, і виробляється множина результатів. Це векторні й матричні системи, у яких по одній команді виконується та сама операція над усіма елементами масиву – вектора або матриці, розподіленими міжпроцесорними (обробними) елементами ПЕ або процесорами. Принцип обробки наведено на рис. 14.2.

Вітчизняні векторні ОС – ПС-2000, ПС-2100. Допускають організацію матричної обробки. Класичний приклад матричної архітектури – ILLIAC-1V (США).

До типу *БКОД* – "багато потоків команд – один потік даних" (*MISD* – "Multiple instruction – Single data") прийнято відносити векторний конвеєр (звичайно в складі ОС, щоб підкреслити основний використовуваний принцип обчислень), наприклад, у складі ОС *Crey-1*, "Електроніка ССБІС". На векторному конвеєрі проводиться послідовна обробка одного потоку даних багатьма обробними пристроями (ступенями, станціями) конвеєра.

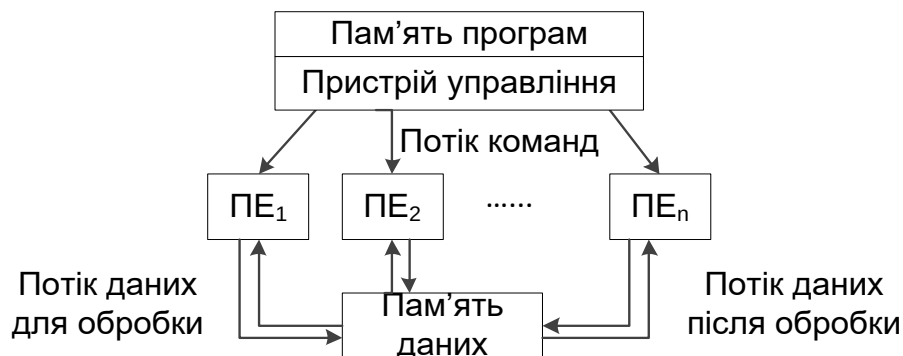


Рис. 14.2. ОС типу ОКМД (SIMD)

До такого ж типу відноситься ОС, що реалізує *макроконвеєр* (ОС "Україна"). У ній задача, що розв'язується циклічно, "розріжеться" на послідовні етапи, що закріплюються за окремими процесорами. Запускається конвеєр багаторазового виконання циклу, що становить завдання.

Принцип обробки наведено на рис. 14.3.

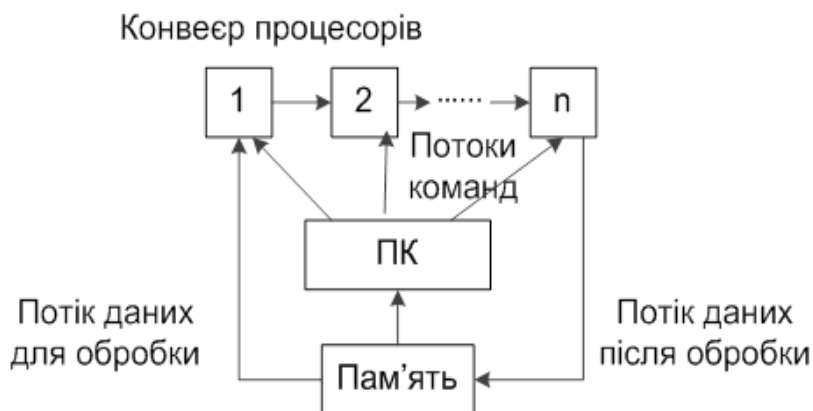


Рис. 14.3. ОС типу МКОД (MISD)

Тип БКМД – "багато потоків команд – багато потоків даних" (MIMD – "Multiple instruction – Multiple data") відповідає більш повному й незалежному розпаралелюванню.

«Фон-Неймановські» і «не-Фон-Неймановські» архітектури

Небагато історії. Першу ЕОМ створив в 1939 р. у США професор Джон Атанасов, болгарин, зі своїм аспірантом К. Беррі. Дві малі ЕОМ, створені ними в період 1937 – 1942 рр., були прототипами великої ЕОМ ABC для розв'язку систем лінійних рівнянь, яка в 1942 р. доводилась по пристроях введення-виведення й повинна була стати до ладу в 1943 р., але призив Атанасова в армію в 1942 р. перешкодив цьому.

Проект електронної ЕОМ Еніак (Electronics numerical integrator and computer) був зроблений в 1942 р. Д. Моучлі й Д. Еккертом і здійснений в 1945 р. у Муровській електротехнічній лабораторії Пенсильванського університету. В 1946 р. Еніак був привселюдно продемонстрований у роботі. У ньому вперше були застосовані тригери.

Народження Еніак вважають початком комп'ютерної ери, присвячуючи йому наукові симпозиуми й інші урочисті заходи.

Однак ще на початку 40-х років XX століття Атанасов поділився з Моучлі інформацією про принципи, закладені в ЕОМ ABC. Хоча Моучлі згодом стверджував, що він не скористався цією інформацією в патенті на Еніак, суд не погодився із цим. Повернувшись із армії після війни, Атанасов дізнався, що могутніша ЕОМ Еніак уже створена, і втратив інтерес до цієї теми, не поцікавившись, наскільки Еніак схожий на його ЕОМ ABC.

Відомий англійський математик Алан Т'юринг був не тільки теоретиком за інформацією й теорії алгоритмів, автором теоретичного автомата «машини Т'юринга», але й талановитим інженером, що створив на початку 1940-х років першу працюючу спеціалізовану ЕОМ. Ця ЕОМ за назвою «Колос» була сконструйована й зроблена їм разом з Х. А. Н'юменом у Блетчі (Англія) і почала працювати в 1943 р. Повідомлення про неї вчасно не публікувалися, тому що

вона використовувалася для розшифрування секретних німецьких кодів під час війни [3].

Основні архітектурно-функціональні принципи побудови ЦОМ були розроблені й опубліковані в 1946 р. угорським математиком і фізиком Джоном фон Нейманом і його колегами Г. Голдстайном і А. Берксом у звіті, що став класичним, «Попереднє обговорення логічного конструювання електронного обчислювального приладу».

Основними принципами ЕОМ на підставі цього звіту є: 1) принцип програмного керування виконанням програми, і 2) принцип збереженої в пам'яті програми. Вони лягли в основу поняття *фон-Неймановської архітектури*, що широко використовує *лічильник команд*.

Повернемося до сьогодення. Лічильник команд відбиває "вузьке горло", яке обмежує потік команд, що надходять на виконання, їх послідовним аналізом.

Альтернативною архітектурою є *"не-фон-Неймановська" архітектура*, що допускає одночасний аналіз більш однієї команди. Пошуки її обумовлені необхідністю розпаралелювання виконання програми між декількома виконавчими пристроями – процесорами. Лічильник команд при цьому не потрібний.

Порядок виконання команд визначається наявністю вихідної інформації для виконання кожної з них. Якщо кілька команд готові до виконання, то принципово можливо їх призначення для виконання такою ж кількістю вільних процесорів. Говорять, що такі ОС управляються *потокком даних (dataflow)* [3].

Загальна схема поточкових ОС наведено на рис. 14.4.

Програма або її частина (сегмент) розміщуються в пам'яті команд ПК, що складається із комірок команд. Команди мають структуру

{код операції, операнд 1, ..., операнд L,
адреси результату 1, ..., адреси результату M}

У командах перевірки умови можливе альтернативне завдання адреси результату (АБО – АБО). Адреси результатів є адресами ПК, тобто, результати виконання одних команд у якості операндів можуть надходити в текст інших

команд. Команда не готова до виконання, якщо в її тексті відсутній хоча б один операнд. Комірка, що володіє повним набором операндів, переходить в збуджений стан і передає в селекторну мережу інформаційний пакет (токен), що містить код операції й необхідну числову й зв'язну інформацію. Він надходить по мережі на одне з виконавчих пристроїв.

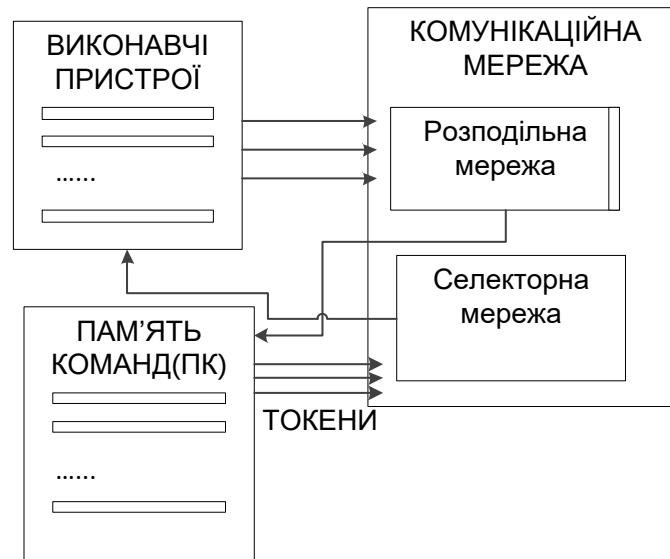


Рис. 14.4. "Ідеальна" потокова ОС

Там же операція виконується, і в розподільну мережу видається результатуючий пакет, що містить результат обчислень і адреси призначення в ПК (можливо, за рахунок вибору альтернативи, тобто такий вибір – теж результат). По цих адресах у ПК результат і надходить, створюючи можливість активізації нових комірок. Після видачі токена в селекторну мережу операнди в тексті команди знищуються, що забезпечує повторне виконання команди в циклі, якщо це необхідно.

Селекторна й розподільна мережі утворюють *комунікаційну мережу ОС*.

Очікувана надвисока продуктивність такої системи може бути досягнута за рахунок одночасної й незалежної активізації великої кількості готових команд, проблематичним припущенням про безконфліктну передачу пакетів по мережах і паралельну роботу багатьох виконавчих пристроїв.

Існує ряд труднощів, у силу яких «не-фон-Неймановські» архітектури не знайшли технічного втілення для масового застосування в «класичному», зазначеному вище, виконанні.

Однак багато пристроїв використовують даний принцип, але найчастіше взаємодія процесорів при спільному розв'язку загального завдання і їх синхронізація при використанні загальних даних засновані на аналізі готовності даних для їх обробки. Це дає підставу багатьом конструкторам заявляти, що у своїх моделях вони реалізували принцип *dataflow*.

Системи із загальною й розподіленою пам'яттю

Системи із загальною (розподілюваною) оперативною пам'яттю утворюють сучасний клас ОС – багатопроцесорних супер-ЕОМ. Однаковий доступ усіх процесорів до програм і даних надає широкі можливості організації паралельного обчислювального процесу (*паралельних обчислень*). Відсутні втрати реальної продуктивності на міжпроцесорний (між завданнями, процесами і т.д.) обмін даними (рис. 14.5а).

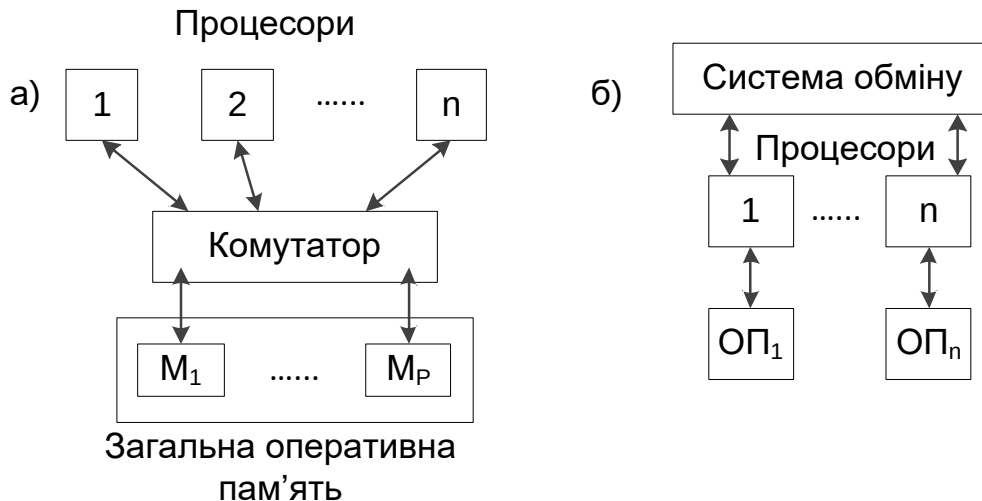


Рис. 14.5. ОС із загальною (а) і розподіленою (б) пам'яттю

Системи з розподіленою пам'яттю утворюють обчислювальні комплекси (ОК) – колективи ЕОМ з міжмашинним обміном для спільного розв'язку завдань. У ОК поєднуються обчислювальні засоби систем керування, що вирішують спеціальні набори завдань, взаємозалежних за даними.

Прийнято казати, що такі ОК виконують *розподілені обчислення*, а самі ОК називають *розподіленими ОК*.

Інше, протилежне втілення принципу МІМД – *маспроцесорні* або високо-паралельні архітектури, що поєднують сотні – тисячі – десятки тисяч процесорів. У сучасних супер-ЕОМ позначилася тенденція об'єднання двох принципів: загальної (що розподіляється) і розподіленої (локальної) оперативної пам'яті (ЛОП). (рис. 14.6).

Способи міжмодульного з'єднання (комплексування)

Розрізняють два протилежні способи комплексування: *із загальною шиною (шинна архітектура)* і *з перехресною (матричною) комутацією* модулів ОР(процесорів, модулів пам'яті, периферії) [3].

На рис. 14.7 наведена система із загальною шиною. Шина складається з ліній, по яких передаються інформаційні й керуючі сигнали.

Шина використовується в режимі поділу часу, при якому лише один модуль у цей момент працює на передачу. Приймати принципово можуть всі модулі, хоча переважно інформація при видачі в неї адресується. Застосовується в мікро- і міні-ЕОМ при порівняно невеликому числі модулів. Практично проводиться поділ шини на керуючу, адресну й шину даних.

У високопродуктивних ОС для можливості одночасного обміну багатьма парами абонентів використовується перехресна або матрична комутація.

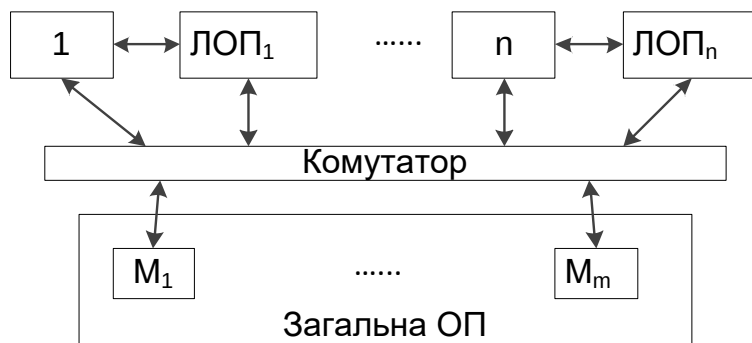
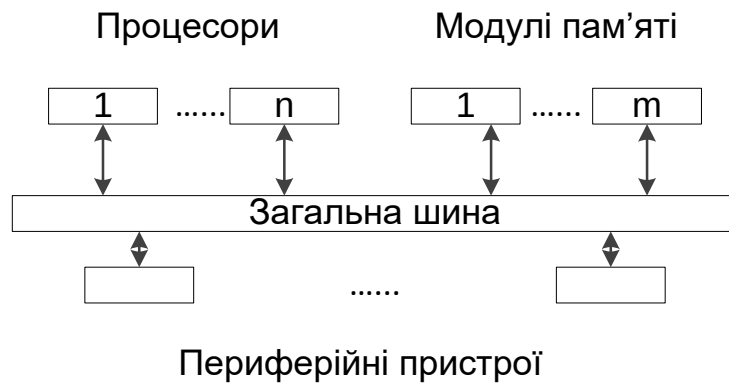


Рис. 14.6. Схема ОС із модулями локальної пам'яті



ис. 14.7. Схема ОС із загальною шиною

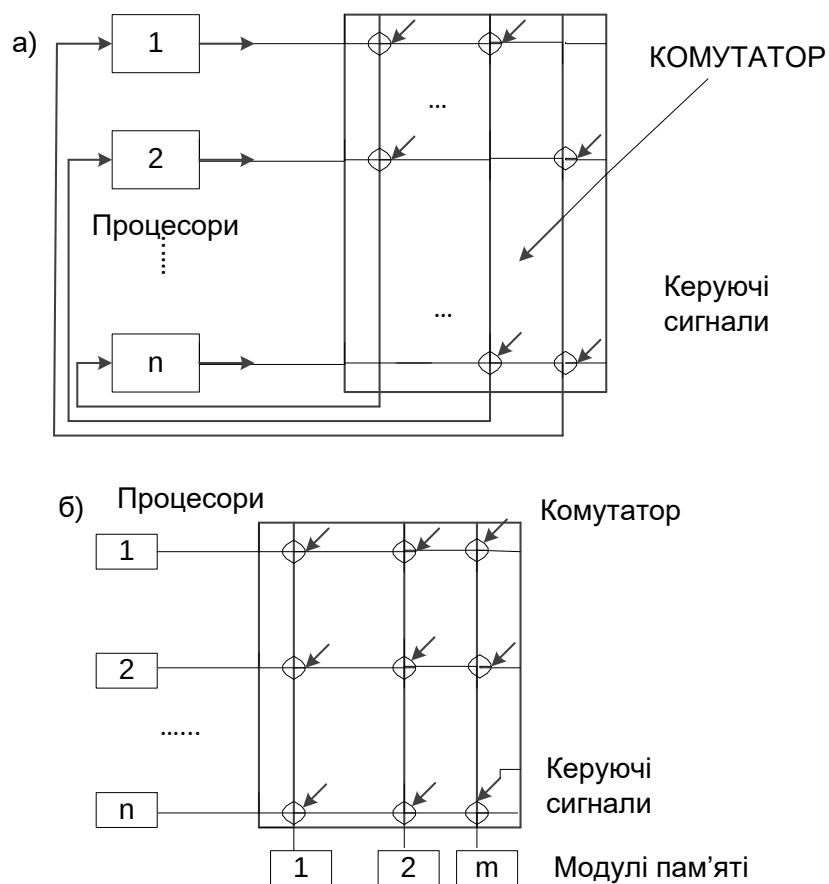


Рис. 14.8. Матричні комутатори: а) – перехресна комутація процесорів, б) – комутація процесорів і модулів пам'яті

Матричний комутатор можна представити прямокутною сіткою шин. До одного кінця кожної приєднане джерело-споживач інформації (рис. 14.8). Точки перетинання – *вузли* цієї сітки – являють собою *керуючі ключі*, які з'єднують або роз'єднують відповідні шини, установлюючи або припиняючи зв'язок між мо-

дулями. Реалізується зв'язок "кожний з кожним". Одночасно можуть зв'язуватися багато (до $n/2$) пар модулів.

На рис. 14.8 а – перехресний зв'язок між процесорами у ОС із розподіленою пам'яттю, на рис. 14.8б – між n процесорами й m модулями ОП [3].

"Історичні" моделі

Векторна ОС ПС-2000

Розроблялася як проблемно-орієнтована ОС для завдань обробки геофізичної інформації, інформації із супутників в інтересах геології, картографії, обробки зображень, моделювання поведінки середовища і т.д. Є типовою ілюстрацією типу ОКМД.

Схема ОС наведена на рис. 14.9.

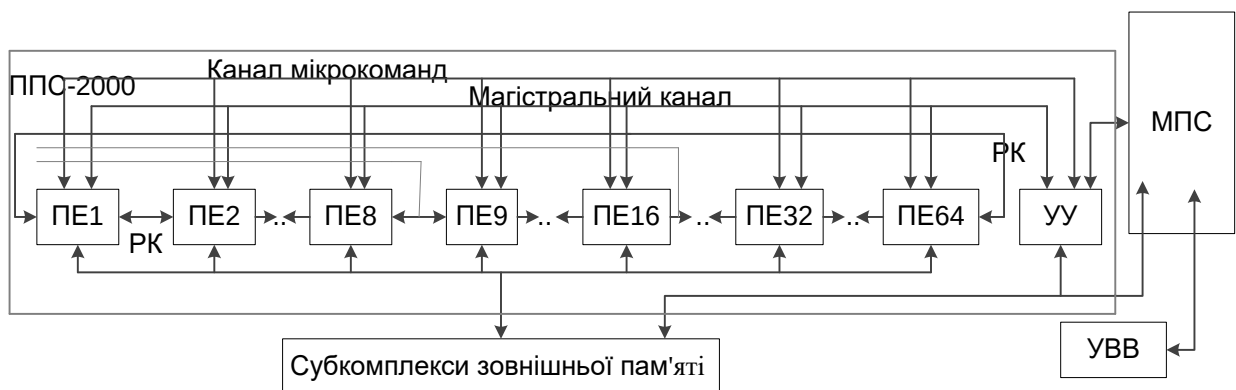


Рис. 14.9. Схема ОР ПС-2000

Основу ПС-2000 становить паралельний процесор із загальним потоком команд ППС-2000, що містить до 64 ПЕ.

Відзначаючи типові архітектурні особливості моделей, ілюструючи їх на архітектурах ОС, що стали класичними, приділимо другорядну увагу швидко застаріваючим характеристикам цих ОР. Однак відзначимо, що ПЕ ППС-2000 – 24-розрядний мікропроцесор із продуктивністю 3 млн оп/с, а це було значним досягненням на початку 1980-х років.

Сусідні ПЕ зв'язані *регулярними каналами РК* для оперативного обміну даними. У зв'язку із проблемною орієнтованістю ОС це забезпечує швидкий

обмін при обробці «сусідніми» ПЕ «сусідніх» же елементів поверхонь, звичайно ж, інформаційно взаємозалежних. Зв'язок за допомогою РК – циклічна, зокрема, ПЕ1 за допомогою РК пов'язаний з ПЕ64, тобто, при «прокатуванні» 64 ПЕ по довгій послідовності оброблюваних елементів поверхні ПЕ1 і ПЕ64 є «сусідніми»: після елемента поверхні, оброблюваного ПЕ64, впливає елемент поверхні, оброблюваний ПЕ1. Потім пішли далі, перетворивши векторну ППС-2000 у матричну ОС.

Повели сегментування множини процесорів (процесорів *вирішального поля*). 64 ПЕ розбиваються або на 8 сегментів по 8 ПЕ, або на 4 сегменти по 16 ПЕ, або на 2 сегмента по 32 ПЕ. Це здійснюється, насамперед, за рахунок установлення зв'язків за допомогою РК між першим і останнім ПЕ кожного сегмента.

Тоді при обробці поверхні ПЕ "прокачуються" циклічно не тільки по "довгому" вектору, але й при сегментуванні, по матричним елементам поверхні, як буде показане далі.

Для загального зв'язку всіх елементів ППС-2000, включаючи КП, використовується *магістральний канал-загальна шина*. По *магістралі мікрокоманд* кожному ПЕ повідомляються команди, мікропрограми операцій і мікрокоманди з КП.

ППС-2000 має розподілену пам'ять.

У ППС-2000 передбачена дворівнева схема керування: на рівні мікрокоманд і команд. Пам'ять для їхнього зберігання – в КП. На мікрокомандному рівні пишуться окремі фрагменти завдання – підпрограми процедур, на командному – послідовність звертання до цих підпрограм.

Основним є обчислювальний процес у *моніторній підсистемі МШС*. У ній у режимі поділу часу можуть виконуватися кілька завдань. Із ППС-2000 у цей момент може бути зв'язане тільки одне завдання. Тобто ППС-2000 стосовно малої машини (типу СМ-2), що виконує функції, МШС, є *інтелектуальним терміналом з послідовним доступом*.

У звичайному режимі набір процедур, що виконуються у ППС-2000, за запитами з боку МШС, визначається заздалегідь. Мікропрограми, що реалізують цей набір процедур, завантажуються до пам'яті ППС-2000 заздалегідь. Якщо в ході обчислювального процесу в МШС виникає необхідність у якій-небудь процедурі із цього набору, завдання запускає відповідну мікропрограму й передає їй необхідні дані [3]. Можливе й динамічне завантаження мікропрограм до пам'яті ППС-2000.

Матричні ОС

Матрична ОС також є типовим представником ОКМД, розширенням принципу векторних ОС.

Класичним прикладом, прототипом і еталоном стала ОС ILLIAC 1V, розроблена в 1971 р. в Ілінойському університеті й на початку 1972 р. установлена в Еймському науково-дослідному центрі NASA (Каліфорнія).

Одна послідовність команд програми управляє роботою множини 64ПЕ, що одночасно виконують ту саму операцію над даними, які можуть бути різними й зберігаються в ОП кожного ПЕ. (Продуктивність – до 200 млн. оп/с)

У цілому структура центральної частини може бути представлена аналогічно векторної ОС (рис. 14.10).

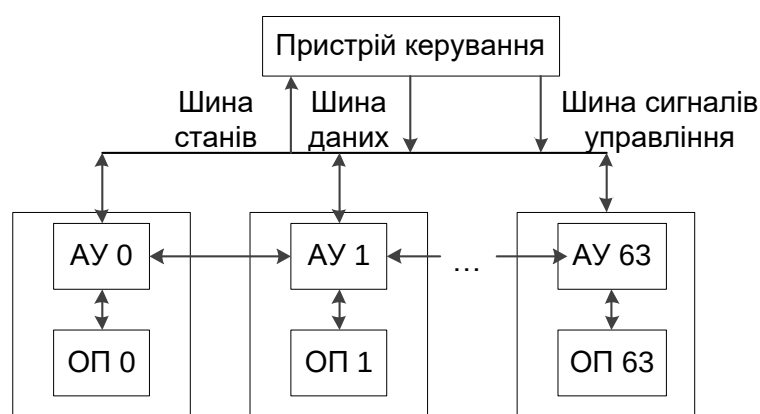


Рис. 14.10. Схема обміну в матричній ОС

ПК – фактично проста обчислювальна машина невеликої продуктивності, що може виконувати операції над скалярами одночасно з виконанням матри-

цею ПЕ операцій над векторами або матрицями. Посилає команди в незалежні ПЕ й передає адреси до їх ОП.

До центральної частини підключена система введення-виведення з керуючою машиною В-6500, пристрій керування введенням-виведенням, файлові диски, буферна пам'ять і комутатор введення-виведення. Операційна система, асемблери й транслятори розміщаються в пам'яті В-6500. Чому ж ОС – матрична?

ПЕ утворюють матрицю, у якій інформація від одного ПЕ до іншого може бути передана через *мережу пересилань даних* за допомогою спеціальних команд обміну. Регістри пересилань (рис. 14.11) кожного i -го ПЕ зв'язані високошвидкісними лініями обміну з регістрами пересилань найближчого лівого ($i-1$ -го) і найближчого правого ($i+1$ -го) ПЕ, а також з регістрами пересилань ПЕ, що відстає вліво на 8 позицій від даного ($i-8$)-й і віддаленого вправо на 8 позицій від даного ($i+8$)-й, при цьому нумерація ПЕ розглядається як циклічна з переходом від 63 до 0 (нумерація зростає по mod 64) зліва на право і від 0 до 63 справа на ліво.

Відстані між ПЕ в мережі пересилань і відповідні шляхи передачі інформації задаються комбінаціями з $(-1, +1, -8, +8)$.

Передбачене маскування ПЕ. Кожний ПЕ може індексувати свою ОП незалежно від інших ПЕ, що важливо для операцій матричної алгебри, коли до двовірного масиву даних потрібен доступ як по рядках, так і по стовпцях.

Доступ до ОП мають АП відповідного ПЕ, ПК системи й підсистема введення-виведення. Керування доступом і розв'язання конфліктів при доступі до ОП здійснює ПК системи.

Система ефективна при розв'язку завдань великої розмірності, що використовують вирахування кінцевих різниць, матричної арифметики, швидкого перетворення Фур'є, обробки сигналів і зображень, лінійного програмування та ін.

Наприклад, наближені методи розв'язку завдань математичної фізики й, насамперед, – системи диференціальних рівнянь використовують вирахування кінцевих різниць (кінцево-різницеві методи розв'язку, "метод сіток").

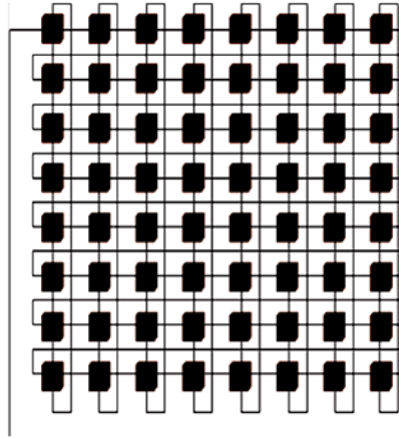


Рис. 14.11. Конфігурація "швидких" зв'язків у ОР ILLIAC-4

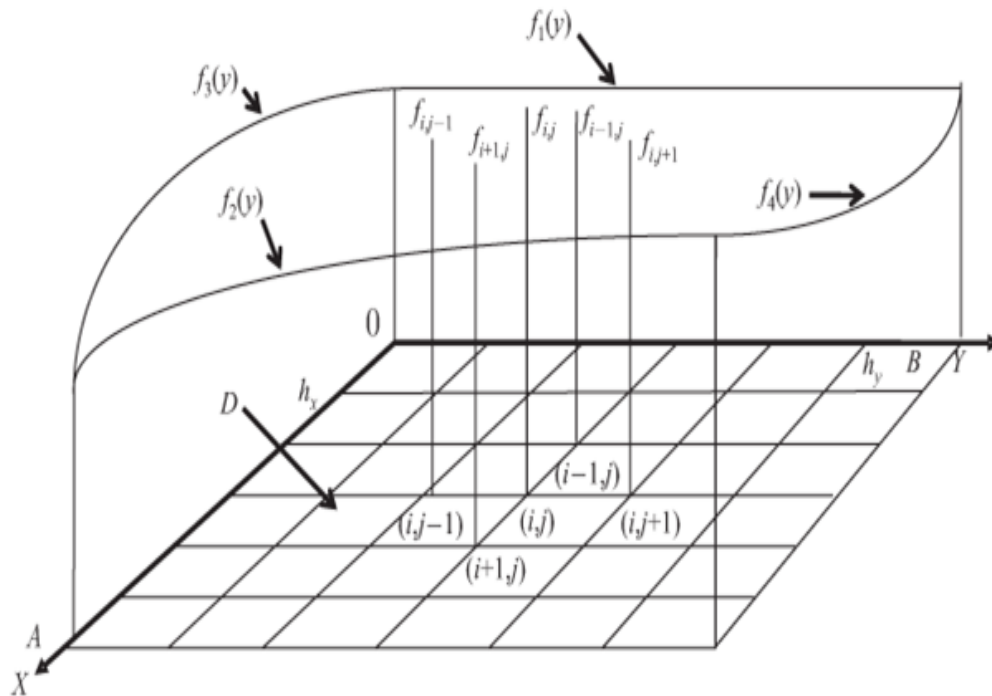


Рис. 14.12. Розв'язок завдання методом "сіток"

Коротко розглянемо план розв'язку подібного завдання на матричній ОС.

Нехай необхідно розв'язати рівняння із частинних похідних

$$\frac{\partial^2 f}{\partial x^2} - \frac{\partial f}{\partial y} = 0$$

на області В:

$$\begin{cases} 0 \leq x \leq A \\ 0 \leq y \leq B \end{cases}$$

Граничні умови:

$$f(0,y) = f_1(y),$$

$$f(A,y) = f_2(y),$$

$$f(x,0) = f_3(x),$$

$$f(x,B) = f_4(x).$$

Частинні похідні можна виразити через кінцеві різниці декількома способами, наприклад:

$$\begin{aligned} \left. \frac{df}{dx} \right|_{i,j} &\cong \frac{f_{ij} - f_{i+1,j}}{h_x} \cong \frac{f_{i+1,j} - f_{ij}}{h_x}; \\ \left. \frac{d^2 f}{dx^2} \right|_{i,j} &\cong \frac{(f_{ij} - f_{i+1,j}) - (f_{ij} - f_{i-1,j})}{h_x^2} = \frac{f_{i+1,j} - 2f_{ij} + f_{i-1,j}}{h_x^2}; \\ \left. \frac{df}{dy} \right|_{i,j} &\cong \frac{f_{i+1,j} - f_{i,j-1}}{2h_y}; \end{aligned}$$

Підставивши отримані вирази в рівняння, одержимо основне рекурентне співвідношення для знаходження значення функції через її значення в сусідніх чотирьох вузлах:

$$f_{i,j} = 0,5 f_{i+1,j} + 0,5 f_{i-1,j} - 0,25 h_x^2 / h_y (f_{i,j+1} - f_{i,j-1}).$$

Розпаралелювання обчислень і "прокатування" матрицею ПЕ області В – багаторазові, до одержання необхідної точності очевидні [1].

А саме: спочатку в одному циклі ітерації 64 ПЕ по виведеній формулі розраховують 8×8 вузлових значень функції біля вершини координат, "чіпляючи" при цьому граничні значення. Потім проводиться зсув праворуч на наступну групу вузлових значень функції. По закінченню обробки по горизонталі прово-

дитися зсув по вертикалі і т.д. При наступній ітерації процес повторюється. Від ітерації до ітерації поширюється вплив граничних умов на значення функції, що розраховуються. Процес сходиться для визначення значень функції-розв'язку у вузлах сітки з необхідною точністю.

Повернемося до вітчизняної розробки ПС-2000 (ПС-2100) – векторної ОС. У ній передбачена можливість сегментації процесорів, при якій останній процесор у сегменті зв'язується регулярним каналом (аналог високошвидкісної лінії) з першим. Усі 64 процесора можуть скласти один сегмент, можуть бути сформовано два сегменти по 32 ПЕ, чотири по 16, 8 по 8 ПЕ. Отже, виходить ідеальна схема для "покриття" двовимірної підобласті області, що обраховується. Правда, відсутні швидкі зв'язки по вертикалі, що повинні бути відшкодовані зберіганням усієї інформації зі стовпців в ОП кожного ПЕ. Однак, при зсуві процесорів на суміжну підобласть по горизонталі, останній ПЕ попередньої підобласті залишається "лівим" для першого в новій підобласті. Тобто, при такому зсуві зберігається регулярність фактичних зв'язків між вузлами сітки.

В ILLIAC-1V така регулярність зберігалася б, якби існували швидкі зв'язки між ПЕ7 і ПЕ0, ПЕ15 і ПЕ8 і т.д., тобто циклічно по рядках матриці. Як бачимо, такі зв'язки враховані при більш пізніх розробках матричних ОС.

ОС Cray-1 ("Електроніка ССБІС")

ОС Cray-1, незважаючи на наступні розробки, залишається еталоном типу МКОД. У Росії розроблений аналог – векторно-конвеєрна ЕОМ "Електроніка ССБІС", що відрізняється деякими збільшеними параметрами комплектації. Структура ОС наведена на рис. 14.13.

Призначена для виконання як векторних, так і скалярних операцій. Має 64 буферних регістрів, що вирівнюють, для 24-бітових адрес (В-Регістри), 64 буферних регістрів, що вирівнюють, для 64-бітових слів даних (Т-Регістри). В-регістри, у свою чергу, поповнюють буфер для однієї з основних груп регістрів машини – 8 адресних регістрів, а Т-Регістри – для групи 8 скалярних регістрів. Адресні регістри використовуються як і індексні. Третя основна група регістрів

–8 векторних реєстрів по 64-бітових слова. Регістр довжини вектора визначає число операцій, що виконуються по векторних командах, тобто, дійсну довжину вектора. Маска вектора визначає елементи, над якими виконується операція.

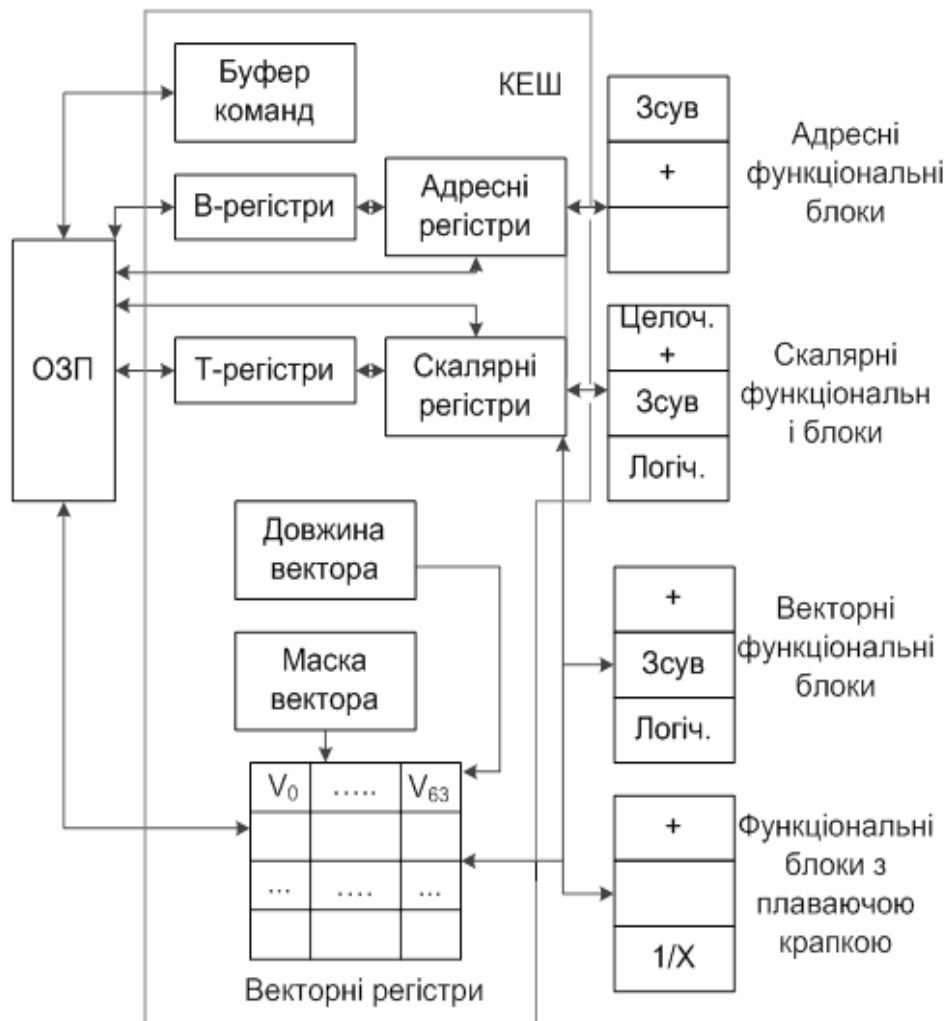


Рис. 14.13. Схема ОС Cray-1

Буфер команд складається із чотирьох ЗП, кожне з яких – з 64 16-розрядних реєстрів. Представляє 4 програми, виконувані в мультипрограмному режимі. (В «Е-ССБІС" – 16 таких ЗП.)

12 спеціалізованих функціональних блоків-конвеєрів (в «Е-ССБІС" 16 блоків) виконують арифметичні, логічні операції й зсув рівнів конвеєра – від 2 до 14. Блоки є незалежними. Кілька функціональних блоків можуть працювати одночасно. Можливо «зчеплення» векторів при виконанні операцій виду $A \times B$

+ C, коли два або більше конвеєрів вишуковуються в один і результат з одного негайно надходить як операнд на іншій [6].

Таким чином, паралелізм в обробці даних організується за допомогою:

1. одночасного використання різних функціональних блоків і різних векторних регістрів;
2. застосування режиму «зчеплення» векторів.

(Нас цікавить архітектура, але не катастрофічно старіючі характеристики. Однак для довідки: такт машини $12\{, \}5\text{нс}$, цикл звертання до ОЗП 50нс , продуктивність вважається: 38 млн. скалярних оп/с і 80 млн. векторних оп/с.)

14.2 Мікропроцесорні системи й способи розпаралелювання

Мультимікропроцесорні обчислювальні системи

У цей час вибір зроблений на користь багатопроцесорних симетричних ОС типу *MIMD* обчислювальних ресурсів, що забезпечують віртуалізацію. Основу такої ОС становить *суперскалер*, що зосередив у собі всі способи досягнення максимальної швидкодії при виконанні одиночної програми. Векторні й векторно-конвеєрні процесори й системи одержали своє місце. Їхня ефективність як самостійних установок могла бути досить високою тільки при розв'язку спеціальних завдань і тестів.

Тому досить швидко з'ясувалося, що ці установки можуть виконувати функції інтелектуальних терміналів при розв'язку основного завдання на іншому універсальному обчислювальному засобі й виконувати лише окремі його заявки. Сьогодні стало остаточно ясно, що перші ефективні лише в ролі спеціалізованих обчислювальних пристроїв для розв'язку спеціальних завдань. Другі твердо зайняли місце в складі багатофункціональних арифметико-логічних пристроїв (АЛП) суперскалерів, тому що без конвеєрів ми не уявляємо собі виконання всіх операцій ОС [16].

Складається й структура пам'яті ОС, яка може сполучати в одній установці всі способи доступу: від поділюваної (загальної) до розподіленої оператив-

ної пам'яті. Однак обмежені можливості ефективної роботи із загальною пам'яттю часто диктують ієрархічну структуру ОС, де рівні ієрархії (кластери) відрізняються або способом доступу до оперативної пам'яті, або тим, що кожний кластер має свою власну фізичну пам'ять у загальному адресному просторі. При цьому принцип буферизації, засновано на багаторівневій по швидкодії (і, звичайно, – різної за технологією) пам'яті, на активному використанні кеш-пам'яті, продовжує розвиватися. Кеш-пам'ять, як пам'ять найвищого рівня, зазнає функціональної розбивки залежно від типу даних, для зберігання яких вона призначена, або, залежно від виду обробки, – програм або даних.

Зараз мікропроцесор, що сконцентрував усі досягнення мікроелектроніки, є основною складовою елементно-конструкторської бази ОС. Тому поняття «мультимікропроцесорні ОС» прийшло на зміну поняттю "мікропроцесорні ОС".

Аналіз сучасних мультимікропроцесорних ОС дозволяє виділити ті характерні розв'язки, що розвиваються, які в умовах мікромініатюризації й зниження енергоємності, «ощадливого» логічного розвитку забезпечують необхідні властивості універсального застосування [17].

Такими розв'язками є наступні.

1. Багатопроеесорні кристали.

Відтворення багатопроеесорної ОС на одному кристалі в значній мірі характерно для сигнальних обчислювальних засобів, що спеціалізуються на обробці двох- і тривимірних зображень, які застосовуються в цифровому телебаченні й радіомовленні, при передачі зображень по каналах зв'язку й ін. Такі засоби ефективно використовуються в якості нейрокомп'ютерів.

Наприклад, на одному кристалі *MVP (Multimedia Video Processor)* сімейства *TMS 320 C80* (фірма *Texas Instrument*) розташовано 4 32-розрядних цифрових сигнальних процесора (*DSP – Digital Signal Processor*) з фіксованою комою (*ADSP-0 –ADSP-3*). Їхня особливість – високий ступінь конвеєризації й довжина командного слова для паралельного виконання декількох операцій до 64 біт. Система команд містить команди над бітовими полями й структурами даних,

що несуть графічну інформацію. Така спеціалізація обумовила поняття – DSP-архітектура.

Процесори працюють незалежно. Тобто, ОС – типу *MIMD* – (Multiple-Instruction, Multiple-Data). Програмуються окремо на асемблері або MBY. Даними обмінюються через загальну внутрішньокристалъну пам'ять.

Кожний з *ADSP* містить кеш-пам'ять команд (2 Кбайти), і через матричний комутатор *Crossba* отримує доступ до 32 наявних 50 Кбайт швидкодіючої статичної внутрішньої пам'яті. Пам'ять розшарована – поділена на сегменти. Якщо два й більш процесори в одному циклі спробують звернутися до одного сегменту, апаратна система керування доступом із циклічною зміною пріоритету (*round robin prioritization*) дозволить зробити це тільки одному процесору.

32-розрядне АЛП *ADSP* може працювати як два 16- або чотири 8-розрядних АЛП. Цього досить для обробки відеозображень. Спеціальні блоки прискорюють обробку графіки. Блоки генерації адреси формують кільцеві (нескінченні) буфери. Апаратно підтримано три вкладені цикли.

RISC-процесор управляє чотирма *ADSP* за допомогою диспетчера. Диспетчер і планувальник завдань тісно взаємодіють із контролером пересилань. Крім того, керуючий процесор самостійно виконує обчислення й забезпечує обмін із зовнішніми пристроями. Містить вбудований блок плаваючої арифметики й набір векторних операцій із плаваючою комою, оптимізованих для обробки зображень, звуку й тривимірної графіки [6].

2. Трансп'ютерна технологія.

Представлена вище архітектура має таку конструктивну закінченість, яка дозволяє як вбудовувати її в деяку систему, так і організувати взаємодію декількох кристалів. Це забезпечується розвиненими засобами зв'язку й обміну даними.

Можливість комплексування привернула увагу ще на ранньому етапі розвитку мікропроцесорів (у середині 1980-х років) і привела до побудови *трансп'ютерів* – мікропроцесорів, оснащених розвиненими засобами комплексування. Таким чином, створювалися «цеглинки», на основі яких можна було

створювати складні структури. Ця тенденція не тільки збереглася, але є необхідним засобом побудови мультимікропроцесорних ОС.

Переслідуючи багатофункціональність засобів обміну, не обов'язково вимагати їхнього розміщення на одному кристалі із центральним процесором. Так, фірма *Analog Devices* пропонує мікропроцесори *ADSP-21060/62 SHARC* («АКУЛА») для цифрової обробки сигналів, спеціально призначені для комплексування.

Засіб комплексування «АКУЛИ»:

- магістраль для підключення 6 «АКУЛ» і одного ХОСТ-процесора (керуючого, із привілейованим доступом до магістралі, а також до пам'яті кожного процесора – через спеціальний порт);
- сигнальні регістри в складі кожного процесора, безпосередньо зв'язані (однією нішкою) з кожним з інших процесорів – для контролю їх стану;
- ЛІНКи – кожний процесор має 6 виходів (ЛІНКів) для безпосереднього зв'язку «процесор – процесор».

3. *Загальний адресний простір.*

Мікропроцесор «АКУЛА», що комплексуються, забезпечує псевдозагальну пам'ять і виключає необхідність програмної організації обміну даними. Якщо адреса фізично належить ОП іншого процесора, то обмін організується автоматично, без втручання користувача (тобто програмно не передбачається).

4. *Міжпроцесорний (магістральний) обмін.*

Ініціюється в тому випадку, якщо адреса зчитування або запису належить адресному простору іншого процесора (одиничний обмін). Аналогічно виникають групові пересилання даних з використанням «чужого» адресного простору.

Користувач не складає програму обміну, навіть для контролера обміну даних. Досить указати «чужі» адреси.

Процесори обмінюються сигналами стану. Тому кожний процесор знає, хто є «хазяїном» магістралі, тобто веде обмін, і свій пріоритет у черзі до магістралі. По завершенню кожного обміну проводиться циклічна зміна пріоритетів

процесорів, яким потрібна магістраль. Процесор з максимальним пріоритетом стає «хазяїном». Обмін може перериватися тільки ХОСТ-процесором.

Мікропроцесор стверджується в ролі основи елементно-конструкторської бази ОС, і це зрозуміли провідні розроблювачі.

У цьому сенсі привертає увагу трансформація інтересів «батька суперкомп'ютерів» С. Крея, який визнав визначальну роль принципу *MIMD* при побудові ОС *Cray Superserver 6400 System (CS640)*, що випускається корпорацією Cray Research у співробітництві з компанією SUN Microsystems (співробітництво з фірмою SUN нині характерно й для провідних російських розроблювачів).

Система передбачає нарощувану конфігурацію від 4 до 64 процесорів *SuperSPARC*. Прийнятий компромісний розв'язок на основі класичної схеми поділу (загальної) ОП при багатопроцесорній обробці й розподіленій пам'яті при паралельній обробці масивів. Щоб працювати із частково розподіленою пам'яттю в ОЗП, ОС має в будь-якій конфігурації 4 шини. Шина використовує мережну технологію «комутації пакетів». Це дозволяє знаходити шлях обміну одиницями інформації відповідно до зайнятості або звільнення шин.

У цілому, архітектуру слід вважати шинною, хоча наявність декількох шин робить її проміжною між шинною, і тією, що використовує матричний комутатор [3].

Напрямок «міні-супер» покликано підтримати персональний комп'ютер

Те, що говорилося вище, «за замовчуванням» відповідає розробці супер-ЕОМ, призначених для розв'язку особливо складних завдань у складі систем керування в реальному часі, моделювання найскладніших фізичних процесів, розв'язку завдань дослідження операцій, завдань штучного інтелекту, виконання ролі мейнфреймів і серверів у локальних, корпоративних і глобальних мережах.

Супер-ЕОМ унікальна, мало тиражована, ціна її висока.

З іншого боку, ніщо вже не може зупинити «переможного ходу» персональних комп'ютерів. Область застосування їх стала всеосяжною. Вони викорис-

товуються і там, де можуть упоратися із завданнями, і там, де вже не справляються, незважаючи на застосування сучасних суперскалерів.

Тоді доцільно поставити наступну проблему.

Уведемо до складу персонального комп'ютера (*PC*), як його зовнішній пристрій, мультимікропроцесорну систему (мультипроцесор), використання якого в монопольному й однозначному режимі може забезпечити успішний розв'язок завдань підвищеної складності.

Дійсно, розв'язок цієї проблеми дозволив би заповнити певну нішу між супер-ЕОМ і *PC*, вивести персональний комп'ютер на рівень міні-супер-ЕОМ. Застосування мультипроцесора *PC* в однопрограму режимі, при жорсткому розподілі пам'яті, використання (див. далі) прогресивної технології "одна програма – багато потоків даних" дозволяють суттєво знизити витрати продуктивності на роботу ОС, легко «врізати» їх у сучасні операційні системи комп'ютерів. Складання такої системи повинне проводитися на основі існуючої мікропроцесорної елементно-конструкторської бази, з мінімальним використанням знову розроблювальних елементів.

Тут відтворюється згадана вище ідея про наявність моніторної системи, на якій вирішується основне завдання, і про наявність інтелектуального терміналу, який бере на себе функції, що забезпечують загальну ефективність системи.

Загальну схему такої установки наведено на рис. 14.14. Обрані конкретні значення параметрів [12].

Мультимікропроцесорну приставку до персонального комп'ютера доцільно розробити на основі дослідження принципів побудови локально-асинхронної архітектури (SPMD-технології).

Важливою перевагою архітектури є зведення традиційних функцій ОС на рівень команд. Тобто. система команд мультипроцесора така, що дозволяє реалізувати функції керування паралельним процесом, не вимагаючи запуску процедур ОС.

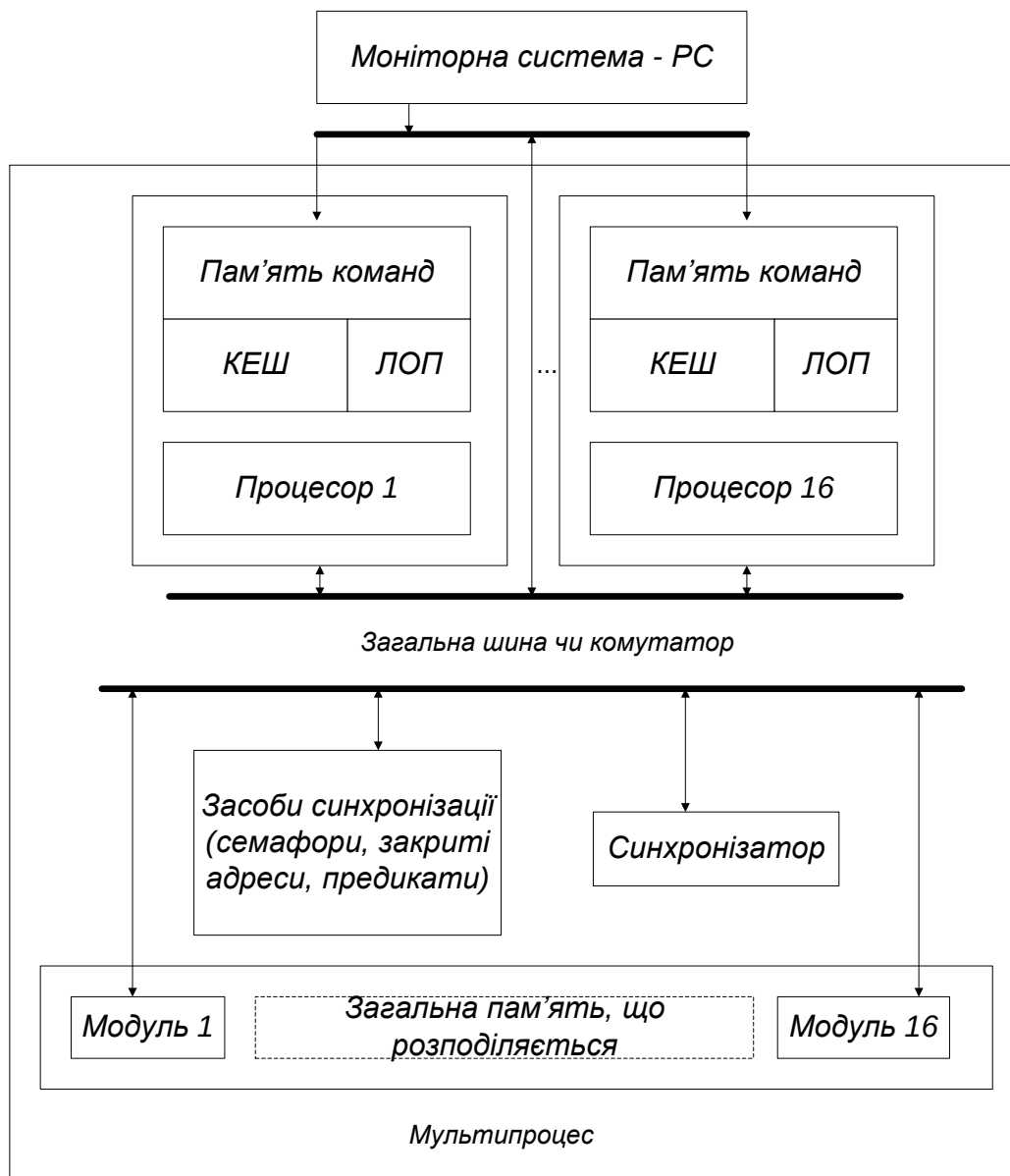


Рис. 14.14. Схема ОР для персонального комп'ютера

Сприяє простоті керування паралельним процесом також монопольний і однозадачний режим використання мультипроцесора. Нижче ми докладніше зупинимося на принципах SPMD-технології. Припускаючи первісне знайомство із цими принципами, відзначимо наступне.

Відомо (див. далі), що семафори – універсальний засіб синхронізації. Однак семафори традиційно використовують ОС. Щоб цього уникнути, семафори слід реалізувати за допомогою *предикатного механізму*, тобто з використанням пам'яті предикатів.

Семафорний механізм може бути ефективно реалізований за допомогою механізму закриття адреса (пам'яті закритих адрес).

Тоді, у загальному випадку застосування семафорів, повинні бути введені команди наступного виду.

Рахувати за семафором (Rax(C)A). Зчитування за адресою проводиться у випадку, якщо зазначений семафор (реалізований у пам'яті реєстрового типу, поряд з індексними й базовими реєстрами) відкритий. Якщо семафор закритий, реалізується очікування даного ПЕ без переривання (тобто в даному застосуванні користувач може отримати доступ до операцій над семафорами).

Записати за семафором (Zap(C)A). Запис за адресою проводиться аналогічно попередній команді.

При використанні пам'яті закритих адрес необхідна лише команда *Закрити адресу*. Будь-яке наступне зчитування по цій адресі циклічно відновлюються доти, поки по цій же адресі інший процесор не зробить запис.

У випадку використання механізму предикатів адреси деякої булевої змінної записуються в спеціальні розряди командного слова.

Команда, для якої зазначений у ній предикат має значення 0, виконується, відповідно до коду операції, у *спекулятивному режимі* у двох модифікаціях [9]:

- очікується присвоєння даному предикату значення 1;
- пропускається виконання даної команди.

ПЕ реалізує ідею RISC-архітектури і являє собою функціонально закінчений пристрій, що складається з мікропроцесора, схем обрамлення й локальної оперативної пам'яті (ЛОП).

Локальна пам'ять процесора містить область для зберігання стеків обчислювального процесу, у тому числі – стеків підпрограм і вкладених циклів. В інших областях цієї пам'яті зберігаються модифікатори, дескриптори масивів і локальні величини. Відразу перебувають мікропрограми, що реалізують систему команд ОР.

Загальна (поділювана) пам'ять (ОП) містить **М** модулів із загальним адресним простором і реалізує принцип *інтерлівінга*, що припускає, що суміжні комірки пам'яті перебувають у різних модулях.

Синхронізатор призначений для забезпечення одночасного пуску програм або їх модулів.

Можливе використання найпростіших комутаторів для обміну ПЕ з модулями пам'яті.

Розподілений і поділюваний обчислювальний ресурс другого рівня. Вирішальні поля.

Зробимо деякі узагальнення.

Отже, другий рівень розпаралелювання передбачає розподіл команд, інструкцій, операцій, елементарних функцій і інших нескладних процедур - для виконання виконавчими пристроями процесорів або в загальному обчислювальному ресурсі симетричної ОР. Отут існують свої проблеми, пов'язані з "елементарним" характером операцій, невеликим обсягом робіт, що містяться в них, з їх ще більшою критичністю стосовно «накладних витрат» на організацію й синхронізацію. Ми припускаємо, що виконавчі пристрої ОР утворюють обчислювальний ресурс іншого рівня (розпаралелювання).

Склалися традиції побудови цього ресурсу, де основна увага приділяється побудові багатофункціональних АЛП. Однак у ряді архітектур поки ще неспішно пробиває собі дорогу об'єднання АЛП в єдиний поділюваний ресурс системи – побудову вирішальних полів.

Реалізація цієї ідеї проводилася неодноразово у вітчизняній практиці розробки ОР. Вона проявлялася у включенні до складу ОР спеціалізованих процесорів на правах інтелектуальних терміналів для ефективного виконання певних операцій. Це були векторні процесори з доступом від декількох ЦП (ПС-3100), векторний співпроцесор у ПС-2100. Ця ж ідея фактично втілена в сімействі "Ельбрус", що допускає включення до свого складу спецпроцесорів – емуляції інших систем, векторно-конвеєрних модулів і ін.

При реалізації ідеї вирішального поля проблема вибору й розвитку обчислювального ресурсу невіддільна від вибору варіантів архітектури системи взагалі. Єдиним засобом обґрунтування й дослідження цього вибору є моделювання. Побудова детермінованих імітаторів дозволяє з будь-якою деталізацією виявити доцільні технічні розв'язки й обґрунтувати мову системи. Метою стохастичних моделей є оцінка ефективності різних архітектур на основі повноти корисного завантаження обладнання.

На основі традицій розробки багатопроцесорних симетричних обчислювальних систем можна зробити висновок про практику й тенденції розвитку обчислювального ресурсу іншого рівня. Усі моделі сімейства МОК "Ельбрус" припускають наявність у складі АЛП процесора декількох виконавчих пристроїв (ВП), спеціалізованих за типами операцій.

Тоді в цілому для ОР можна сказати, що обчислювальний ресурс іншого рівня є *розподіленим і неоднорідним* (рис. 14.15).

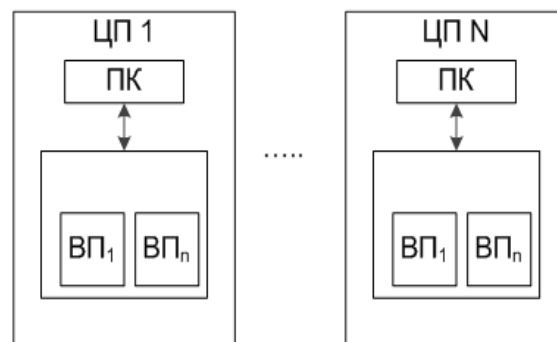


Рис. 14.15. ОР із розподіленим вирішальним полем

Вище говорилося, що використання ресурсу іншого рівня невіддільно від загальних ідей функціонування процесорів ОР – від їхньої архітектури й від архітектури ОР у цілому. Тому сказаного про розподілений ресурс недостатньо, треба говорити й про спосіб його використання.

Узагальнений алгоритм динамічного завантаження засновано на проміжному перекладі безадресних команд у трьохадресні. Поява адреси аргументів і результатів у кожній команді дозволяє на основі спільного аналізу декількох команд, представлених в "вікні перегляду", виявляти їхню часткову впорядко-

ваність і виділяти незалежні команди для одночасного виконання. Це – адресний метод розпаралелювання, який розглядався в архітектурі *data flow*. Команди проходять стадії обробки, як показано на рис. 14.16.

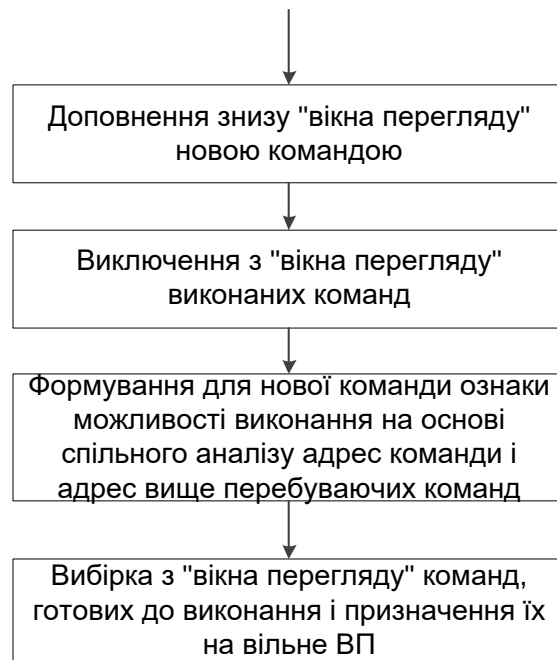


Рис. 14.16. Схема оптимізатора-компоновщика "довгих" командних слів

Способи розпаралелювання

Розрізняють два основні способи розпаралелювання: за керуванням й за інформацією [4].

Перший спосіб – представлення алгоритму завдання у вигляді частково-упорядкованої послідовності виконуваних робіт. Потім у результаті диспетчерування реалізується оптимальний план виконання робіт у ОР при обмеженнях на час виконання всього алгоритму або за мінімальний час.

Основою є представлення алгоритму граф-схемою G , що відбиває інформаційні зв'язки між роботами (завданнями, процесами, процедурами, операторами, макрокомандами і т.д.), на які розбитий алгоритм. Граф G – зважений, орієнтований, без контурів.

Для дослідження графа й диспетчерування використовують матриці слідування S ; їх доповнюють стовпцем T важелів – одержують розширені матриці проходження S^* (рис. 14.17).

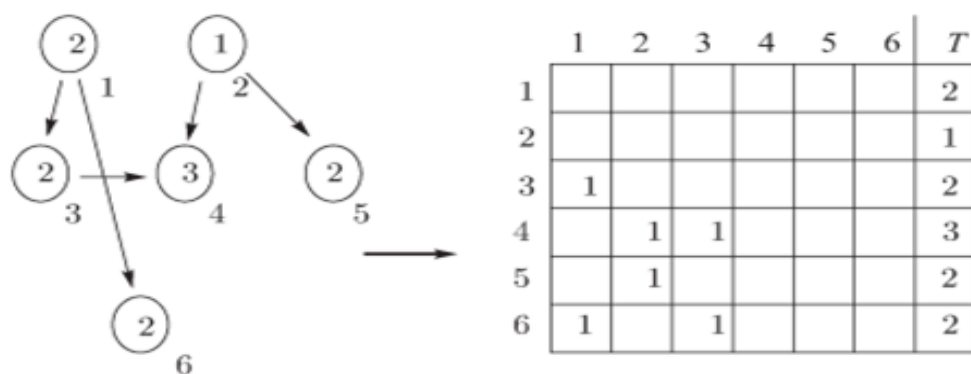


Рис. 14.17. Вхідна інформація для розпаралелювання

Тут припускаємо, що ОР – однорідна, із загальною (поділюваною) пам'яттю, тобто втратами часу на обмін між роботами можна знехтувати.

Нехай ОР містить два процесори ($n = 2$). Тоді в результаті оптимального розподілу одержимо план (рис. 14.18).

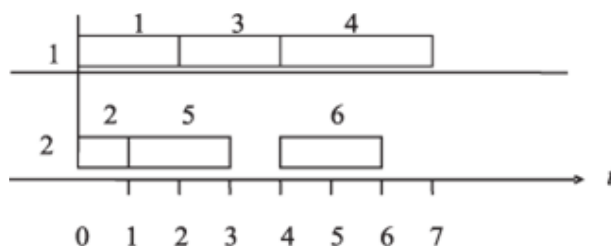


Рис. 14.18. Часова діаграма паралельного виконання робіт

План дійсно збігається з оптимальним, тому що довжина розкладу $T = 7$, що збігається з довжиною критичного шляху в графі, $T_{кр} = 7$ (шлях $1 \rightarrow 3 \rightarrow 4$).

У загальній схемі організації паралельного обчислювального процесу ми не повністю розкрили зміст блоку 3 – інтерпретації потоку макроінструкцій у вигляді, зручному для роботи диспетчера. Зараз ми визначили, що такий вид – це матриця слідування.

Виходить, якщо буде потреба автоматичного формування матриці проходження треба визначати інформаційний взаємозв'язок макроінструкцій у межах видимості, тобто в «вікні перегляду». Таким чином, за потоковим вмістом «вікна перегляду» треба формувати поточний вид матриці слідування [5].

Згадаємо, що ми вже в спрощеному виді вирішували подібне завдання, наприклад, коли по формованому потоку трьохадресних команд визначали їхній інформаційний взаємозв'язок і визначали можливість одночасного виконання цих команд.

Узагальнимо це завдання. Вертаючись до названої схеми, уявимо собі, що потік макроінструкцій (блок 2) впливає через «вікно перегляду» так, що для планування оптимального завантаження процесорів диспетчер може аналізувати деяку безліч цих макроінструкцій і з їх вибирати варіант призначення їх на процесори для виконання.

Кожна макроінструкція може інтерпретуватися і як процедура, де можна виділити ім'я θ_μ множини $\{\alpha_\mu\}$ вхідних параметрів, множини $\{\beta_\mu\}$ вихідних параметрів.

На рис. 14.19 відображене «вікно перегляду», через яке впливає потік макроінструкцій.

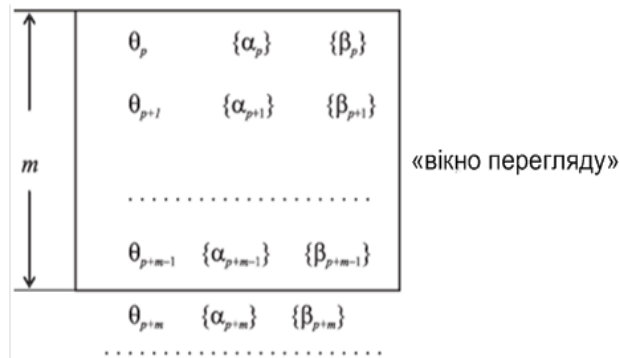


Рис. 14.19. Обробка «вікна перегляду»

Складемо за його вмістом відповідну матрицю слідування розмірності $m \times m$:

$$S = \left\| \alpha_{\mu\nu} \right\|_I^m = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1m} \\ \dots & \dots & \dots & \dots \\ \alpha_{m1} & \alpha_{m2} & \dots & \alpha_{mm} \end{pmatrix}$$

$$\alpha_{\mu\nu} = \begin{cases} 1, \text{ якщо } \varepsilon_{\mu\nu} \neq \otimes \\ 0, \text{ інакше} \end{cases}$$

$$\varepsilon_{\mu\nu} = (\{\alpha_\mu\} \cap \{\beta_\nu\}) \cup (\{\beta_\mu\} \cap \{\alpha_\nu\} \cup \{\beta_\nu\}) \text{ для усіх } \nu < \mu$$

По матриці слідування S диспетчер робить призначення.

Після виконання макроінструкцій вони виключаються з «вікна перегляду» макроінструкції, що залишилися, ущільнюються нагору, а знизу «вікно перегляду» поповнюється новими макроінструкціями.

З урахуванням макроінструкцій, що знову надійшли, уточнюється поточний вид матриці слідування S і процес диспетчерування триває.

За такою ж схемою, а саме, на основі першого способу розпаралелювання – за керуванням – вирішується інше важливе завдання розпаралелювання: *компонування довгих командних слів в трансляторі, що оптимізує*. Призначення роботи на ВП здійснюється тут у вигляді запису відповідного до інструкції в позицію довгого командного слова, що відповідає ВП.

Тобто план паралельного виконання робіт (команд, операцій) фіксується в довгих командних словах, у яких передбачені інструкції кожному ВП, які вони повинні почати виконувати з даного такту.

Другий спосіб розпаралелювання – за інформацією – використовується тоді, коли можна розподілити оброблювану інформацію між процесорами для обробки за ідентичними алгоритмами (за одним алгоритмом).

1. Розглянемо завдання множення матриць $A \times B = C$:

$$\begin{pmatrix} a_{11} & \dots & a_{1m} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} \times \begin{pmatrix} b_{11} & \dots & b_{1m} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mm} \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1m} \\ \dots & \dots & \dots \\ c_{m1} & \dots & c_{mm} \end{pmatrix}$$
$$c_{ij} = \sum_{k=1}^m a_{ik} b_{kj}$$

Розгорнемо матрицю – результат C – у лінійний(одномірний) масив, перейменуємо її елементи й замінимо два індекси на один:

$$\begin{matrix} c_{11} & c_{12} & \dots & c_{1m} & c_{21} & \dots & c_{2m} & c_{31} & \dots & c_{mm} \\ d_1 & d_2 & \dots & d_m & d_{m+1} & \dots & d_{2m} & d_{2m+1} & \dots & d_m^2 \end{matrix}$$

Нехай ОР містить n процесорів. Виберемо наступний план рахунку елементів матриці C :

процесор 1 рахує елементи $d_1, d_{1+n}, d_{1+2n}, \dots$

процесор 2 рахує елементи $d_2, d_{2+n}, d_{2+2n}, \dots$

.....

процесор n рахує елементи $d_n, d_{2n}, d_{3n}, \dots$

Очевидно, усі вони будуть виконувати ту саму програму, але обробляти різні набори даних. (Ми знову зіштовхнулися з доцільністю SPMD-технології.)

Тут не потрібна була будь-яка синхронізація паралельного обчислювального процесу.

2. Розглянемо завдання розрахунку способом «піраміди».

Це завдання ми досліджували при розгляді ОР типу SPMD. Подивимося ще раз, яка синхронізація нам тут буде потрібна.

Нехай необхідно перемножити всі елементи деякого масиву $\{a_1, a_2, \dots, a_{10}\}$. Кожний елемент займає одну комірку пам'яті. Нехай число процесорів у ОР $n=4$. Щоб розпаралелити цей процес, приймемо схему розрахунку "пірамідою" (рис. 14.20).

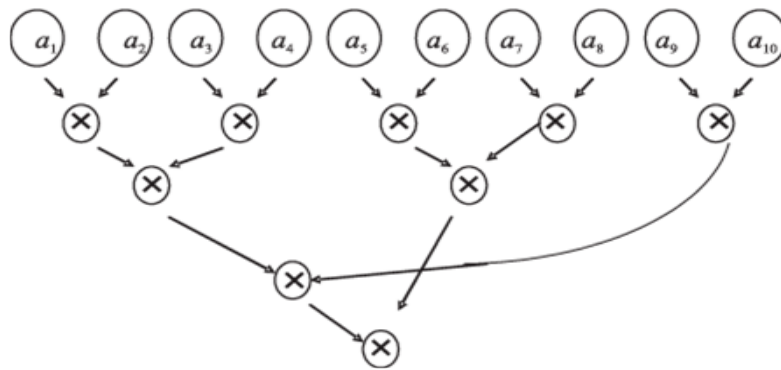


Рис. 14.20. Граф-схема виконання операції «згортки»

Кількість рівнів операцій у ній $\lceil \log_2 m \rceil = \lceil \log_2 10 \rceil = 4$ ($\lceil x \rceil$ – найближче ціле, не менше x).

Розширимо масив, доповнивши його комірками, у яких будемо зберігати проміжні часткові добутки. Тоді весь план розрахунку приймемо таким, як наведено на рис. 14.21. Відзначені процесори, що виконують зазначену операцію.

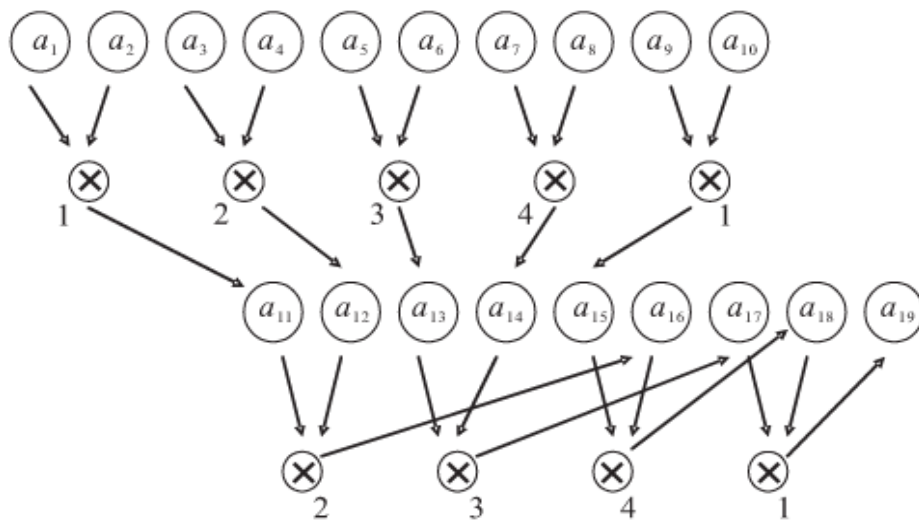


Рис. 14.21. Схема виконання операції згортки чотирма процесорами

Отже, треба так написати програму, одну для всіх процесорів, передбачивши необхідну переадресацію для вибірки й обчислення «своїх» даних, щоб по ній вибиралися два сусідні елементи цього подовженого масиву, а результат їх множення відправлявся в чергову комірку цього «подовження».

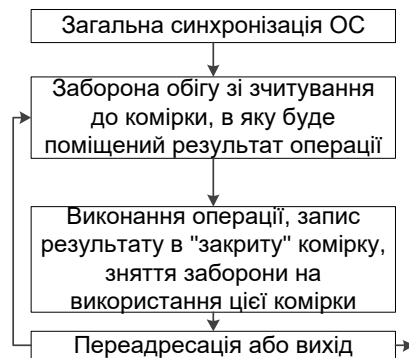


Рис. 14.22. Схема програмної синхронізації при виконанні операції «згортки»

Виникає тільки одна складність: для перших п'яти добутків дані є, а наступні добутки повинні виконуватися тоді, коли для них будуть знайдені вхідні дані.

Виходить, процесори, яким випало зробити такі множення, повинні «вміти» виявляти відсутність даних і чекати їхньої появи. Тобто, потрібна синхронізація процесорів по використанню загальних даних.

Тут розпаралелювання за даними замикається з розпаралелюванням за керуванням. Можлива схема загальної для всіх процесорів програми – на рис. 14.22. Вона реалізована в прикладі для ОР типу *SPMD*.

14.3. Розпаралелювання у ОР на рівні виконавчих пристроїв

Вивчається проблема іншого рівня розпаралелювання – рівня виконавчих пристроїв. Розглядається організація конвеєрів операцій, векторних конвеєрів, виконання операцій на стеку. Досліджуються принципи динамічного розпаралелювання в багатофункціональному арифметико-логічному пристрої.

Конвеєри операцій

Виконання будь-якої операції складається з декількох послідовних етапів, кожний з яких може виконуватися своїм функціональним вузлом. Це легко показати на операціях додавання й множення. Виконання розподілу мантис (порядки віднімаються) найчастіше проводяться за допомогою віднімання з діленого дільника, зсув ліворуч отриманого залишку, нового віднімання дільника з результату зсуву і т.д. У деяких ОР знаходиться зворотна величина дільника за допомогою апроксимуючих поліномів. Потім ділене множиться на цю величину [5]. Нехай задана операція, виконання якої розбито на n послідовних етапів. Нехай t_i – година виконання i -го етапу. При послідовному їхньому виконанні операція виконується за час

$$t_{\text{посл}} = \sum_{i=1}^n t_i [c],$$

а швидкодія ЕОМ або одного процесора ОР, що виконує тільки цю операцію, складатиме

$$S_{\text{посл}} = \frac{1}{t_{\text{посл}}} = \frac{1}{\sum_{i=1}^n t_i} [\text{операцій} / c].$$

Оберемо *час такту* – величину $t_T = \max t_i$ і вимагаємо при розбивці на етапи, щоб для будь-якого $i = 1, \dots, n$ виконувалася умова $t_i + t_{(i+1) \bmod n} > t_T$. Тобто

щоб ніякі два послідовні етапи (включаючи кінець і новий початок операції) не могли бути виконані за час одного такту.

Функціональні вузли, що виконують послідовні етапи однієї операції, доцільно вишикувати в єдину конвеєрну лінію, де пристрій, що виконує деякий етап, закінчивши його для операції над одним набором даних, переходив би у наступному такті до виконання цього ж етапу тієї ж операції для іншого набору вихідних даних [4]. Наприклад, на рис. 14.23 наведено конвеєр виконання операції додавання.

Нехай реалізується потік команд одного процесора або існує доступ до цього пристрою декількох процесорів так, що в кожному такті можливе завдання на виконання додавання нової пари чисел.

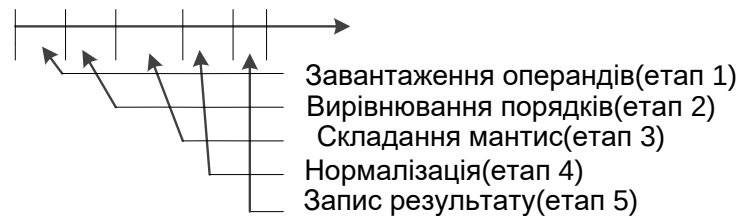


Рис. 14.23. Виконання операції додавання на конвеєрі

Тоді часова діаграма роботи конвеєра може мати вигляд, що наведено на рис. 14.24.

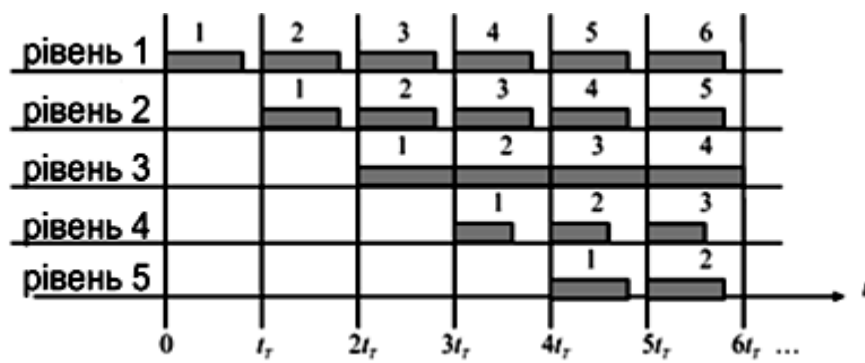


Рис. 14.24. Схема заповнення конвеєра

Максимальна швидкодія процесора при повному завантаженні конвеєра становить:

$$S_{\text{кон}} = \frac{1}{t_T} [\text{операцій} / \text{с}]$$

Число n – кількість рівнів конвеєра, або глибина перекриття, тому що в кожний такт на конвеєрі паралельно виконуються n операцій. Чим більше число рівнів (станцій), тем більший виграш у швидкодії може бути отриманий.

Реальний виграш у швидкодії виявляється завжди менше, чим зазначень вище, оскільки:

1. Деякі операції, наприклад, над цілими, можуть виконуватися за меншу кількість етапів, ніж інші арифметичні операції. Тоді окремі станції конвеєра будуть простоювати.
2. При виконанні деяких операцій на певних етапах можуть вимагатися результати більш пізніх, ще не виконаних етапів попередніх операцій. Доводиться зупиняти конвеєр.
3. Потік команд породжує недостатню кількість операцій для повного завантаження конвеєра.

Векторні конвеєри. «Зачеплення» векторів

Поряд з використанням конвеєрів для обробки одиничних (скалярних) даних, використовують так звані *векторні конвеєри*, одиничною інформацією для яких є вектори – масиви даних. Застосування векторних конвеєрів визначило клас ОР – векторно-конвеєрних ОР, що сьогодні ще є основою побудови деяких супер-ЕОМ – ОР надвисокої продуктивності.

Для ефективності векторно-конвеєрних ВОР (наприклад, для підготовки алгоритмів розв'язку завдань на ОР «Електроніка-ССБІС») необхідна векторизація завдань. Це – таке перетворення алгоритму, при якому максимально виділяються (якщо не всі завдання зводяться до цього) елементи обробки масивів даних однаковими операціями. Сюди входять усі завдання, засновані на матричних перетвореннях, обробка зображень, сигналів, моделювання поведінки середовища і т.д.

В основі векторного конвеєра лежить та ж сама розбивка операції на рівні або етапи виконання, але він доповнюється засобами апаратної підтримки, що дозволяють за інформацією про вектори організувати послідовне завантаження конвеєра елементами векторів, враховуючи їх довжину [5].

Нехай необхідно виконати операцію $C = A \times B$, тобто $c_j = a_j + b_j, j = 1, \dots, N$. Нехай на регістрах НОЗП записані вектори A і B . Група регістрів відведена для результатів C . Для керування цим процесом відомі дескриптори векторів D_A, D_B, D_C , де $D^\alpha = \{a^\alpha, h^\alpha, N\}$, $\alpha = A, B, C, h^\alpha$ – крок переадресації. Якщо завантаження векторів проводиться завжди в ті самі регістри АЛП, то досить знати значення N . Може задаватися маска M довжиною N , що складається з нулів і одиниць. Кожний елемент M відповідає елементу вектора-результату C . Якщо елемент $m_j = 1$ (логічна змінна), то операція одержання c_j проводиться, а якщо ні, то відповідні елементи векторів пропускаються. Це застосовується для альтернативного розрахунку у відповідності зі значенням логічних змінних.

Нехай операція множення виконується за три етапи. Тоді можна представити часову діаграму одержання N результатів при припущенні про призначення функціональних пристроїв (рис. 14.25).



Рис. 14.25. Множення векторів на конвеєрі

У складі АЛП може бути два й більше конвеєрні пристрої, кожний з яких спеціалізований для виконання деякої операції. Тоді можливо й ефективне «зачеплення» векторів, що ілюструється прикладом на рис. 14.26 для виконання складної операції над векторами: $D=A \times B+C$.

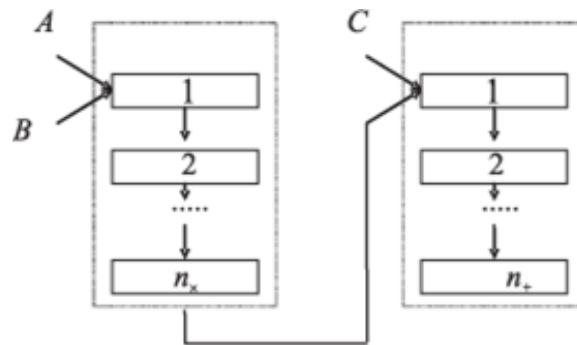


Рис. 14.26. "Зачеплення" векторів

Тут два конвеєри утворювали один, із глибиною перекриття $n = n_x + n_+$. Черговий результат множення негайно направляється на конвеєр додавання, куди паралельно направляється необхідний співмножник.

Виконання операцій на стеку

В обчислювальній техніці повсюдно використовується така структура даних, як стік. Його застосування при виконанні арифметичних і логічних операцій в арифметико-логічному пристрої (АЛП) дозволяє реалізувати безадресну систему команд, що, у свою чергу, дає можливість мінімізувати число звертань до оперативної пам'яті, здійснити буферизацію при багаторівневій пам'яті, кодувати велику кількість команд в одному слові.

Розглянемо механізми обробки інформації на стеку й формування безадресних команд.

Насамперед – про ідею застосування безадресних команд.

Розглядаючи структуру команд, ми можемо відзначити, що практично команди бувають одне-, двох-, трьохадресні. Трьохадресна команда, як правило, по закінченості відповідає одному оператору, тому що відображає дію над

двома операндами й результат. Двоадресна команда – це така команда, де одна з адрес мається на увазі (наприклад, акумулятор або суматор), або результат направляється по одній із зазначених адрес. Те ж відносно одноадресної команди: по числу адрес – число звертань до пам'яті, не вважаючи зчитування самої команди.

Стек являє собою множину послідовно пронумерованих регістрів НОЗП або комірок ОП, що постачається *показчиком вершини стека ПВС*, у якому автоматично при записі й зчитуванні встановлюється номер (адреса) останнього зайнятого регістру – *вершини стека*. При записі в стек (завантаження стека) слово пишеться в наступний по номеру регістр, а ПВС збільшується на одиницю. При зчитуванні отримується слово з адресою, зазначеною в ПВС. Потім ПВС зменшується на одиницю. Таким чином, у стеці реалізується правило «останній прийшов – перший вийшов».

Адреси пам'яті фігурують тільки в командах завантаження стека й запису зі стека до пам'яті [5].

Значення виразів можна обчислювати повністю безадресним способом. Команда, що складається тільки з коду операції, отримує зі стека один або два операнди, виконує операцію й заносить результат до стеку. На рис. 14.27 наведено приклад перетворення стека при виконанні трьох команд програми.

Обчислення з використанням стеків зручно задавати, описувати й програмувати за допомогою *польського інверсного (бездужкового) запису* арифметичних виразів *ПОЛІЗ* (запропонував польський математик Лукашевич в 1929 р.).

Запис проводиться за правилом «читаємо арифметичний вираз з ліва на право і послідовно один за одним виписуємо операнди, що зустрічаються». Як тільки з'ясується, що всі операнди деякої операції виписані, записуємо знак цієї операції, умовно враховуємо замість операндів і знака операції потенційне ім'я її результату.

Це може породити можливість запису нової операції; або ж будемо продовжувати виписувати операнди, якщо умовно врахований результат не дозво-

ляє записати знак операції. Запис на ПОЛІЗ складається з ланцюжків імен і ланцюжків операцій.

Наприклад, $y := (k + (l - mn))(a - b) \rightarrow yklmn \times - + ab - x :=$

УВС:

2

 УВС:

3

 УВС:

2

 УВС:

1

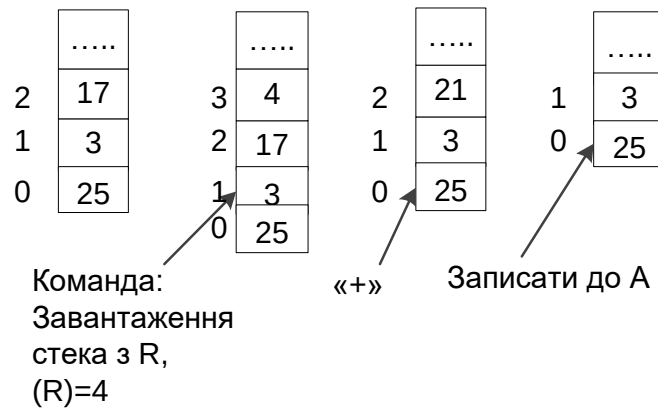


Рис. 14.27. Виконання команд на стеку

Послідовність символів у ПОЛІЗ може розглядатися як програма обчислення значення даного виразу, якщо під іменами розуміти команди завантаження стека (крім випадку, коли за ім'ям впливає знак операції $:=$, тоді це ім'я визначає запис зі стека), а під знаками операцій - безадресні команди, що містять тільки коди операцій. Зручно ввести команду виду $Зп\ Y$ – запис зі стека за адресою Y . Тоді ми одержимо остаточний вигляд програми: $klmn \times - + ab - \times 3nY$ (14.1).

Не будемо відображати ПВС, інтерпретуючи стек магазинною пам'яттю. Тоді схема виконання даної програми зводиться до наведеної на рис. 14.28.

Узагальнена процедура динамічного розпаралелювання в багатofункціональному АЛП

У багатопроцесорному обчислювальному комплексі «Ельбрус-2» використовується динамічний розподіл робіт між виконавчими пристроями АЛП. При цьому успішно дозволяється протиріччя між безадресною системою команд, що припускає їхнє виконання на стеку, і можливістю одночасного завантаження багатьох виконавчих пристроїв.

При ближчому розгляді стек представляється суцільно послідовнісною структурою. Робота зі стеком, як через «вузьке горло», проводиться з використанням його вершини, і у випадку багатофункціональних АЛП (тобто АЛП, що складаються із набору декількох спеціалізованих по операціях виконавчих пристроїв) представляється проблематичною. Здавалося б, стек по своїй природі не призначений для розпаралелювання.

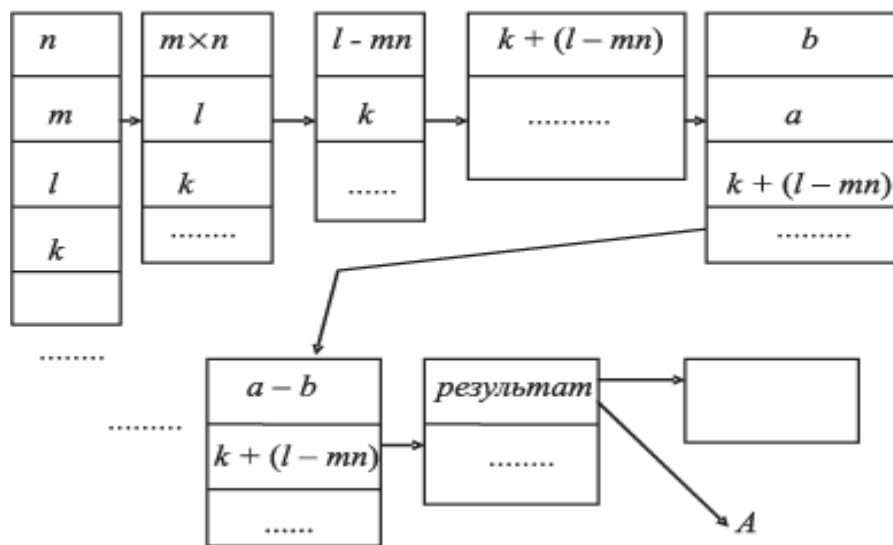


Рис. 14.28. Виконання програми на стеку

Формування потоку трьохадресних команд

В ВП АЛП в основному виконуються двомісні операції. Для виконання такої операції треба знати код операції, адреси двох операндів (як правило, – у НОЗП) і адреси результату. Тобто, АЛП в загальному випадку оперує трьохадресними командами виду $\theta A_1 A_2 A_3$.

Для ефективного використання багатофункціонального АЛП треба в кожний момент мати в розгляді велику кількість таких команд, щоб на їхній основі проводилося повне завантаження завданнями й паралельна робота усіх ВП.

Виходить, треба мати деякий буфер команд АЛП (рис. 14.29), у якому формується й підтримується в достатній кількості множина команд даної структури. Готові до виконання команди із цього буфера призначаються на ВП. Бу-

фер «знизу» поповнюється пристроєм керування новими командами замість призначених для виконання.

Такий буфер і є «вікном перегляду», через яке проходять виконавчі команди, сформовані на основі виконуваної програми, m - максимальне число команд у «вікні перегляду». Буфер може заповнюватися ПК після базування й індексації команд.

Вище говорилося, що стек максимально знижує число звертань до ОП. Крім того, ПОЛІЗ – природній і простий результат трансляції з будь-якої алгоритмічної мови. Однак виконання операцій на стеку полягає в послідовному перетворенні його вершини.

Це, здавалося б, виключає паралелізм, але й на ПОЛІЗ він видний. Наприклад, по наведеному вище запису можна виділити перші паралельні операції рахунку: $klmn \times - + ab - \times 3_n Y$ - дві двомісні операції множення й віднімання, якими можна б було завантажити два ВП.

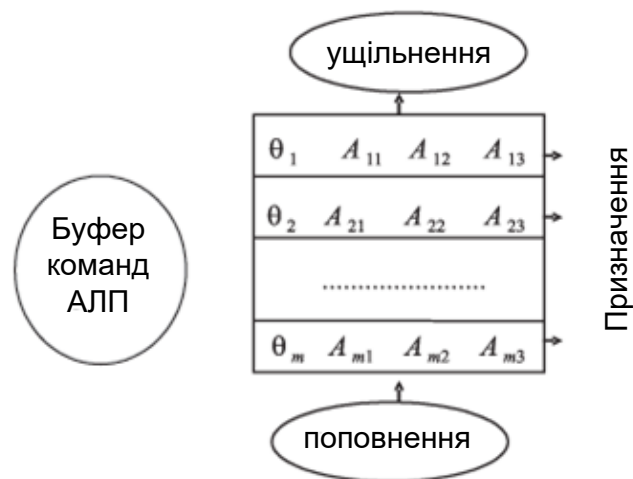


Рис. 14.29. Обробка буфера команд АЛП

Виходить, щоб розпаралелити виконання програми на стеці, треба виконати проміжний переклад – динамічно, у процесі виконання програми.

Тобто, від безадресної системи необхідно перейти до системи програмування, що використовує адреси. Тоді команди будуть мати вигляд тих процедур, про які говорили вище.

Для цього передбачається, що стік формується не на основі даних, як вище, а на основі їх адрес в НОЗП, поза стеку.

Тобто. дані й проміжні результати розташовуються в регістрах НОЗП, а їх адресами завантажувється стек; у стеку представлені не дані, а їх адреси.

Такий стек називається *адресним стеком*. Розташування ж даних (включаючи проміжні результати) у НОЗП планується апаратно, динамічно так, щоб оптимально використовувати обмежений обсяг НОЗП (або області, наданої йому) при досягненні максимального паралелізму. Швидкі регістри НОЗП є ресурсом, що розподіляється [4]. Подібну загальну схему роботи АЛП наведено на рис. 14.30.

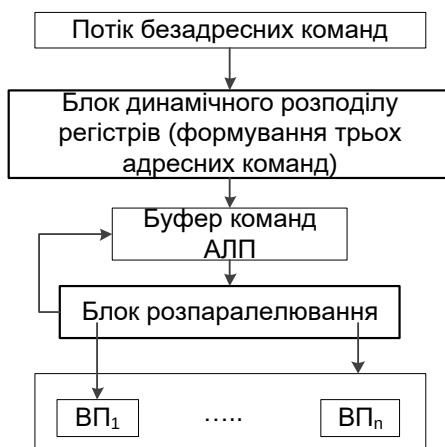


Рис. 14.30. Схема розпаралелювання виконання команд

Розглянемо роботу *блока динамічного розподілу регістрів*.

Він обробляє два списки адрес регістрів НОЗП: *адресний стек* і *список вільних регістрів*. Раніше ми бачили, що в програмі ПОЛІЗ є три види команд: завантаження стеку, виконання операції, запис зі стека. Кожна із цих команд породжує трьохадресну команду в такий спосіб.

Завантаження стеку породжує трьохадресну команду, у якій адреса величини, що зчитується до пам'яті, вказується по першій адресі команди, що формується. По третій адресі вказується перша адреса зі списку вільних регістрів. Ця адреса утворює нову вершину адресного стеку. (Тобто регістр із цією адресою утворює нову вершину стека, на якому проводиться розрахунок.)

Команда виконання операції породжує трьохадресну команду, де одним з операндів є вершина стеку. Якщо операція одномісна, адреса, що стоїть у вершині адресного стеку, стає першою адресою команди. Якщо операція двомісна, то адресою з вершини адресного стека стає друга адреса команди, що формується, а першою адресою стає адреса, що становить другий рівень адресного стеку. Третя адреса береться зі списку вільних регістрів. Використані адреси операндів виключаються з адресного стека й переводяться в кінець списку вільних регістрів. Адреса регістру, що становить третю адресу сформованої команди, тобто адресу результату, включається в адресний стек як його нова вершина.

Запис зі стека породжує трьохадресну команду, де перша адреса – адреса з вершини адресного стеку, третя адреса – адреса запису (у НОЗП або ОП). Використана адреса з вершини адресного стека виключається із цього стека й записується в кінець списку вільних регістрів.

Приклад.

$$klmn \times - + ab - \times 3_n Y$$

$$A := (a + b) : (m \times n) \rightarrow \overset{\text{ПОЛІЗ}}{Aab + mn \times} \overset{\text{програма}}{:= \rightarrow} ab + mn : 3nA$$

Адресний спосіб розпаралелювання

Продемонструємо загальний підхід при реалізації *адресного розпаралелювання*, що використовується після формування трьохадресних команд. Нехай у буфері команд АЛП, або в «вікні перегляду», формується потік трьохадресних команд, як розглядалося вище: $\{\theta j a_j A_{j_2} A_{j_2}\}$.

У таблиці 14.1 наведено переклад програми з безадресної системи команд у трьохадресну.

Очевидно, що одночасно або з перекриттям у часі можуть виконуватися такі дві команди, кожна з яких не використовує в якості операнда результат виконання іншої. Якщо ж команда змінює значення деякої величини, то попередні

команди у «вікні перегляду» повинні раніше виконання даної команди встигнути використати попереднє значення. При цьому повинна дотримуватися послідовність присвоювання значень однієї величині.

Таблиця 14.1.

Переклад програми з безадресної системи команд у трьохадресну

N _k	Адресний стек	Список вільних регістрів	Команди, що формуються			
			θ	A ₁	A ₂	A ₃
1		r ₁ , r ₂ , r ₃ , r ₄ , r ₅ , r ₆	Зч	<a>		r ₁
2	r ₁	r ₂ , r ₃ , r ₄ , r ₅ , r ₆	Зч			r ₂
3	r ₂ , r ₁	r ₃ , r ₄ , r ₅ , r ₆	+	r ₁	r ₂	r ₃
4	r ₃	r ₄ , r ₅ , r ₆ , r ₁ , r ₂	Зч	<m>		r ₄
5	r ₄ , r ₃	r ₅ , r ₆ , r ₁ , r ₂	Зч	<n>		r ₅
6	r ₅ , r ₄ , r ₃	r ₆ , r ₁ , r ₂	×	r ₄	r ₅	r ₆
7	r ₆ , r ₃	r ₁ , r ₂ , r ₄ , r ₅	:	r ₃	r ₆	r ₁
8	r ₁	r ₂ , r ₄ , r ₅ , r ₃ , r ₆	Зп	r ₁		<A>

Нехай α_j – логічна змінна (ознака), відповідно до можливості призначення j-ї команди для виконання. Тоді в процесі призначення команд на ВП j-а команда може бути призначена для виконання у відповідності зі значенням істинності наступних предикатів:

$$\begin{aligned}
 \alpha_1 &= 1 \\
 \alpha_j &= (A_{j1} \neq A_{13}) \wedge (A_{j12} \neq A_{13}) \wedge (A_{j1} \neq A_{23}) \wedge (\dots) \wedge \\
 &\wedge (A_{j1} \neq A_{j-13}) \wedge (A_{j2} \neq A_{j-13}) \wedge (A_{j3} \neq A_{11}) \wedge (A_{j3} \neq A_{12}) \wedge \dots \wedge \\
 &\wedge (A_{j3} \neq A_{j-11}) \wedge (A_{j3} \neq A_{j-12}) \wedge (A_{j3} \neq A_{j-13}) \\
 j &= 2, \dots, m.
 \end{aligned}$$

При наявності незайнятих ВП необхідної спеціалізації або при наявності вільних регістрів у буферах цих ВП, команди з ознакою $\alpha_i = 1$ призначаються для виконання.

Призначені команди забезпечуються ознакою призначення β_i і не виключаються з «вікна перегляду» до сигналу ВП про завершення виконання. Це необхідно для виключення передчасного початку виконання інших команд, що використовують результат даної.

Легко побачити, що в кожній з команд, що одночасно виконуються на різних ВП, у цьому випадку використовуються адреси НОЗП, яких немає в жодній з інших команд.

Тобто одночасно працюючі ВП використовують непересічну множину адрес. Це виключає конфлікти звертання до реєстрів НОЗП, прискорює роботу. У такому випадку говорять, що ВП *повністю сегментовані*. Хоча в загальному випадку роботи на загальних обчислювальних ресурсах (вирішальних полях) це не обов'язково.

14.4. Паралельна обробка стека й статичне розпаралелювання у вирішальному полі

Підстеки і їх взаємодія

У процесорах супер-ЕОМ використовуються багатофункціональні АЛП, що складаються із спеціалізованих по операціям виконавчих пристроїв. Широке застосування мікропроцесорів дозволяє реалізувати в складі АЛП *вирішальні поля* на основі універсальних виконавчих пристроїв, які можуть виконувати послідовності команд або цілі процедури. Розглянемо можливість динамічного розпаралелювання в таких АЛП при виконанні арифметичних операторів програми в безадресній системі команд процесора, що відтворює виконання робіт на стеку.

Існують шляхи узагальнення такої структури ОР на основі комплектації багатопроцесорних ОР на загальному обчислювальному ресурсі, - вирішальному полі. Така структура найбільшою мірою адекватна концепції двох основних рівнів розпаралелювання: рівня програм і рівня команд.

Процесор у звичайному сенсі виконує функції пристрою керування – керування виконанням програми. Виконавчі команди виконуються процесорними елементами (ПЕ) вирішального поля ВП, які становлять загальний обчислювальний ресурс ОР. Тобто, ВП виконує заявки процесорів на виконання операцій. При цьому порядок використання ПЕ визначається динамічно відповідно до потоку команд програми й з можливостями їх паралельного виконання.

Однак при такій структурі й універсальності ПЕ доцільно розподіляти між ними не окремі команди, а групи команд або закінчені оператори. І ухвалювати розв'язок можна на основі аналізу бездужкового запису (польському інверсному запису – ПОЛІЗ) арифметичних операторів програми, або, що те ж саме, – на основі потоку безадресних команд програми. З'ясовується, що робити розподіл (розпаралелювання) можна, минаючи проміжний переклад команд із *безадресної форми* в трьохадресну, – безпосередньо по тексту програми.

Тоді, якщо представити, що програма в ПОЛІЗ припускає її безпосереднє виконання на стеку, то, отже, проводиться розпаралелювання обробки стека. Робота одного стека зводиться до паралельної роботи декількох взаємозалежних *підстеків* даного стека. Кожний підстек реалізується на стеку виділеного для цього ПЕ. Якщо припустити, що дані перебувають у НОЗП й не підлягають переміщенню при організації спільної роботи ПЕ, то для їхньої обробки доцільно організувати в локальній регістровій пам'яті кожного ПЕ адресні стеки, як це розглядалося вище.

Процесорний елемент і його адресний стек зі своїм оточенням наведено на рис. 14.31.



Рис. 14.31. Підстек

Розглянемо арифметичний вираз $A := a - (b \times (c + d) - e : f) : (g \times h \times i)$.

Його бездужковий запис $Aabcd + \times ef : - gh \times i \times : - :=$.

Запишемо програму, зробивши очевидне перетворення, що оптимізує і скорочує кількість ланцюжків імен і операцій: $abcd + \times ef : - ghi \times \times : - 3na$.

Складемо інформаційний граф G , відповідний до порядку виконання операцій на стеку при розрахунку значення цього виразу (рис. 14.32).

Будувати цей граф будемо в порядку виконання операцій. Спочатку зобразимо вершини a, b, c, d відповідно до виклику їх до стеку. Потім зобразимо вершину $+$, що відповідає додаванню c і d . Потім $-$ вершину $\times$, що відповідає множенню b на $c+d$. Ланцюжок операцій закінчився, тож зобразимо вершини e і f і т.д. У результаті послідовних дій розвернеться граф G , що ілюструє паралельну структуру алгоритму розрахунку значення даного виразу. Виділені підструктури, які можуть виконуватися паралельно.

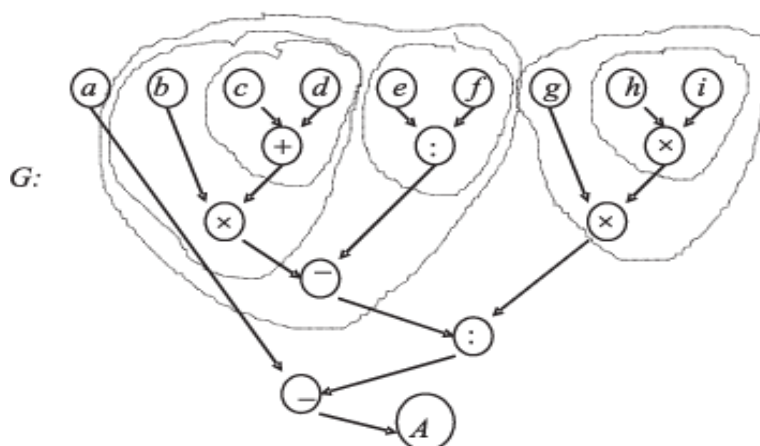


Рис. 14.32. Граф-схема розрахунку арифметичного виразу

Важливий висновок на основі аналізу графа: можна незалежно (і паралельно) витягати з пам'яті величини за всіма іменами, що становлять ланцюжок імен у ПОЛІЗ. Тобто їх можна завчасно зчитувати в будь-якій послідовності, у тому числі й паралельно.

Однак при можливості послідовного звертання до пам'яті кращим є аналіз ланцюжків імен з ліва на право, а кожного ланцюжка – з права на ліво, тому що використання величин в операціях проводиться з права на ліво. (У нижченаве-

дених прикладах завантаження стеків проводилося традиційно.) Тоді розів'ємо стековий механізм, що дозволяє робити розпаралелювання обчислень.

За числом ланцюжків імен (і операцій) у ПОЛІЗ сформуємо декілька підстеків, кожний з яких має свою вершину. Підстеки, сформовані для розрахунку одного виразу, упорядковані так, що кожний з них може мати *посилання* на свій *правий* підстек (крім останнього).

Лівий стосовно даного підстек утворюється ланцюжком імен, що перебуває в бездужковому записі безпосередньо лівіше ланцюжка імен, що відповідає даному підстеку. Кожному підстеку ставитися у відповідність наступний за ланцюжком імен ланцюжок операцій і дозволяється його виконання. На рис. 14.33. а показані сформовані підстеки і їх початкове завантаження (не за стековим принципом – послідовно через вершину, а по можливості паралельно або з перекриттям у часі).

Визначимо можливий варіант реалізації: підстеки можуть заповнюватися не відповідними величинами, а їх адресами в НОЗП. Тобто традиційні прийоми використання КЕШ-пам'яті повинні бути застосовані тут.

Взаємозв'язок підстеків впливає з нерівності довжин ланцюжків імен і відповідних їм ланцюжків операцій. Можливі ситуації:

1. *підстек виродився* у свою вершину, але ланцюжок операцій виявився не вичерпаним при тому, що перша із цих операцій – двомісна; такий підстек назвемо *не повністю виродженням*;

2. ланцюжок операцій вичерпався, але на підстеці міститься одна або більше величин (або їх імен, адрес), – такий підстек назвемо *виродженням*; якщо на підстеці перебуває єдина величина, що складає вершину, то такий стек назвемо *повністю виродженням*.

Якщо даний підстек не повністю *вироджений*, то наступна операція можлива над вершиною цього підстеку й вершиною лівого підстека, якщо той є *виродженням* або повністю *виродженням*.

Якщо лівий підстек також не повністю *вироджений* або виконання на ньому можливих операцій не закінчилося, то даний підстек перебуває в стані

очікування. Для виконання операції над вершинами двох підстеків (якщо правий підстек є *виродженим*) правий підстек переводиться на даний підстек так, що його вершина стає новою вершиною даного лівого підстека. Посилання правого підстека замінює собою посилання лівого підстека. Таким чином, на основі двох підстеків формується один і з'являється можливість подальшого виконання операцій з ланцюжка.

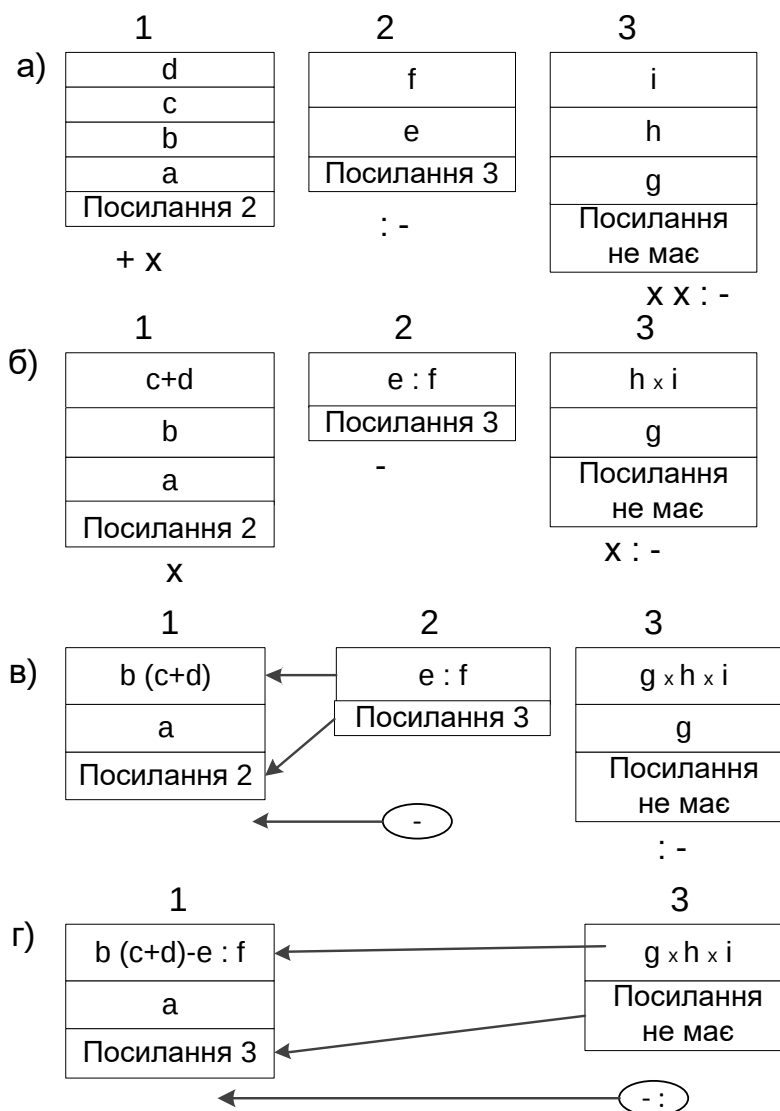


Рис. 14.33. Взаємодія підстеків

При правильному бездужковому записі кожний не повністю підстек дочекається можливості продовження виконання операцій за рахунок *виродженого* або повністю *виродженого* лівого підстека. Він, у свою чергу, перетвориться

у вироджений або повністю вироджений і поповнить лівий не повністю вироджений, якщо такий є. Остаточне значення даного виразу утворюється у вершині єдиного, що залишився повністю виродженим підстеку, що не містить посилення на правий підстек.

Розглянемо роботу системи по крокам (тактам), для спрощення припускаючи, що кожна операція виконується за один такт.

У першому такті процесор, аналізуючи програму символ за символом, починає аналіз першого ланцюжка імен. З обчислювального ресурсу призначається ПЕ 0, і він починає формування адресного стеку у своєму локальному НОЗП. В другому такті, "запустивши" формування першого ланцюжка, процесор знаходить другий ланцюжок імен. За ним закріплюється ПЕ 1. Так як визначилося місце обробки цього, правого для попереднього ланцюжка імен, ланцюжка, то процесорному елементу 0 повідомляється посилення на свій правий підстек – *Посилання 1*. У цьому ж такті в адресний стек ПЕ 0 завантажується адреса е. Однак процесор зафіксував той факт, що він при аналізі програми ввійшов в оператор if-then-else на першому лексикографічному рівні.

У третьому такті ПЕ 0 завантажує у свій стек адреси f, ПЕ 1 завантажує адреси l, а процесор знаходить наступний ланцюжок, за яким закріплює ПЕ 2. Посилання на цей процесорний елемент він повідомляє ПЕ 1, а ПЕ 2 завантажує у свій стек єдину адресу с.

У четвертому такті ПЕ 0 завантажує у свій стек ланцюжок операцій. Такі ж дії виконує й ПЕ 1. За першим ланцюжком імен оператора then закріплюється ПЕ 3. Тоді ПЕ 2 одержує посилення поки тільки на цей процесорний елемент, на якому перебуває один з його правих підстеків. ПЕ 3 завантажує адресу l до свого адресного стеку. Однак процесор фіксує той факт, що він знову входить в оператор if-then-else на іншому лексикографічному рівні.

У п'ятому такті процесор аналізує оператор then. За першим (єдиним) його ланцюжком імен закріплюється ПЕ 4. Він завантажує у свій адресний стек адресу А. ПЕ 3 завантажує у свій стек адресу g і одержує посилення на ПЕ 4.

ПЕ 0 і ПЕ 1 виконують операції зі своїх ланцюжків операцій. ПЕ 2 перебуває в стані очікування.

У шостому такті процесор приступає до аналізу першого ланцюжка імен оператора `else` на іншому лексикографічному рівні. За цим ланцюжком імен закріплюється процесорний елемент 5, який завантажує у свій адресний стек адресу А. При цьому ПЕ 4 не тільки не одержує посилання на правий підтек, але й одержує підтвердження про відсутність такого посилання. Це необхідно, тому що значення А є одним з альтернативних значень арифметичного виразу. ПЕ 0 продовжує розрахунок. Так як у п'ятому такті на стеці ПЕ 1 виконання всіх можливих операцій закінчилося, а ПЕ 2 перебував у стані очікування саме цього, то у відповідності зі своїм посиланням ПЕ 1 переводити адресний стек ПЕ 2 у вершину свого адресного стеку. Своє посилання він заміняє посиланням ПЕ 2. Звідти ж він бере ланцюжок операцій. (У загальному випадку він *поперед*у доповнює свій ланцюжок операцій). ПЕ 2 звільняється, тобто переводиться в загальний ресурс системи. ПЕ 3 завантажує у свій адресний стек адресу h. ПЕ 4 завантажує ланцюжок операцій, формуючи тим самим не повністю вироджений підтек.

У сьомому такті процесор закінчив аналіз оператора `else` на іншому лексикографічному рівні й приступає до аналізу оператора `else` на першому лексикографічному рівні. За ланцюжком імен цього оператора закріплюється ПЕ 6. У його адресний стек завантажується адреса с. ПЕ 5 завантажує у свій адресний стека адресу l. Так як на його підстеці вважається інше альтернативне значення арифметичного вираження, то підтверджується відсутність посилання на правий підтек. Так як на двох лексикографічних рівнях визначилися підстеки, на яких виконуються оператори `else`, то визначилися й інші посилання в ПЕ 1 і ПЕ 3. ПЕ 3 вводить ланцюжок операцій.

Нехай при виконанні процесорним елементом 1 операції відносин у *восьмому такті* рівність виявилася справедливою. Тоді результат розрахунку значення арифметичного оператора визначається результатом виконання оператора `then` на першому лексикографічному рівні. Тобто посилання на два альтерна-

тивні оператори повинні визначитися відповідно до вибору, – на ПЕ 3. Це й проводиться в даному такті. Однак у цьому ж такті ПЕ 6 ще продовжує формування інформації для розрахунку; йому процесор повідомляє про відсутність посилання на правий підстек, тому що це єдиний підстек розрахунку альтернативного оператора *else* на першому лексикографічному рівні. ПЕ 5 завантажує собі ланцюжок операцій.

Так як адресний стек ПЕ 1 став повністю виродженим, то його лівий підстек, що очікує – у ПЕ 1 у *дев'ятому такті* переводить його до себе, точніше, ті посилання, що від нього залишилися, на ПЕ 3. ПЕ 2 звільняється й переходить в ресурс системи. Тепер у ресурс системи переводиться й ПЕ 6.

ПЕ 3 виконує операцію відношення. ПЕ 4 перебуває в стані очікування, а ПЕ 5 виконує чергову операцію зі свого ланцюжка операцій.

У *десятому такті* припустимо, що в результаті виконання операції відношення процесорним елементом 3 виявилася справедливою нерівність, що означає необхідність виконання оператора *else* на іншому лексикографічному рівні.

Тим самим визначається значення посилання ПЕ 3 на правий підстек. ПЕ 4 доповнює загальний обчислювальний ресурс системи.

Так як ПЕ 5 містить не повністю вироджений підстек, що очікує того, хто його підхопить для подальшої обробки, то у відповідності зі своїм посиланням ПЕ 3 у *одинадцятomu такті* переведе на свій адресний стек замість адресного стеку й поля посилання, а також ланцюжок операцій - процесорного елемента 5. ПЕ 5 доповнює обчислювальний ресурс системи.

У *дванадцятomu такті* повністю визначилося завершення розрахунку процесорним елементом 0.

Розглянута на принциповому рівні схема розподілу робіт у вирішальному полі ОР містить ряд напрямків розвитку, що усувають її недоліки.

Її головним недоліком є прихильність до способу формування виразів польського інверсного запису. Розмноження "маленьких" паралельних стеків (підстеків) у системі, крім залучення надвеликої кількості процесорних елемен-

тів до розрахунку, з очевидністю збільшує накладні витрати на керування їх роботою. Хотілося б мати більше можливостей керування деталізацією робіт, що розподіляються. Отже, програма повинна бути попередньо оптимізована.

На шляху оптимізації програми на ПОЛІЗ, тобто в безадресних командах, видно два рівні.

Перший з них припускає "укрупнення" стеків на основі алгебри їх перетворень.

Так, у нашому прикладі явно проглядається доцільність заміни

$$al \times c \leq \rightarrow cal \times > .$$

Ця заміна дозволяє використовувати один стек замість двох.

У першому прикладі ми згадали очевидну заміну

$$gh \times i \times \rightarrow ghi \times \times .$$

Вона також виключає зайве "подрібнювання" робіт.

Однак таких перетворень, що приводять до укрупнення й скорочення числа підстеків, може бути недостатньо.

Тому наступний рівень перетворень може використовувати сформовані транслятором вказівки про неподільних операторів, що виконуються на одному стеку, тобто одним процесорним елементом. Для цього можуть використовуватися операторні дужки, що вказують, що їх зміст виконується на стеку одного ПЕ. Це може зробити зайвими навіть ті перетворення з укрупнення підстеків, про які ми говорили тільки що.

Наприклад, програма розрахунку значення арифметичного виразу з умовами, які ми розглядали вище, може після оптимізації мати вигляд:

$$aef : + \quad \text{if}(al \times c \leq) \quad \text{then} \quad \text{if} \quad lgh \times \quad \text{then} \quad A \times \\ \text{else} \quad Al \quad + \times \text{else} \quad ch16 \quad + \times \times 3nY$$

Тут виділена конструкція в таких операторних дужках.

Хоча наші приклади виявилися не настільки багатими для демонстрації подібних укрупнень робіт, складні програми обчислювального характеру потребують такого укрупнення безсумнівно.

Статичне розпаралелювання в АЛП. VLIW- і EPIC-архітектури

З розвитком архітектурної складності ОП і із проектуванням великих інтегральних схем з'явився новий напрямок у розвитку архітектур – використання *принципу програмного керування кожним тактом машини*.

Це означає, що програма складається з команд, що задають у такті виконання інструкції кожному ВП АЛП. Команда зображується «довгим командним словом» (звідси назва VLIW-архітектура, *Very Long Information Word*). У ньому передбачені позиції, що відповідають кожному ВП.

Вони вказують, яку роботу повинен почати виконувати кожний ВП. При цьому враховується стан даного ВП, тимчасові співвідношення для раніше ініційованого вступу на нього інформації і т.д.

Тобто, стан усіх пристроїв процесора планується програмно, статично, не залишаючи елементів динамічного планування, як розглядалося вище.

Однак стан не всіх пристроїв може бути точно передбачений на кожний момент часу. Коли необхідне звертання до ОП, через непередбачуваність ефекту розшарування пам'яті й можливості конфліктів при одночасному обігу й інших процесорів використовуються елементи динамічного планування звертань до ОП.

Кожний регістр буфера стеку й буфера масивів має біт значимості, який вважається рівним нулю при видачі заявки на зчитування з ОП у цей регістр і вважається рівним одиниці при приході необхідного числа.

Тоді, якщо в команді проводиться звертання до регістру, у який ще не надійшло число, проводиться пропуск такту, виконання команди *NOP (No Operation)*. Операція повторюється неодноразово до завершення зчитування.

Транслятор повинен з випередженням планувати звертання до ОП, щоб мінімізувати число порожніх тактів.

Широко використовується безпосередня передача результатів з одних ВП на інші. Для цього результати операцій зберігаються кілька тактів у протоколі результатів роботи кожного ВП.

Однак "довгі" команди непродуктивно витрачають пам'ять, і з розвитком архітектури замінені командами змінної довжини.

У результаті проведених досліджень, у контексті розробок найбільших закордонних фірм, склалась уява про архітектуру мікропроцесора, використуваного в багатопроцесорній обчислювальній системі, на основі спільного застосування всіх існуючих способів оптимізації паралельної обробки інформації.

Однак ефективність розпаралелювання залежить від розв'язку (статичного або динамічного) завдань оптимального планування завантаження устаткування. Пошук у цьому напрямку привів до доцільності EPIC-архітектури (*Explicitly Parallel Instruction Computing*), що для користувача (транслятора) виражається як *система команд із явним паралелізмом*.

Така система команд дозволяє планувати завантаження всіх пристроїв процесора в кожному такті. Планування проводиться в статичному режимі при трансляції або програмуванні завдання й фактично полягає в побудові потактового розкладу робіт, тобто формування «довгих» командних слів (*VLIW*), у позиціях яких зазначено до виконання якої роботи повинен приступити кожний виконавчий пристрій (ВП) у даному такті.

Важливою особливістю EPIC-архітектури є можливість паралельного розгалуження у двох випадках: при виконанні команд умовного переходу й при виконанні конструкцій *if-then-else* у складі арифметичних операторів.

Умовний перехід в «традиційному» виконанні загрожує зупинкою конвеєра. В EPIC-архітектурі передбачений запуск додаткового конвеєра по команді підготовки переходу за кілька тактів до розгалуження.

Інтенсивне розгалуження у виконуваній програмі здатне привести до лавиноподібного запуску додаткових конвеєрів і необхідності статистичної оцінки достатньої їхньої кількості при обґрунтуванні засобів апаратної підтримки.

Тому другий випадок розгалуження, здійснюваного усередині лінійної ділянки програми, кращий, хоча також приводить до надмірності устаткування, що використовується при обчисленнях.

14.5. Апаратна підтримка мови користувача – основна концепція мульти-процесорних систем

Існують три рівні алгоритмічних мов: *асемблери*, *автокоди*, *мови високого рівня (МВУ)*.

Асемблери – машинно-орієнтовані мови, низького рівня, що допускають трансляцію «один в один».

Автокоди – також машинно-орієнтовані мови, що допускають конструкції, не трансльовані «один в один». (Деяка класифікація мов виключає автокоди, припускаючи, що асемблер може мати макрозасоби (такий асемблер – макроасемблер), що забезпечують його лаконічність.)

МВУ – не машинно-орієнтовані (проблемно-орієнтовані) мови, що володіють лаконічністю, більшим набором виразних засобів, що забезпечують вільну рекурсивну структуру, із вкладеними умовними виразами, із процедурною нотацією й інфіксними арифметичними виразами. МВУ – мова користувача-алгоритміста, якому немає необхідності сходити на рівень машинного представлення, у тому числі до рівня машинних команд.

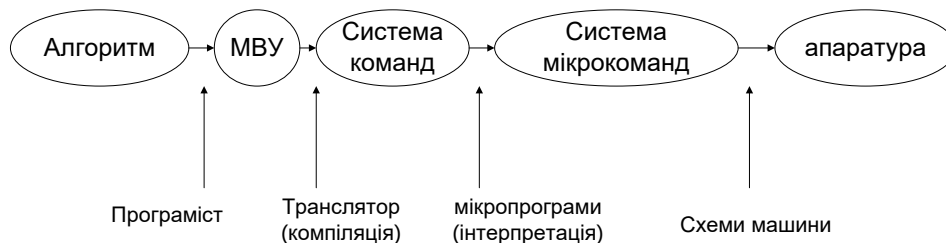
Тому сучасна ідея розробки ОР полягає в наступному: *мінімально доступний рівень мови користувача повинен відповідати мові високого рівня*. Однак раніше це було не більше ніж гасло, тому що транслятори з МВУ для відомих машин менш ефективні (по довжині програми й за часом її виконання), ніж транслятори з асемблерів.

Підвищити цю ефективність можна за рахунок апаратної реалізації конструкцій МВУ: апаратної підтримки типів даних, виконання процедур, у т.ч. вкладених і рекурсивних, циклів, організації й індексації одномірних і багатомірних масивів, виконання умовних операторів і т.д. У цьому випадку МВУ для даної ОР максимально наближається до рівня її асемблера, залишаючись для користувача мовою високого рівня.

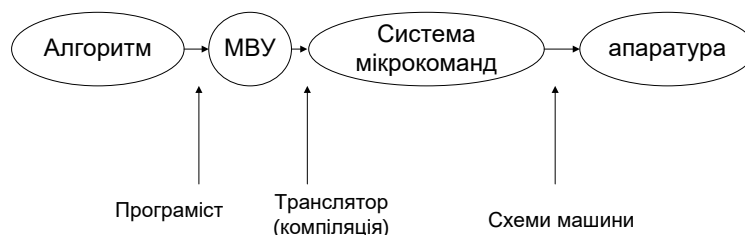
Уведення в машинну мову складних операцій, відповідних до конструкцій МВУ, служить підвищенню продуктивності.

Є інший аспект підвищення рівня вхідної мови (найнижчого рівня мови користувача): створення сімейств програмно-сумісних ОР. Сумісність на рівні асемблерів (реалізована в ЄС ЕОМ) значно стримує розвиток нових поколінь, залишаючи їх у рамках незмінних архітектурних розв'язків. Вимога сумісності на рівні асемблера-МВУ надає широкі можливості пошуку, вибору, розвитку поколінь. Кожна система може мати свою внутрішню машинну мову, обумовлену своїми архітектурними особливостями, тобто свою для кожної моделі апаратної підтримки мовних конструкцій. Вона не доступна користувачеві й проявляється тільки в об'єктному коді – коді відтрансльованої програми, до якої користувач відношення не має.

Раніше говорилося, що в основу операційних пристроїв сучасних ОР покладені мікропроцесори з досить розвиненою системою команд. Відповідно до ідей RISC-архітектури на основі цієї системи команд створюються мікропрограми системи команд ОР. Тобто користувач, що програмує на асемблері, володіє системою команд, реалізованих мікропрограмно. У цьому випадку трансляція з МВУ проводиться в систему команд ОР, а вже з неї за допомогою мікропрограм проводиться вплив на апаратуру:



Відсутність асемблерного програмування при досить розвиненій системі команд мікропроцесора – процесора RISC-архітектури дозволяє трансльовати безпосередньо в мікрокоманди, крім необхідності системи команд ОР. Вона не потрібна ні користувачеві, ні транслятору:



Така ідеологія обговорюється у зв'язку з розвитком сімейства «Ельбрус». Однак тут немає повної ясності через те, що для ефективності системи трансляції безпосередньо в систему команд мікропроцесора потрібна дійсно розвинена система його команд.

Справа в тому, що будь-яка трансляція використовує два принципи – компіляції й інтерпретації. Існує оптимальне співвідношення цих принципів при реалізації трансляції. Компіляція на мову занадто низького рівня (з якого потім проводиться інтерпретація апаратно), крім власної складності й трудомісткості, приводить до значного збільшення об'єктного коду. Треба враховувати, що в порівнянні з «ручним» програмуванням транслятор і без того дає програми, за обсягом і за часом виконання значно (у два рази) менш ефективні. Така неефективність збільшується низьким рівнем мови, на якій проводиться трансляція. Виходить, необхідна проміжна «оптимальна» мова, на якій доцільна компіляція, а з неї потім – інтерпретація. Такою мовою й може бути машинно-орієнтована асемблерна мова ОР. Це відповідає як загальній ідеї RISC-архітектури, що достатньо себе зарекомендувала, так і тому очевидному факту, що комплексування великої кількості мікропроцесорів у ОР дає нову якість, що виражається в розвиненій системі команд ОР. У такій системі команд зокрема повинні відбиватися способи організації й синхронізації паралельних процесів. Не всі команди реалізуються тільки комплектуючими мікропроцесорами, а вимагають використання й інших засобів ОР. Таким чином, на практиці доцільність повної відсутності асемблерного програмування може не підтвердитися.

Підвищувати ж рівень вхідної мови користувача за рахунок розвитку системи команд ОР, тобто забезпечення апаратно-мікропрограмної підтримки мовних конструкцій, доцільно в кожному разі.

Підтримка типів – тегова архітектура

Розглянемо лише ті засоби підтримки мовних конструкцій, які визначають концептуальний підхід, реалізований у сімействі «Ельбрус».

МВУ за способом роботи з типами даних діляться на два класи: з динамічними й статичними типами, або на *динамічні* (не типізовані) і *статичні* (типізовані) мови. У статичних мовах тип даних задається один раз на весь час їх існування. (У Паскалі – в описі, для динамічних змінних – в описанні посилань). У динамічних мовах вказівка типу супроводжує кожне входження даних у текст програми й, загалом кажучи, може мінятися протягом життя цих даних. Наприклад, одній величині можна в одному місці програми привласнити тип ціле, а в іншому – бітовий набір. (Скажемо, при моделюванні ОР ми можемо за допомогою операцій цілочисельної арифметики обчислити адреси, потім треба зробити структурне дослідження цієї адреси, наприклад – виділити групу розрядів цієї адреси і т.д.)

Введення тегів, що не виключає застосування статичних МВУ, визначило поняття *тегової архітектури*.

Теги забезпечують:

1. апаратне керування алгоритмами обробки даних. Дані на вході ВП розпізнаються за типами, і ВП налаштовується на вид обробки: наприклад, додавання цілих, додавання дійсних і т.д.;
2. спрощення трансляції, тому що скорочується кількість кодів операцій;
3. семантичний контроль обчислень: можлива сигналізація (переривання) у тому випадку, коли виконується операція, не відповідна до типу даних (наприклад, операція, яка не повинна проводитися над адресною інформацією);
4. контекстний захист даних: а) кожна процедура виконується над відведеною їй областю пам'яті; б) відомі області пам'яті тих процедур, всередині яких вона запущена, і с) усе це утворює *адресний контекст процедури*, що виконується. Він визначається сукупністю доступних їй дескрипторів масивів (сегментів), що описують області доступної пам'яті. Дескриптори, як адресна інформація, мають теги. Апаратура здійснює контроль над тим, щоб над адресною інформацією не виконувалися дії, що приводять до розширення адресного простору, тобто до порушення захисту пам'яті. Контекст може змінюватися тільки при процедурних переходах і повністю визначається міткою процедури.

Мітка – теж адресна інформація, і дії над нею також контролюються апаратурою. Змінити мітку не можна, по ній можна тільки викликати нову процедуру.

Таким чином, апаратний контроль адресної інформації за допомогою тегів виключає можливість звернення за межі того адресного простору, який було виділено даній процедурі. Ця «попутна» властивість тегової архітектури виключає необхідність інших засобів захисту даних, хоча статус захисту може вказуватися для потреб користувача або ОС у дескрипторах масивів.

Пункти 3 і 4 визначають *типовий контроль*, здійснюваний за допомогою тегів;

5. реалізацію формування параметрів процедур «за посиланням» і «процедурою». Адреси для команд запису й зчитування задаються операндами адресного типу. Формування параметрів процедур «за посиланням» проводиться в наступному випадку: якщо результат зчитування – адресна інформація, то операція зчитування автоматично повторюється за новою адресою, і так доти, поки не буде зчитано значення іншого типу.

Другий випадок – формування параметрів процедур «процедурою»: якщо результат зчитування – ім'я процедури, то вона автоматично запускається, і результат її виконання розглядається як зчитане значення.

Стековий механізм виконання процедур

Для побудови складних програмних комплексів і можливості розпаралелювання завдання представляється у вигляді одного або декількох взаємодіючих процесів.

Процеси – це логічно завершені, переважно значні за обсягом роботи (програми), на які можна й доцільно розбити розв'язуване завдання (виконуваний алгоритм).

У МОК одночасно можуть виконуватися, тобто перебувати в активному стані, кілька процесів, не більше числа процесорів. Процес відображений елементарною робочою програмою, сформованою зі звертань до *процедур*. Можливий будь-який спосіб супідрядності (вкладеності) процедур. Відтворюється виконання процедур на стеку.

Описи процедур (сегменти процедур) зберігаються окремо (рис. 14.34). Існує *словник сегментів* даної програми, сформований при трансляції. У ньому зберігаються дескриптори сегментів, у яких зазначено місце в пам'яті, обсяг і ін. Один з базових реєстрів B0 кожного процесора під час виконання програми вказує на її словник сегментів. Номер виконуваної команди перебуває в *реєстрі номеру команди (RNK)* і (на відміну від адреси комірки в традиційних EOM) складається з номера сегмента й зсуву – адреси в середині сегмента.

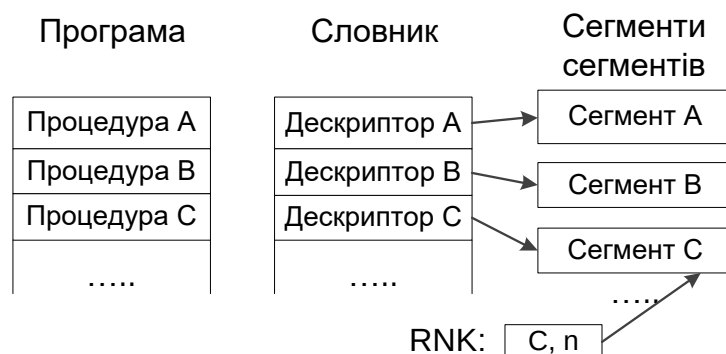


Рис. 14.34. Організація програмних сегментів

У НОЗП кожного процесора організовано *стек для даних процедур, що виконуються* (рис. 14.35) у складі процесу, що виконується – *стек процесу*.

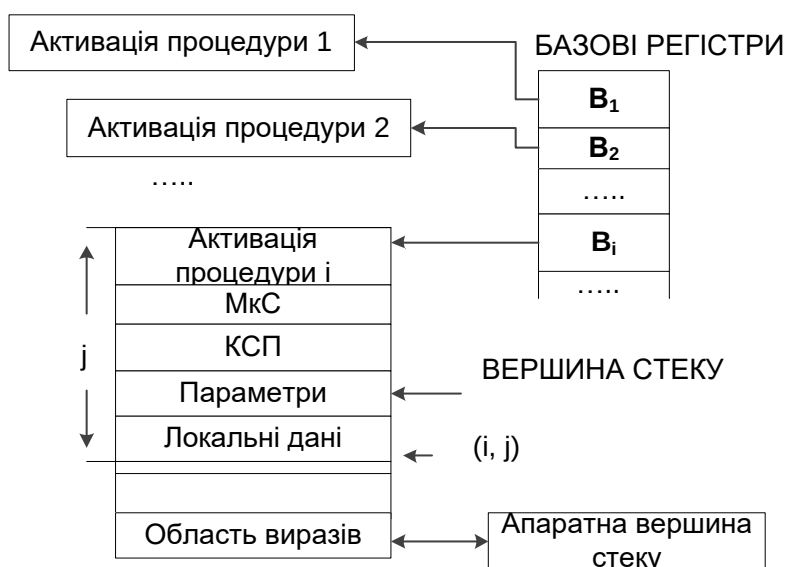


Рис. 14.35. Виконання програми на стеці

У вершині стека перебувають дані й відбивається процедура, що виконується в цей момент (ця область називається *активацією процедури*) найвищого на даний момент рівня ієрархії (вкладеності). При завершенні виконання процедури у вершині стека, відведена їй область НОЗП повертається в ресурс. При запуску ж вкладеної процедури формується нова вершина стека, тобто нова активація. Уся інформація про дані перерваних процедур як би поринає на один рівень нижче. Кожна активація починається зі сполучної інформації, що складається з *маркера стека* M_kC (у ньому посилання на M_kC процедури, що охоплює, у якій описана дана процедура) і *керуючого слова повернення* КСП (у ньому інформація для повернення в процедуру, що запустила).

M_kC утворюють *статичний ланцюжок*, КСП – *динамічний ланцюжок*.

Примітка. Процедури можуть бути описані усередині інших процедур; інформація про це міститься в M_kC . У цьому випадку M_kC утворюють *статичний ланцюжок*. Запускатися ж процедури можуть довільно з різних процедур; ця інформація втримується в КСП. Тоді КСП утворюють *динамічний ланцюжок*.

Рівень вкладеності i (*лексикографічний рівень*) відображений у базовому реєстрі B_i , що містить початок активації цієї процедури.

Адресація даних в активації процедури при її виконанні проводиться за допомогою адресної пара (i, j) , тобто адреси НОЗП $\{A_{ij} = (B_i) + j\}$.

Примітка. Процедура програмується в пам'яті «з нуля», включаючи розподіл її даних. Тоді за допомогою базування вона може вільно переміщатися по пам'яті, включаючи розташування у виділеній області НОЗП на стеці. Рівень вкладеності i (*лексикографічний рівень*) відображений у базовому реєстрі B_i – з тією ж самою адресою реєстру в НОЗП кожного процесора; він містить початок активації цієї процедури. Область виразів – це область активації процедури, де запам'ятовується стан стека АЛП, на якому виконується власно рахунок, у випадку переривань і процедурних переходів. Стек АЛП й реєстри ЦП, необхідні для відновлення рахунку процедури, утворюють *апаратну вершину сте-*

ка. Розглянемо на прикладі (рис. 14.36) ієрархічну структуру фрагмента програми, де в процедурі А (перший лексикографічний рівень, ЛР 1) послідовно використовуються процедури В і D (ЛР2), а в кожній з них послідовно запускається процедура З (ЛР3).

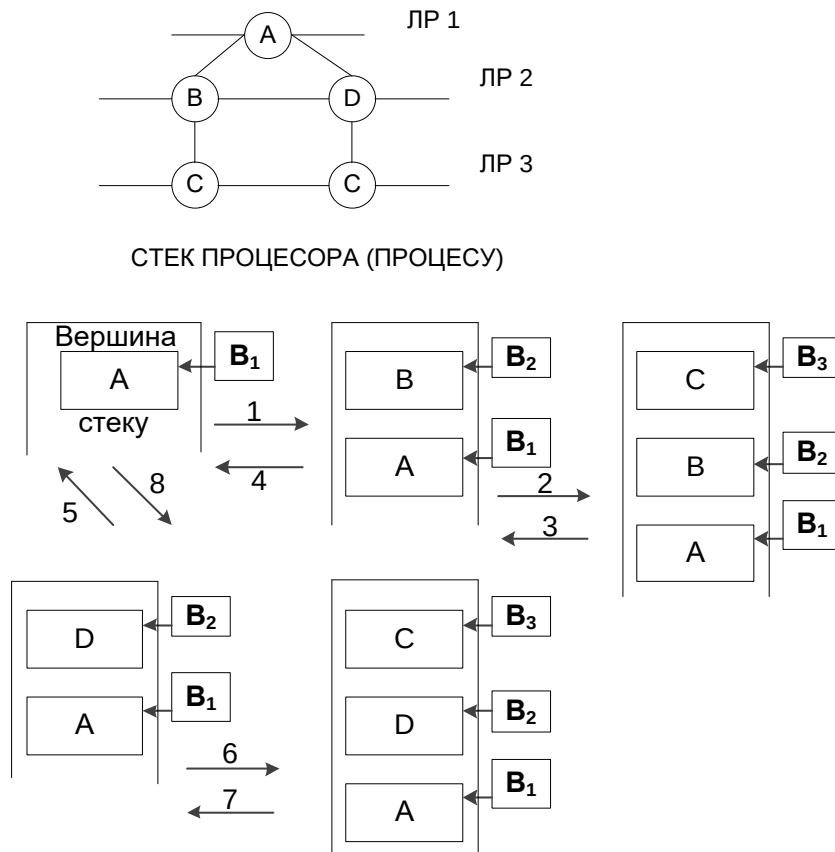


Рис. 14.36. Перетворення стека

Розглянута організація виконання процедур призводить до їх властивості повторної входимості. Та сама процедура може виконуватися одночасно на стеках різних процесорів або одного процесора, запускатися рекурсивно і т.д. Засоби ОС передбачають випадки, коли процедура або множина процедур не містяться в пам'яті, що виділяється, НОЗП, число лексикографічних рівнів перевищує допустиме і т.д.

Обробка масивів

Змінні з індексами, тобто елементи одно- і багатомірних масивів, представляють більші труднощі при трансляції й виконанні, якщо не забезпечені засобами апаратної підтримки на рівні МВУ.

Так як основна робота усередині процедури проводиться з регістрами НОЗП, то звертання до змінних з індексами зводиться до двох операцій: зчитування й запису.

Нехай оголошено тривимірний масив $A[0:P-1; 0:Q-1; 0:R-1]$. У тексті програми зустрілося звертання до елемента цього масиву $A_{[i,j,k]}$. Треба встановити адресу цього елемента. Який підхід може лежати в основі апаратної підтримки встановлення адреси елемента масиву за значеннями індексів? Важливо врахувати, що пам'ять лінійна, одномірна (пронумерована своїми адресами). При заданих транслятору значеннях P, Q, R тривимірний масив A зводиться в ній до одномірного масиву довжиною $P \times Q \times R$, розташованому починаючи з деякої адреси a , як показано на рис. 14.37.

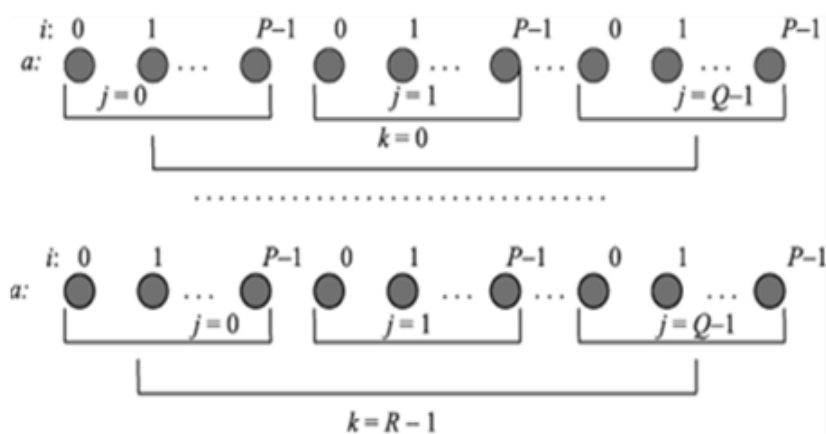


Рис. 14.37. Перетворення тривимірного масиву в лінійний

При цьому кроки переадресації по індексам становлять $h_i = 1, h_j = P, h_k = P \times Q$.

При заданих $i, j, k < A[i, j, k] \Rightarrow a + i + P \times j + P \times Q \times k$. Виходить, існує принципова можливість розрахувати адресу змінної з індексами, заданими в символах МВУ, у тому числі апаратно.

У загальному випадку для знаходження адреси індексованої змінної запускається процедура ОС. Засоби ж апаратної підтримки використовуються пов-

ністю для оптимізації обробки масивів у циклі. При цьому кількість вимірів масиву обмежено трьома.

Отже, масив займає суцільну нерозривну область пам'яті. Початок і розмір області (у цьому випадку a , PQR), а також формат елементів масиву й статус звернення (захист по запису або зчитуванню) описує *дескриптор масиву*. Зміна кожного індексу означає «проходження» по пам'яті з постійним кроком. Величини кроку по пам'яті (наприклад, як вище h , h_j , h_k) у кожному вимірі упаковані в спеціальному слові – *паспорті масиву*.

Дескриптор і паспорт масиву, а також індекси по вимірах є достатньою інформацією для звертання до елемента масиву.

Передбачається, що у виконуваному циклі параметр циклу збігається з одним з індексів масиву. Тобто від циклу до циклу необхідно змінювати адресу елемента масиву на крок переадресації по цьому індексу.

При першому виконанні циклу, тобто при першому звертанні до масиву, по дескриптору, паспорту й початковим значенням індексів обчислюються адреса необхідного елемента й крок переадресації по параметру циклу. Вони заносяться в АП.

При наступних звертаннях до масиву в цьому ж циклі з АП вилучається адреса елемента, складається із кроком переадресації й одночасно з використанням як адреси нового елемента, заноситися в АП замість адреси старого елемента. АП розрахована на зберігання інформації про шість масивів. Час обчислення адрес елементів у циклі для шести масивів – 1 такт.

Передбачений механізм автоматичного попереднього «підкачування» елементів масиву з ОП в асоціативний ЗП масивів. При цьому темп підкачування даних, як правило, перевищує темп використання.

Виконання циклів

Звичайно апаратно підтримують такі «передбачувані» оператори циклу, як *оператор циклу з параметром* у Паскалі:

$$\begin{aligned} &\text{for } i := E_1 \text{ to } E_2 \text{ do, якщо } E_1 < E_2, i = E_1, E_1 + 1, \dots, E_2 \\ &\text{for } i := E_1 \text{ downto } E_2 \text{ do, якщо } E_1 > E_2, i = E_1, E_1 - 1 \text{ dots } E_2 \end{aligned}$$

(ПАСКАЛЬ – алголоподібна мова; в Алголі цьому циклу відповідає *цикл типу арифметичної прогресії*. Циклу типу *перерахування* відповідає заголовку виду `for i:=E while B do`, де E - арифметичний вираз, B – умовний оператор. Аналог існує в Паскалі. Такий цикл реалізується за допомогою умовного переходу).

Апаратно реалізується *елемент циклу* в трьох регістрах НОЗП $\{ЛЦ, ГЗЦ, Ану\}$, де $ЛЦ$ – Лічильник Циклу (фіксує значення параметра циклу), $ГЗЦ$ – Граничне значення E_2 параметра Циклу, $Ану$ – Адреса початку циклу. Використовується команда типу $КЦ$ – Кінець Циклу, по якій $(ЛЦ)$ дорівнює з $(ГЗЦ)$; у випадку рівності $(ЛЦ):= 0$ і відбувається вихід із циклу (на наступну команду).

У випадку нерівності – $(ЛЦ):= (ЛЦ) + 1$ і безумовний перехід по $Ану$. Вхід у цикл проводиться по команді ЦИКЛ із параметрами, що задають $(ЛЦ) = E_1$ і $(ГЗЦ) = E_2$. По цій же команді фіксується адреса $Ану$ – наступної команди (або мітки початку циклу). Однак, якщо до цього $(ЛЦ) \neq 0$, то проводиться вхід у вкладений цикл. Тоді елемент циклу, що переривається, заноситися в *стек елементів циклів*. При виході із циклу проводиться, якщо необхідно, відновлення зовнішнього циклу. Так реалізується система вкладених циклів.

Прискорення умовних переходів

Непередбачені умовні переходи припиняють роботу конвеєрів. Використовуються умовні переходи з *перевагою*. Наприклад, дві модифікації одного умовного переходу за результатом порівняння двох величин $УП_1 < a > < b > C$ використовуються, коли можна передбачити, що з великою ймовірністю $a < b$; якщо очікується, що скоріше $a \geq b$, то використовується інша модифікація $УП_2 < a > < b > C$.

У цих випадках завантаження конвеєра проводиться в напрямку більш ймовірного розгалуження. У випадку протилежного напрямку розгалуження зупиняється конвеєр, відміняються відповідні дії й запускається конвеєр за іншим напрямком. Очевидно, що в циклі кращий перехід на початок.

Якщо в результаті перевірки умова керування повинна бути передана наступній команді, ця ознака знищується, виконання команд триває. Якщо повинен бути перехід за адресою, знищуються всі позначені команди (на всіх станціях конвеєра). Починається виконання розгалужень програми за адресою переходу.

Команда підготовки переходу ставиться на кілька тактів раніше розгалуження. За нею розташовується в асоціативному БК адреса переходу. Якщо такого в БК немає, то організовується підкачування в нього ділянки програми із цієї адреси. Команди з адреси переходу надходять на додатковий буфер дешифрації команд, тобто на додатковий конвеєр. Так починають виконуватися обидва розгалуження в режимі умовного виконання. При виконанні умови по самій команді переходу без затримки триває виконання потрібного розгалуження.

Пам'ять предикатів застосовується для мінімізації числа умовних переходів при виконанні умовних арифметичних виразів. Вона дозволяє зберігати значення булевих змінних, отриманих при перевірці умов. Планування альтернативних дій може виконуватися в *спекулятивному режимі*.

Це означає, що в команді може вказуватися ім'я булевої змінної. Операція може бути виконана, якщо значення цієї змінної дорівнює 1. А якщо ні, то її виконання або пропускається, або – залежно від прийнятих умов – у режимі "жужжання" очікується (наприклад, при зчитуванні) присвоювання цій змінній значення 1.

КОНТРОЛЬНІ ПИТАННЯ

1. Що розуміють під ефективністю роботи пристрою в складі ОС?
2. Назвіть два основних рівня, на яких у ОС застосовуються практичні методи розпаралелювання?
3. Які способи реалізації повного завантаження окремих виконавчих пристроїв ви знаєте?
4. Які типи архітектури ЕОМ вам відомі? Надайте коротку характеристику кожного з них.

5. Назвіть складові частини комунікаційної мережі ОС.
6. Приведіть коротку характеристику системи із загальною (розподілюваною) оперативною пам'яттю.
7. Які способи міжмодульного з'єднання вам відомі?
8. Які ОС називаються матричними? Наведіть хоча б один класичний приклад такої ОС.
9. Розкрийте суть трансп'ютерної технології.
10. Що таке «семафор»? У чому полягає суть семафорного механізму?
11. Які команди використовуються для застосування семафорів?
12. Що таке синхронізатор? Для чого він призначений?
13. Які способи розпаралелювання вам відомі?
14. Дайте визначення поняття «векторний конвеєр». Що лежить в основі векторного конвеєра?
15. Що таке стек? Що постачає покажчик вершини стека?
16. В чому полягає загальний підхід при реалізації *адресного* розпаралелювання, що використовується після формування трьохадресних команд?
17. Назвіть основні особливості EPIC-архітектури ЕОМ?
18. Які види алгоритмічних мов вам відомі? Дайте визначення кожної з них.
19. Зазначте класифікацію МВУ за способом роботи з типами даних?
20. Що вам відомо про тегову архітектуру? Що забезпечують теги?

15. RISC-ПРОЦЕСОРИ

15.1. Мікропроцесори з RISC- архітектурою

Ідеологія RISC-архітектури побудови процесорів (Reduced Instruction Set Computing – обчислення зі скороченим набором команд) складалася наприкінці 1970-х – початку 1980-х років, коли потрібні були нові ідеї для підвищення продуктивності процесорів. Висновки різних груп дослідників були узагальнені у вигляді так званого правила "80/20": 80 % часу виконання програм займає виконання 20 % команд, що входять до складу системи команд. Тобто у визначенні продуктивності процесора основну роль відіграє лише п'ята частина всіх команд, інші ж команди зустрічаються досить рідко, і час їх виконання істотного впливу на продуктивність процесора не чинить. Виходячи із цього було ухвалене рішення побудувати процесор, у якому виділена невелика група команд виконувалася б максимально швидко за рахунок її апаратної реалізації, а інші команди або взагалі видалялися із системи команд, або реалізовувалися на мікропрограмному рівні.

В результаті цього склалася ідеологія RISC-архітектури, яка опиралася на наступні *принципи*:

- набір команд скорочений до 70-100 команд (замість декількох сотень в CISC-мікропроцесорів);
- більшість команд виконується за 1 такт, і лише деякі – за декілька або навіть кілька десятків тактів;
- усі команди обробки даних оперують тільки змістом регістрів процесора, а для звертання до більш повільної оперативної пам'яті передбачені винятково інструкції виду "завантажити в регістр" і "записати до пам'яті";
- команди мають простий, чітко заданий формат;
- з набору команд виключені рідко використовувані інструкції, а також команди, що не вписуються в прийнятий формат;

— склад системи команд повинен бути зручним для застосування компіляторів з мов високого рівня, що оптимізують.

Такий підхід дозволив зменшити обсяг апаратури процесора за рахунок скорочення блоку керування приблизно в 10 раз, суттєво збільшити тактову частоту роботи процесора й знизити його тепловиділення.

Незважаючи на свою назву, основою RISC-архітектури є те, що вся обробка зосереджена тільки у внутрішніх регістрах мікропроцесора.

Так як вся обробка проходить у регістрах, відпадає необхідність у великій кількості режимів адресації операндів, а в системі команд можна застосовувати трьохадресні команди, найбільш ефективні з погляду організації обчислювального процесу й у той же час, що не мають їхнього головного недоліку - великої довжини команди.

Простий формат команди легко піддається декодуванню на відповідному щаблі роботи конвеєра. Згадаємо, що довжина команди в CISC-архітектурі IA-32 міняється в межах від 1 до 15 байт, а наявність, формат і призначення багатьох полів команди неоднозначні й визначаються структурою інших полів.

Природно, що цей підхід потребував використання в мікропроцесорі регістрової пам'яті великого обсягу (до 128 регістрів).

А для забезпечення погодженої роботи швидких внутрішніх конвеєрів і щодо повільної оперативної пам'яті в RISC-мікропроцесорах передбачається кеш-пам'ять великої ємності.

Наявність великої кількості регістрів створює гарну основу для роботи компіляторів, що оптимізують, які дозволяють ефективно використовувати всі конвеєри мікропроцесора.

Простий формат команди й орієнтація на регістрову обробку дозволили безболісно впровадити в RISC-процесорах *конвеєрний принцип обробки* інформації.

Така організація забезпечила істотне підвищення продуктивності RISC-мікропроцесорів у порівнянні з мікропроцесорами CISC-архітектури. Це приве-

ло до переваги МП даного типу в тих областях, де продуктивність була основним фактором, наприклад, у серверах.

У той же час вони не знайшли свого місця на найбільш розвиненому ринку обчислювальної техніки – ринку персональних комп'ютерів. Тому є кілька причин:

- велика вартість RISC-процесорів і систем на їхній основі: споконвічно ці процесори були орієнтовані на потужні робочі станції й сервери, тому розробляючи використовували в них рішення, занадто дорогі для персональних комп'ютерів; навіть спеціальні, "дешеві" варіанти RISC-комп'ютерів коштували набагато дорожче порівнянних з ними ПК на базі процесорів Intel через малі обсяги виробництва;

- відсутність широких наробітків в області програмного забезпечення: традиційною операційною системою для персональних комп'ютерів була DOS, до неї згодом приєдналися 16-розрядні версії Windows, під які написана величезна кількість популярних і добре знайомих користувачам програм. Різні RISC-платформи звичайно використовували несумісні між собою різновиди UNIX, для котрих існувало значно менше програм, головним чином науково-технічних (для робочих станцій) або мережних додатків (для серверів);

- RISC-процесори за своїм основним положенням мають несумісні з x86 набори команд, тому єдиним способом виконання коду x86 була емуляція, яка знижувала продуктивність від десятків до сотень відсотків, що зводило нанівець швидкісні переваги RISC-процесорів;

- відсутність інтересу до проникнення на цей ринок у самих виробників RISC-систем: багато "серйозних" фірм таких як DEC або Sun вважали, що немає потреби здешевлювати свої RISC-станції, тому що користувачі однаково виберуть їхні системи через очевидні технічні переваги.

Розвиток архітектури RISC-мікропроцесорів йшов по декільком напрямкам. За рахунок підвищення технологічних можливостей виробництва мікропроцесорів зм'якшилися вимоги до складу й форматів використовуваних команд.

У цей час їх системи команд розширилися з первісних 70-100 до 100-120. Збільшилася також і кількість використовуваних форматів команд.

Однак основний принцип RISC-архітектури залишається незмінним: обробка даних виконується тільки над змістом внутрішніх реєстрів МП без звертання до оперативної пам'яті.

Замість вимоги виконання команди за один такт використовується вимога одержання чергового результату в черговому такті роботи, тобто фактично закріплений *принцип конвеєрної обробки* даних.

Для обробки даних мікропроцесори одержали не один, а кілька конвеєрів зі своїми виконавчими пристроями.

Найбільш відомими RISC-мікропроцесорами в цей час є МП сімейства SPARC фірми Sun Microsystems, Alpha 21x64 фірми Digital Equipment і Rx000 фірми MIPS Computer Systems. За останні роки активно впроваджуються в різну апаратуру RISC-мікропроцесори сімейства PowerPC. Серед фірм, що випускають RISC-мікропроцесори, перебувають також Intel, Hewlett Packard.

Спільний проект компаній Apple, Motorola і IBM - мікропроцесор PowerPC (Performance Optimization With Enhanced RISC) – був орієнтований на створення недорогого, але потужного RISC-процесора й платформи для нього.

До появи архітектури Intel Netburst процесори PowerPC майже завжди перевершували чипи Intel у швидкості обчислень із плаваючою крапкою на десятки відсотків, при цьому споживаючи набагато меншу потужність. З різних причин на заключній стадії цього проекту серед розроблювачів залишилася лише компанія IBM.

Розглянемо організацію роботи RISC-мікропроцесора на прикладі МП Power4 фірми IBM, який є логічним розвитком архітектури PowerPC. Мікропроцесори Power4 містять понад 180 млн транзисторів на кристалі й випускаються із частотами до 1,7 ГГц при досить низькому для високопродуктивних процесорів рівні енергоспоживання (70 Вт). З погляду продуктивності вони залишаються основними й практично єдиними конкурентами 64-розрядних мік-

ропроцесорів архітектури IA-64. Структуру мікропроцесора Power4 наведено на рис. 15.1.

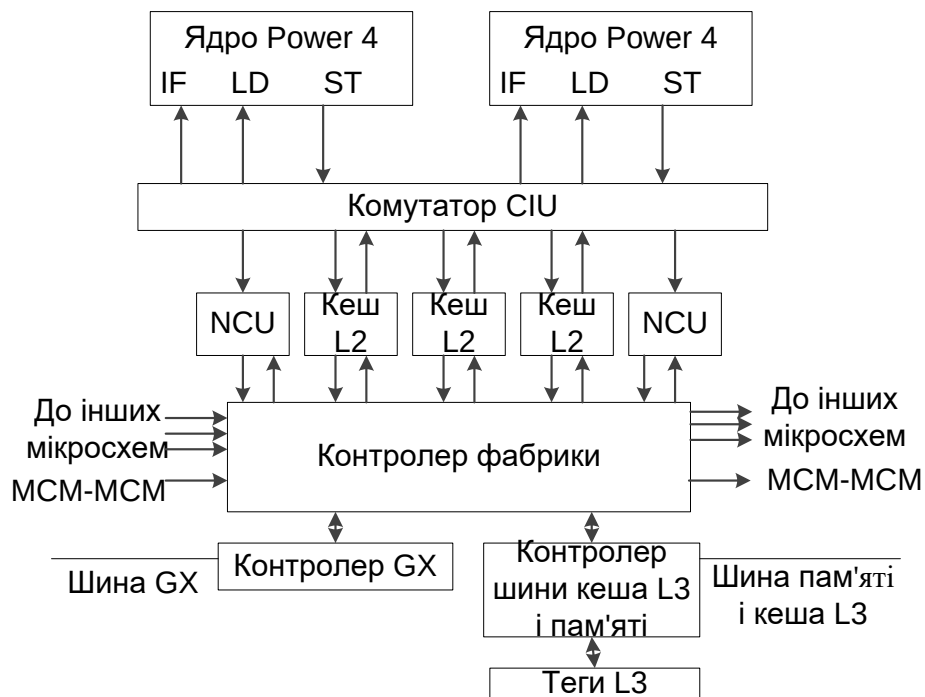


Рис. 15.1. Структура мікропроцесора Power4

Залишаючись по своїй суті RISC-мікропроцесором, Power4 у той же час має багато особливостей. Відзначимо дві головні з них.

По-перше, Power4 став першим мікропроцесором, у якому була реалізована ідея розміщення декількох процесорних ядер на одній мікросхемі. До складу МП Power4 входять два однотипні процесорні ядра.

По-друге, на кристалі Power4 містяться інтегровані апаратні засоби, що визначають важливі архітектурні риси мікропроцесорних систем на його основі. У числі цих засобів порти каналів, що прямо зв'язують Power4 з іншими процесорами усередині так званого мікросхемного модуля (Multi-Chip Module – MCM), канал введення-виведення (шина GX), інтегрований контролер шини оперативної пам'яті й кешу третього рівня, поділюваного всіма процесорами MCM.

Кожний двоядерний процесор Power4 упакований у керамічний мультипроцесорний модуль разом із трьома іншими МП. Такий мікросхемний модуль MCM, що містить чотири МП Power4, у підсумку поєднує вісім процесорів.

Структуру ядра МП Power4 наведено на рис. 15.2.

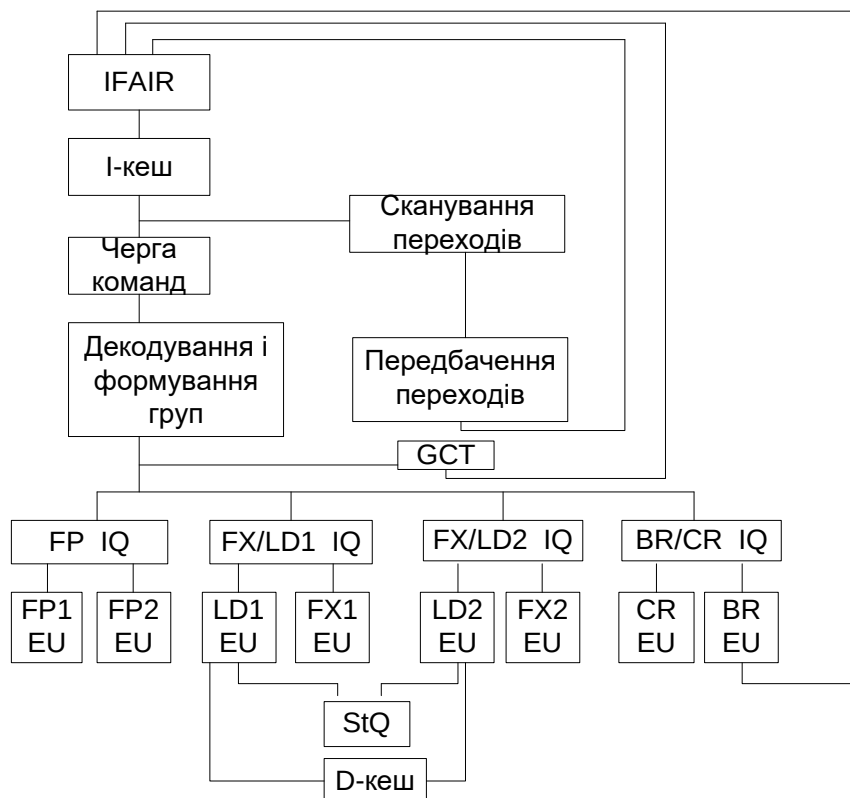


Рис. 15.2. Структура ядра мікропроцесора Power4

У складі кожного ядра Power4 є 8 конвеєризованих виконавчих пристроїв (executiveunit - EU):

- два пристрої обробки чисел із плаваючою крапкою (FP EU), кожне з яких може виконувати команди "помножити-і-скласти", що дає 4 результати із плаваючою крапкою за такт. Операції добування квадратного кореня й ділення із плаваючою комою не конвеєризовані;
- два пристрої для роботи із числами з фіксованою комою (FX EU), розподіл може виконуватися тільки в одному із пристроїв FX EU;
- два пристрої завантаження регістрів / збереження (LD EU), які виконують арифметичні дії по генерації адрес пам'яті;

- пристрій обробки команд переходів (BR EU);
- пристрій виконання логічних операцій з регістром умов (CR EU).

Кожне з ядер має свій кеш першого рівня, окремий для команд (I-Кеш) і даних (D-Кеш). I-Кеш ємністю 64 Кбайт є таким, що адресується прямо, і може читати або записувати 32 байта за такт, що еквівалентно 8 командам. Він використовує IFO-алгоритм заміщення даних. D-Кеш першого рівня ємністю 32 Кбайт має двоканальну організацію, що сприяє збільшенню ефективності використання його ємності в порівнянні із організацією, що адресується прямо, I-Кеша.

Кеш другого рівня в ядрах загальний. Він є вже 8-канальним і має відносно невелику ємність 1,4 Мбайт. Це, однак, компенсується використанням з Power4 кешу третього рівня великої ємності (32 Мбайт на двоядерну мікросхему Power4 або 128 Мбайт на мікросхемний модуль MCM із чотирьох МП).

Кеш другого рівня організовано у вигляді трьох незалежних блоків, кожний зі своїм контролером і інтерфейсом доступу з боку ядра (core Interface unit - CIU). Розподіл обробки запитів у кеш другого рівня між трьома блоками сприяє збільшенню продуктивності. Кожне ядро має по три порти до CIU (для вибірки команд - IF, завантаження регістрів - LD і збереження даних - ST). Контролери кешу другого рівня працюють одночасно, обробляючи по 32 байти за такт процесора.

Для підтримки погодженого представлення даних у кеші другого рівня використовується розширений протокол MESI, що включає 7 станів. Кеш-пам'ять другого рівня розділяється не тільки двома процесорами кристала, але також і зовнішніми процесорами інших кристалів мікросхемного модуля через лінії зв'язку шириною 16 байт, що працюють на частоті більш 500 МГц. Логічною частиною кешу другого рівня є два так звані некешуючих пристрої (non-caching unit - NCU), кожний приписаний до свого процесора. NCU-пристрої відповідають за серіалізацію команд і виконання некешуючих операцій в ієрархії пам'яті.

Через спеціальний пристрій - "контролер фабрики" (fabric controller – FC) дані можуть бути спрямовані в кеш третього рівня й оперативну пам'ять. Усе введення-виведення здійснюється через контролер шини GX, що має порт до FC.

Порт кристала Power4, призначений для підключення кеш-пам'яті 3-го рівня, має ширину 16 байт для кожного із двох напрямків пересилання даних. Порт функціонує на 1/3 від тактової частоти процесорів кристала, що забезпечує пропускну здатність до пам'яті на рівні 10 Гбайт/с.

Слід зазначити, що хоча кеш третього рівня є зовнішнім стосовно МП, його теги розташовані усередині кристала, що скорочує час пошуку потрібної інформації. Кеш третього рівня є загальним для модуля MCM, який може містити крім мікросхеми кешу третього рівня чотири мікросхеми Power4. Для передачі даних між мікросхемами Power4, що входять у MCM, в FC передбачені виділені порти, орієнтовані на створення мультимікропроцесорної системи.

Для звертання до оперативної пам'яті використовується буфер асоціативної трансляції сторінкової адреси TLB. В TLB підтримуються сторінки пам'яті ємністю не тільки 4 Кбайт, але й 16 Мбайт.

Крім цього є асоціативний буфер таблиці сегментів SLB (Segment Lookaside Buffer) ємністю 64 рядка.

Мікропроцесор Power4 несе в собі всі риси RISC-архітектури.

Він має 32 реєстри загального призначення й 32 реєстри для роботи із числами із плаваючою крапкою. Усі обчислення проводяться в реєстрах, а не в основній пам'яті. Система команд – трьохадресна. Деяке представлення про неї дають наступні команди.

ld RA, 0(RB); завантажити в RA число з комірки пам'яті, адреса якої задана в RB

add RA, RB, RC; $RA = RB + RC$

Кожна команда має довжину 32 біти. Перші 6 біт визначають код операції, а інші мають різне значення, що залежить від команди. Той факт, що команди мають фіксовану довжину, дозволяє процесору виконувати їх більш ефе-

ктивно. Відзначимо, що всі обчислювальні команди використовують перший аргумент як місце приймача результату. Регістри у всіх цих командах позначаються тільки їх номерами.

Наприклад, команда, що завантажує число 12 у регістр 5, має вигляд li 5, 12. Знаючи формат команди, можна визначити, що 5 – це номер регістру, а 12 – це число. Ніяких інших показчиків на це немає.

Система команд Power4 має максимальне для RISC-мікропроцесорів число команд, виконуваних за один такт, включаючи одну команду переходу.

Збереження в Power4 системи команд PowerPC, пов'язане з підтримкою двійкового коду користувачів, вимагало застосування не тільки однотактових команд, але й мікропрограм і навіть переривань для програмної реалізації найбільш складних інструкцій.

На стадії підготовки до виконання RISC-команди архітектури PowerPC декодуються в так звані внутрішні мікрооперації (IOP). Таким чином, IOP є аналог мікрооперацій в Pentium 4.

Однак на відміну від Pentium 4, IOP-команди, як правило, збігаються зі звичайними командами PowerPC.

IOP містять тільки команди, які читають не більше двох регістрів і записують не більше ніж в один регістр. Виключення зроблене для деяких команд із плаваючою крапкою типу "помножити-і-скласти", у яких застосовується три регістри-джерела. Інші команди при необхідності розбиваються на послідовність IOP-команд. Так, команда завантаження декількох слів у регістри перетворюється в декілька IOP-команд завантаження одного слова.

На початкових стадіях конвеєра перевіряються можливі взаємозалежності команд (наприклад, за даними), що перешкоджають їхньому одночасному виконанню, і планується об'єднання команд у групи. Так що реально в мікропроцесорі виконуються внутрішні команди, які поєднуються в групи по 5 команд. За один такт 5 команд із однієї групи пересилаються в чергу виконання. Вибірка IOP-команд із черг на виконання у виконавчі пристрої EU може здійснюватися в позачерговому порядку.

За один такт може завершитися тільки одна група команд. Мається на увазі завершення виконання з фіксацією результатів не тільки всіх команд даної групи, але й усіх попередніх груп. Для керування групами використовується таблиця завершення груп GCT ємністю 20 рядків, у кожній з яких міститься адреса першої команди групи й інформація про завершення команд групи. Одночасно в Power4 може перебувати на різних стадіях виконання до 215 команд, а в Pentium 4 - до 126 мікрокоманд.

Процесори Power4 є суперскалярними, що мають конвеєр довжиною 15 стадій і здійснюючими позачергове спекулятивне виконання команд із використанням техніки перейменування регістрів. Для сучасних суперскалярних мікропроцесорів, що мають довгі конвеєри, висока точність передбачення переходів вкрай важлива, а методи передбачення переходів швидко розвиваються. За один такт процесор може вибрати з I-Кешу й запустити на виконання до 8 команд, але "підтримуваний темп", тобто число команд, що завершують виконання за один такт, дорівнює 5. Логіка передбачення переходів сканує їх, і за один такт може обробити до 2 команд переходу. Подібно іншим сучасним RISC-процесорам, в Power4 задіяно кілька механізмів передбачення переходів, включаючи статичне й динамічне передбачення. При цьому напрямок безумовних переходів не передбачується, а напрямок умовних переходів передбачується завжди. Для переходів, які використовуються при викликах підпрограм, застосовується спеціальний стек адрес, а для команд переходів у циклі використовується особливий кеш лічильника. Запис у цей кеш, як і в стек адрес переходу при викликах підпрограм, відбувається, якщо на це є вказівка в коді програми.

Для динамічного передбачення в Power4 використовуються три таблиці історії переходів. Перша – традиційна таблиця історії переходів ємністю 16К рядків, індексом у якій є адреса команди переходу. Друга таблиця пов'язана з унікальною особливістю диспетчерування команд в Power4 групами по 5 команд. Ця таблиця також містить 16К рядків і відслідковує шлях виконання програми, що призводить до переходу. Цей шлях виконання представлений 11-розрядним вектором глобальної історії, по 1 біту на групу команд для всіх по-

передніх обраних 11 груп. Нарешті, третя таблиця - таблиця селектора - має ті ж самі 16К рядків, у яких відслідковується, яка із двох схем передбачення працює краще. На її основі генерується остаточне передбачення переходів.

Динамічне передбачення переходів може бути скасоване статичним, що генерується програмно, для чого використовуються два раніше зарезервовані біти в командах умовного переходу. Якщо ці біти скинуті, використовується апаратне динамічне передбачення.

Кінцевим результатом роботи логіки передбачення переходів є завантаження в регістр IFAR адреси обраної команди.

Підсумки. Розглянуті принципи побудови й розвитку архітектури RISC-процесорів. Розглянута архітектура мікропроцесора Power4, що поєднує в собі RISC-архітектуру й багатоядерність.

15.2. Багатопроцесорні й багатомашинні обчислювальні системи

У цей час тенденція в розвитку мікропроцесорів і систем, побудованих на їхній основі, спрямована на все більше підвищення їх продуктивності. Обчислювальні можливості будь-якої системи досягають своєї найвищої продуктивності завдяки двом факторам: використанню високошвидкісних елементів і паралельному виконанню великої кількості операцій. Напрямки, пов'язані з підвищенням продуктивності окремих мікропроцесорів, ми розглядали в попередніх розділах, а в цьому розділі зупинимося на питаннях розпаралелювання обробки інформації.

Існує кілька варіантів класифікації систем паралельної обробки даних. Очевидно, самою ранньою й найбільш відомою є класифікація архітектур обчислювальних систем, запропонована в 1966 році М. Флінном. Класифікація базується на понятті потоку, під яким розуміється послідовність елементів, команд або даних, оброблювана процесором. На основі числа потоків команд і потоків даних виділяються чотири класи архітектур: SISD, MISD, SIMD, MIMD.

Як і будь-яка інша, наведена вище класифікація недосконала: існують машини, що не попадають прямо в неї, є також важливі ознаки, які в цій класифікації не враховані. Розглянемо класифікацію багатопроцесорних і багатомашинних систем на основі іншої ознаки – ступеню поділу обчислювальних ресурсів системи.

У цьому випадку виділяють наступні 4 класи систем:

- системи із симетричною мультипроцесорною обробкою (symmetric multiprocessing), або SMP-системи;
- системи, побудовані за технологією неоднорідного доступу до пам'яті (non-uniform memory access), або NUMA-системи;
- кластери;
- системи обчислень із масовим паралелізмом (massively parallel processor), або MPP-системи.

Найвищий рівень інтеграції ресурсів має система із симетричною мультипроцесорною обробкою, або *SMP-система* (рис. 15.3).

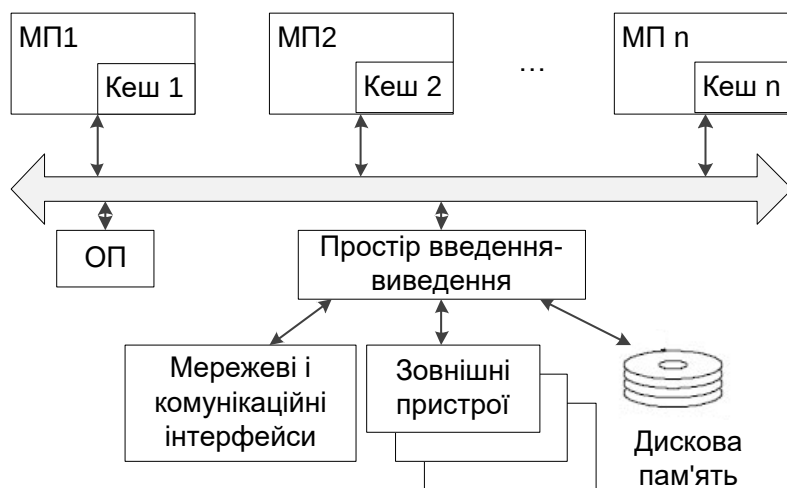


Рис. 15.3. Система із симетричною мультипроцесорною обробкою

У цій архітектурі всі процесори мають рівноправний доступ до всього простору оперативної пам'яті й введення/виведення. Тому SMP-архітектура називається симетричною.

Її інтерфейси доступу до простору введення/виведення й ОП, система керування кеш-пам'яттю, системне ПЗ й т.п. побудовані таким чином, щоб забезпечити погоджений доступ до поділюваних ресурсів. Відповідні механізми блокування закладені й у шинному інтерфейсі, і в компонентах операційної системи, і при побудові кешу.

З погляду прикладного завдання, *SMP-система* являє собою єдиний обчислювальний комплекс із обчислювальними ресурсами, пропорційними кількості процесорів. Розпаралелювання обчислень забезпечується операційною системою, установленою на одному із процесорів. Уся система працює під керуванням єдиної ОС (звичайно UNIX-подібної, але для Intel-платформ підтримується Windows NT).

ОС автоматично в процесі роботи розподіляє процеси по процесорних ядрах, оптимізуючи використання ресурсів. Ядра задіюються рівномірно, і прикладні програми можуть виконуватися паралельно на всій безлічі ядер. При цьому досягається максимальна швидкодія системи. Важливо, що для синхронізації додатків замість складних механізмів і протоколів міжпроцесорної комунікації застосовуються стандартні функції ОС. Таким чином, простіше реалізувати проекти з розпаралелюванням програмних потоків. Загальна для сукупності ядер ОС дозволяє за допомогою службових інструментів збирати статистику, єдину для всієї архітектури. Відповідно, можна полегшити налагодження й оптимізацію додатків на етапі розробки або масштабування для інших форм багатопроцесорної обробки.

У загальному випадку додаток, написаний для однопроцесорної системи, не вимагає модифікації при його переносі в мультипроцесорне середовище. Однак для оптимальної роботи програми або частин ОС вони переписуються спеціально для роботи в мультипроцесорному середовищі.

Порівняно невелика кількість процесорів у таких машинах дозволяє мати одну централізовану загальну пам'ять і об'єднати процесори й пам'ять за допомогою однієї шини.

Стримуючим фактором у подібних системах є пропускна здатність магістралі, що приводить до їхньої поганої масштабованості. Причиною цього є те, що в кожний момент часу шина здатна обробляти тільки одну транзакцію, внаслідок чого виникають проблеми розв'язання конфліктів при одночасному звертанні декількох процесорів до тих самих областей загальної фізичної пам'яті. Обчислювальні елементи починають заважати один одному. Коли відбудеться такий конфлікт, залежить від швидкості зв'язку й від кількості обчислювальних елементів. Крім того, системна шина має обмежене число слотів. Усе це очевидно перешкоджає збільшенню продуктивності при збільшенні числа процесорів. У реальних системах можна задіяти не більше 32 процесорів.

У сучасних мікропроцесорах підтримка побудови мультипроцесорної системи закладається на рівні апаратної реалізації МП, що робить багатопроцесорні системи порівняно недорогими.

Так, для забезпечення можливості роботи на загальну магістраль кожний мікропроцесор фірми Intel починаючи з Pentium Pro має вбудовану підтримку дворозрядного ідентифікатора процесора – APIC (Advanced Programmable Interrupt Controller). За замовчуванням CPU з найвищим номером ідентифікатора стає процесором початкового завантаження. Така ідентифікація полегшує арбітраж шини даних в *SMP-системі*. Подібні засоби ми бачили й у МП Power4, де на апаратному рівні підтримується створення мікросхемного модуля MSM з 4 мікропроцесорів, що включає в сукупності 8 процесорних ядер.

Сьогодні SMP широко застосовують у багатопроцесорних суперкомп'ютерах і серверних додатках. Однак якщо необхідно детерміноване виконання програм у реальному масштабі часу, наприклад, при візуалізації мультимедійних даних, можливості суто симетричної обробки досить обмежені. Може виникнути ситуація, коли додатки, виконувані на різних ядрах, звертаються до одного ресурсу ОС. У цьому випадку доступ одержить тільки одне з ядер. Інші будуть простоювати до вивільнення критичної області. Звичайно, при цьому різко знижується продуктивність додатків реального часу.

Вичерпання продуктивності системної шини в *SMP-системах* при доступі великої кількості процесорів до загального простору оперативної пам'яті й принципові обмеження шинної технології стали причиною стримування росту продуктивності *SMP-систем*. На даний момент ця проблема одержала два розв'язки. Перший - заміна системної шини на високопродуктивний комутатор, що забезпечує одночасний доступ, що не блокує, до різних ділянок пам'яті. Другий розв'язок пропонує *технологія NUMA*.

Система, побудована за *технологією NUMA*, являє собою набір вузлів, кожний з яких, по суті, є функціонально закінченим однопроцесорним або SMP-комп'ютером. Кожний має свій локальний простір оперативної пам'яті й введення/виведення. Але за допомогою спеціальної логіки кожний має доступ до простору оперативної пам'яті й введення/виведення будь-якого іншого вузла (рис. 15.4). Фізично окремі пристрої пам'яті можуть адресуватися як логічно єдиний адресний простір - це означає, що будь-який процесор може виконувати звертання до будь-яких комірок пам'яті, у припущенні, що він має відповідні права доступу. Тому іноді такі системи називаються системами з розподіленою поділюваною пам'яттю (DSM - distributed shared memory).

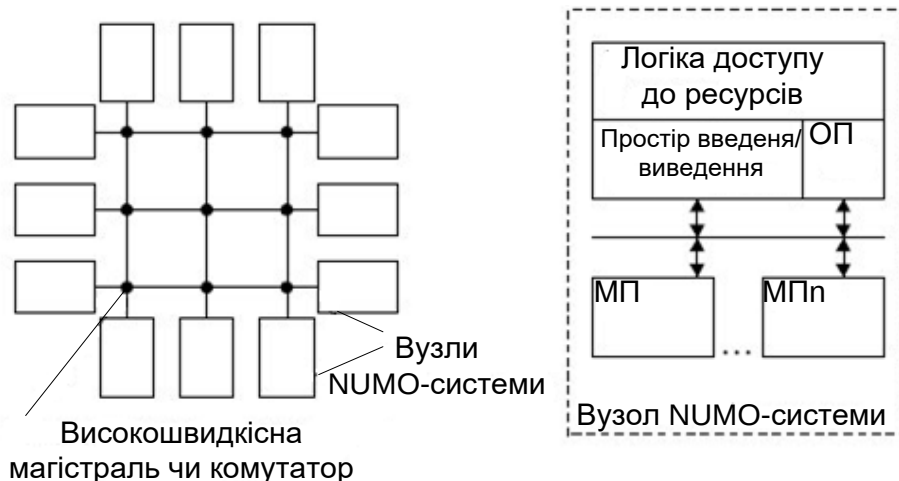


Рис. 15.4. Система, побудована за технологією неоднорідного доступу до пам'яті

При такій організації пам'ять кожного вузла системи має свою адресацію в адресному просторі всієї системи. Логіка доступу до ресурсів визначає, до

пам'яті якого вузла ставиться вироблена процесором адреса. Якщо вона не належить пам'яті даного вузла, організується звертання до іншого вузла відповідно закладеного у логіку доступу карті адрес. При цьому доступ до локальної пам'яті здійснюється в кілька разів швидше, ніж до віддаленої.

При використанні найпоширенішого зараз варіанта cc-NUMA (cache-coherent NUMA - неоднорідний доступ до пам'яті з узгодженням змісту кеш-пам'яті) забезпечується кешуванням даних оперативної пам'яті інших вузлів.

Звичайно вся система працює під керуванням єдиної ОС, як в SMP. Можливі також варіанти динамічного поділу системи, коли окремі розділи системи працюють під керуванням різних ОС.

Досить великий час доступу до оперативної пам'яті сусідніх вузлів у порівнянні з доступом до ОП свого вузла в *NUMA-системах* на даний момент робить таке використання не цілком оптимальним.

Так що повною функціональністю *SMP-систем NUMA-комп'ютери* на сьогоднішній день не володіють. Однак серед систем загального призначення NUMA-системи мають один з найбільш високих показників по масштабованості й, відповідно, по продуктивності. На сьогодні максимальне число процесорів в cc-NUMA-системах може перевищувати 1000 (серія Origin3000). Один з найбільш продуктивних суперкомп'ютерів - Tera 10 - має продуктивність 60 Тфлопс і складається з 544 SMP-вузлів, у кожному з яких перебуває від 8 до 16 процесорів Itanium 2.

Наступним рівнем в ієрархії паралельних систем є комплекси, що також полягають із окремих машин, але лише частково поділяючи деякі ресурси. Мова йде про *кластери*.

Кластер являє собою систему з декількох комп'ютерів (у більшості випадків, що серійно випускаються), що мають загальний поділюваний ресурс для зберігання спільно оброблюваних даних (звичайно набір дисків або дискових масивів) і об'єднаних високошвидкісною магістраллю (рис. 15.5).

У кластерній системі деякий розподілений додаток паралельний на декількох вузлах обробляє загальний набір даних, як правило, таким чином, щоб у користувача виникла ілюзія роботи на одній машині.

Звичайно в кластерних системах не забезпечується єдине операційне середовище для роботи загального набору додатків на всіх вузлах кластера. Тобто кожний комп'ютер кластера - це автономна система з окремим екземпляром ОС і власними, що належать тільки їй системними ресурсами: набором заведених користувачів, системними буферами, областю свопінгу й т.п. Додаток, запущений на ньому, може бачити тільки загальні диски або окремі ділянки пам'яті. На вузлах кластера працюють спеціально написані для такої конфігурації додатки, що паралельно обробляють загальний набір даних. На кожній з машин вони представлені рядом процесів, програм, взаємодіючих за допомогою кластерного програмного забезпечення. Таким чином, кластерне ПЗ - це лише засіб для взаємодії вузлів і синхронізації доступу до загальних даних. Кластер як паралельна система формується на прикладному рівні, а не на рівні операційної системи.

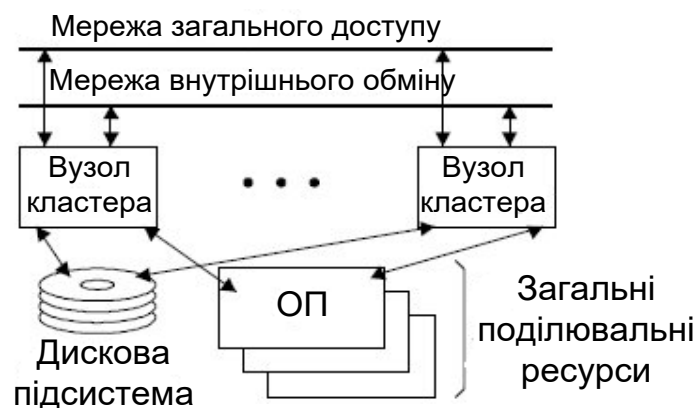


Рис. 15.5. Структура кластера

У цей час такі системи мають дві основні області застосування: паралельні сервери баз даних і високонадійні обчислювальні комплекси. Ринок паралельних СУБД і є фактично ринок кластерів додатків. Високонадійні комплекси

являють собою групу вузлів, на яких незалежно один від одного виконуються деякі важливі додатки, що вимагають постійної, безперервної роботи.

Тобто в такій системі на апаратному рівні фактично підтримується основний механізм підвищення надійності – резервування. Причому вузли перебувають у так званому "гарячому" резерві, і кожен з них у будь-який момент готовий продовжити обчислення при виході з ладу якого-небудь вузла. При цьому всі додатки з вузла, що відмовив, автоматично переносяться на інші машини комплексу.

Така система також формально є кластером, хоча в ній відсутня паралельна обробка загальних даних. Ці дані звичайно монополюють використовуються виконуваними в рамках кластера додатками й повинні бути доступні для всіх вузлів.

Якщо в кластері його вузли розділяють деякі ресурси, то паралельні системи іншого класу – *системи обчислень із масовим паралелізмом* (MPP) – будуються з окремих повністю незалежних комп'ютерів, з'єднаних тільки високошвидкісною магістраллю або комунікаційними каналами (рис. 15.6).

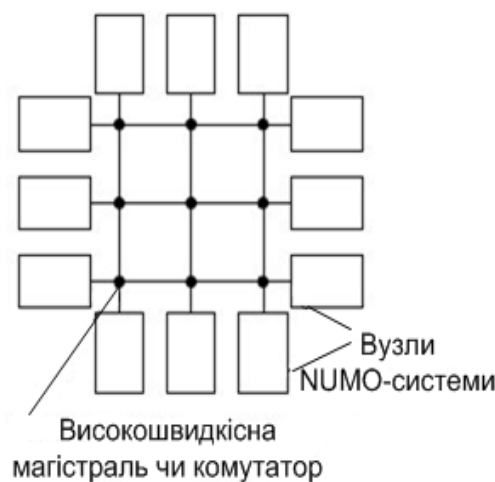


Рис. 15.6. Структура MPP-системи

Це можуть бути або декілька UNIX-машин, що серійно випускаються, з'єднаних за допомогою високопродуктивного мережного середовища, або спеціально сконструйована система з окремих функціональних блоків, об'єднаних

комутатором. У такій системі адресний простір складається з окремих адресних просторів, які логічно не зв'язані між собою й доступ до яких не може бути здійснений апаратно іншим процесором.

При цьому для обміну даними використовується механізм передачі повідомлень між процесорами. Тому ці машини часто називають машинами з передачею повідомлень. Користувач може визначити логічний номер процесора, до якого він підключений, і організувати обмін повідомленнями з іншими процесорами.

На машинах MPP-архітектури використовуються два варіанти роботи операційної системи. В одному з них повноцінна операційна система працює тільки на керуючій машині (front-end); на кожному окремому модулі функціонує сильно урізаний варіант ОС, що забезпечує роботу тільки розташованої в ньому частини паралельного додатка. У другому варіанті на кожному модулі працює повноцінна, як правило, UNIX-подібна ОС, встановлена окремо.

Програмування в такій системі - досить складне завдання.

Воно вимагає спеціального інструментарію й особливого системного програмного забезпечення для роботи паралельних додатків, які орієнтовані на функціонування паралельних процесів, розподілених по вузлах MPP-системи, з обміном повідомленнями між ними.

Підвищення продуктивності машин з масовим паралелізмом шляхом збільшення в них числа процесорів має певні обмеження. Чим більше число процесорів входить до складу MPP-системи, тим довше канали передачі керування й даних, а виходить, і тим менше тактова частота. Зростання норми масивності для більших машин до 512 і навіть 64К процесорів, що відбулося, обумовлене не ростом розмірів машини, а збільшенням ступеня інтеграції схем, що дозволила за останні роки різко підвищити щільність розміщення елементів у пристроях. Топологія мережі міжпроцесорного обміну в такого роду системах може бути різною.

Головною перевагою MPP-систем є їхня гарна масштабованість: на відміну від SMP-систем, тут кожний процесор має доступ тільки до своєї локальної

пам'яті, у зв'язку із чим не виникає необхідності в потактовій синхронізації процесорів. Практично всі рекорди з продуктивності на сьогодні встановлюються на машинах саме такої архітектури процесорів, що складаються із декількох тисяч.

Основні недоліки систем даного типу наступні:

- відсутність загальної пам'яті помітно знижує швидкість міжпроцесорного обміну, оскільки немає загального середовища для зберігання даних, призначених для обміну між процесорами;
- потрібна спеціальна техніка програмування для реалізації обміну повідомленнями між процесорами;
- кожний процесор може використовувати тільки обмежений обсяг локального банку пам'яті;
- внаслідок зазначених архітектурних недоліків потрібні значні зусилля для того, щоб максимально задіяти системні ресурси, наслідком чого є висока ціна програмного забезпечення для MPP-систем з роздільною пам'яттю.

Підведемо деякі підсумки, що стосуються областей застосування систем паралельної обробки даних різних типів.

SMP-системи потенційно мають достатні можливості для забезпечення необхідної для більшості застосувань продуктивності: цілком природно збільшувати число процесорів, а не ставити поруч ще один комп'ютер. Додавання одного процесора гарантовано збільшує продуктивність, а додавання, наприклад, вузла в кластер адекватного прискорення не дасть. Більше того, у деяких випадках загальна продуктивність системи може навіть впасти, коли вузли кластера починають активно конкурувати за доступ до загальних ресурсів, і взаємні блокування зводять нанівець переваги паралельної обробки.

NUMA-системи створюються для цілком певних цілей – забезпечення масштабних розрахунків. Системи, що використовують цю архітектуру, насамперед застосовуються для унікальних високоякісних і високопродуктивних прикладних програм, що вимагають більш восьми процесорів. Однак вони мають високу вартість і вимагають унікального ПЗ (прикладні програми й ОС).

Для сучасних систем крім цілком традиційних вимог по продуктивності, масштабованості, важливі додаткові високі вимоги пред'являються до надійності їх роботи. Саме із цих міркувань обчислювальні комплекси на основі кластерів або MPP-машин завоюють усе більшу популярність.

MPP-системи мають ряд переваг, головною з яких є краща серед усіх розглянутих архітектур масштабованість. Саме тому MPP-комп'ютери звичайно використовуються при більших ресурсомістких обчисленнях. Звичайно, вони застосовуються й при побудові більших баз даних, і в відмовостійких обчислювальних комплексах. Але тут їх використання достатньо обмежене. Це частково пов'язано з тим, що вони все-таки дорожче кластерів і мають досить більшу початкову ціну. Кластер же можна побудувати з відносно дешевих машин довільної конфігурації.

Наведена класифікація систем паралельної обробки даних досить умовна. Розроблювачі обчислювальних систем не проектують машину якогось спеціального класу, а намагаються створити більш продуктивну архітектуру. Крім цього, сам користувач може з використанням стандартних компонентів спроектувати комплекс, архітектурно й функціонально найбільш підходящий для розв'язку конкретного завдання.

Трансп'ютери

Для побудови багатопроцесорних систем можуть бути використані спеціально розроблені процесори, які називають Трансп'ютерами. Вони були створені в середині 1980-х років фірмою INMOS Ltd (нині – підрозділ Stmicroelectronics).

Трансп'ютер - це мікропроцесор із вбудованими засобами міжпроцесорної комунікації, призначеної для побудови багатопроцесорних систем. Його назва походить від слів Transfer (передавач) і computer (обчислювач).

Трансп'ютер містить у собі засоби для виконання обчислень (ЦП, АЛП із плаваючою крапкою, внутрішньокристалъну пам'ять) і 4 канали для зв'язку (лінка) з іншими трансп'ютерами й/або іншими пристроями. Кожний лінк являє собою 2 односпрямованих послідовних канали передачі інформації. Вбудова-

ний інтерфейс дозволяє підключати зовнішню пам'ять ємністю до 4 Гбайт (рис. 15.7).

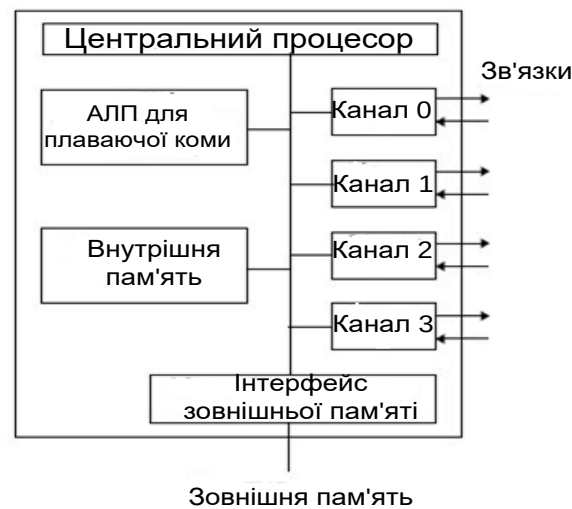


Рис. 15.7. Структура трансп'ютера

Багатопроцесорна система може створюватися з набору трансп'ютерів, які функціонують незалежно й взаємодіють через послідовні канали зв'язку (рис. 15.8).

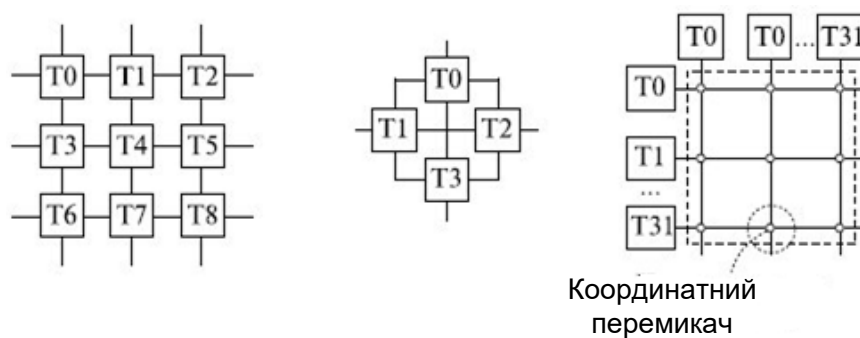


Рис. 15.8. Структури багатопроцесорних систем на базі трансп'ютерів

Організація трансп'ютерів і багатопроцесорних систем на їхній основі базується мовою Оссат, спеціально розробленою на початку 1980-х років групою вчених з Оксфорда під керівництвом Девіда Мея за завданням компанії INMOS у рамках робіт зі створення трансп'ютерів.

Відзначимо основні характеристики цієї мови з погляду його застосування для програмування трансп'ютерних систем:

- виконання завдання розбивається на процеси, які можуть виконуватися паралельно;
- розміщення процесів не прив'язане до конкретного пристрою: необов'язково кожному процесу повинен ставитися у відповідність свій трансп'ютер - кілька процесів можуть виконуватися на одному трансп'ютері;
- мова дозволяє описати розміщення процесів по пристрою;
- мова дозволяє описати, як ці процеси обмінюються між собою даними (який процес приймає інформацію, від якого процесу й у якому обсязі).

При передачі даних у лінк процес повинен виконати команду виводу. Процес, що виконав таку команду, затримується доти, поки всі дані не будуть передані. Аналогічно, при прийманні даних з лінка процес повинен виконати команду введення. При виконанні такої команди процес блокується доти, поки буфер не буде заповнений даними. Взаємодія із зовнішнім пристроєм через лінк дозволяє трансп'ютеру синхронізувати свою діяльність із цими пристроями без використання механізму переривань.

Використання такого підходу дозволило організувати віртуальні канали зв'язку між процесами, які могли розміщатися як на єдиному трансп'ютері, так і на декількох трансп'ютерах, і віртуальні лінки між процесами. Будь-який трансп'ютер може одночасно утворювати будь-яке число паралельних процесів. Він має спеціальний планувальник, який робить розподіл процесорного часу між цими процесами. Тим самим надається можливість, маючи всього лише один трансп'ютер, написати паралельну програму, яка повністю виконується на ньому. Завдання розбивається на ряд процесів, і всі ці процеси паралельно протікають усередині одного трансп'ютера, періодично зупиняючись для одержання даних одне від одного.

Систему можна розширити іншими трансп'ютерами й перенести на них ряд процесів. При цьому потрібно просто перевизначити таблицю зв'язків процесів, указавши, на якому трансп'ютері тепер виконується той або інший про-

цес. Сама ж програма змін не перетерплює, а обчислювальна потужність системи, звісно, збільшується.

Крім цікавих можливостей, пов'язаних з побудовою мультипроцесорних систем без залучення додаткового встаткування, у трансп'ютерах були реалізовані ідеї, спрямовані на підвищення їх обчислювальної потужності. Серед них хотілося б відзначити те, що блок реєстрів трансп'ютера організований у вигляді стека. Це привело до використання переважно безадресної системи команд, що забезпечило навіть більш високу продуктивність, ніж RISC-архітектура. Другий момент, який слід виділити, - це одночасне виконання групи, у яку входило до 8 команд, що забезпечувало повне завантаження пристроїв процесора. І все це було реалізовано наприкінці 1980-х років, задовго до появи EPIC і Itanium. На момент своєї появи трансп'ютери були самими швидкодіючими 32-розрядними мікропроцесорами. У процесі свого недовгого розвитку їх характеристики досягли наступних значень: продуктивність: 200 MIPS, 25 MFLOPS (на 64-розрядному процесорі із плаваючою крапкою); ємність внутрішньокристалльної пам'яті: 16 Кбайт; швидкість обміну по лінку: 100 Мбит/с.

У середині 1990-х, в епоху розквіту мікропроцесорів цього сімейства, фірма INMOS крім власнотрансп'ютерів поставляла широкий набір тремів (Trem - Transputer Extension Module) - пристроїв введення-виведення з лінком у якості інтерфейсу. Зокрема, поставлялися тремі, що дозволяли підключити до трансп'ютера через лінк адаптери Ethernet або SCSI. З метою збільшення числа фізичних зв'язків був розроблений програмувальний комутатор, що здійснює передачу повідомлення з кожного з 32 входів на кожний з 32 виходів.

Трансп'ютери успішно використовувалися в різних областях від вбудованих систем до супер ЕОМ. Однак технологія трансп'ютерів серйозного розвитку не одержала, тому що починаючи з Pentium Pro в універсальні мікропроцесори введена можливість з'єднання процесорів у мікропроцесорну систему, що знецінила головну перевагу трансп'ютерів – можливість побудови багатопроцесорних систем без додаткових апаратних витрат. У цей час трансп'ютери не ви-

робляються, вони витиснуті схожими розробками конкурентів, особливо Texas Instruments (TMS320) і Intel (80860).

15.3. Процесори цифрової обробки сигналів

Процесори цифрової обробки сигналів являють собою клас спеціалізованих мікропроцесорів, призначених для розв'язку завдань *цифрової обробки сигналів* (ЦОС), до яких відносяться обробка звукових сигналів, обробка зображень, розпізнавання мови, розпізнавання образів, цифрова фільтрація, спектральний аналіз і ін.

Часто в літературі такі мікропроцесори називаються *цифровими сигнальними процесорами* (ЦСП), або DSP (Digital Signal Processors).

Перші процесори цього класу з'явилися наприкінці 1970-х років. Вимоги практики, пов'язані із широким розвитком мобільного бездротового зв'язку, стаціонарних систем широкосмугового зв'язку, використанням цифрової обробки сигналів у побутовій аудіо- і відеотехніці й пристроях комп'ютерної периферії, з однієї сторони й колосальний прогрес електронної промисловості з іншої привели до того, що до теперішнього часу продуктивність ЦСП зросла до сотень мільйонів операцій у секунду, а ціна впала більше ніж на 90 % і навіть для самих потужних процесорів становить у цей час менше \$20. Низька споживана потужність (менше 1 Вт на максимальній частоті роботи процесора) забезпечує їхнє широке використання в різних пристроях, що вбудовуються, від побутової електроніки до бортових систем спеціального призначення.

Цифрова обробка сигналів – це арифметична обробка послідовності значень амплітуд сигналу, одержуваних через рівні проміжки часу. Головне, що відрізняє цю інформацію, – вона необов'язково заноситься до пам'яті й тому може виявитися недоступною в майбутньому. Отже, обробляти її потрібно в реальному масштабі часу (РМВ).

До основних особливостей цифрової обробки сигналів, які багато в чому визначають архітектуру процесорів DSP, відносяться:

- потоковий характер обробки більших обсягів даних у РМВ;
- забезпечення можливості інтенсивного обміну із зовнішніми пристроями.

Для ефективної реалізації алгоритмів цифрової обробки сигналів необхідна апаратна підтримка базових операцій ЦОС. Тому розглянемо спочатку принципи цифрової обробки сигналів, що мають особливий вплив на архітектуру ЦСП.

Будь-який аналоговий сигнал можна представити у вигляді характеристик -або амплітуда-час, або частота-амплітуда. Для переходу від однієї форми представлення до іншої використовується *перетворення Фур'є*. Операції, що виконують це перетворення, є базовими в *цифровій обробці сигналів*.

Перетворення Фур'є являє собою в загальному випадку роботу з якоюсь безперервною функцією. З безперервним перетворенням Фур'є зручно працювати в теорії, але на практиці ми звичайно маємо справу з дискретними даними. Для обробки звукових і відеосигналів у комп'ютері вони спочатку перетворюються в цифрову форму й представляються у вигляді деякого набору відліків частот і амплітуд, вироблених через певні проміжки часу (період дискретизації). Тому тут варто говорити не про інтегральний, а про *дискретне перетворення Фур'є* (ДПФ).

Нехай дана кінцева послідовність чисел $x_0, x_1, x_2, \dots, x_{N-1}$ (у загальному випадку комплексних). *Дискретне g_j перетворення Фур'є* полягає в пошуку іншої послідовності $X_0, X_1, X_2, \dots, X_{N-1}$, елементи якої обчислюються за формулою:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-j2\pi kn}{N}}$$

Нехай дана кінцева послідовність чисел $X_0, X_1, X_2, \dots, X_{N-1}$ (у загальному випадку комплексних). *Зворотне дискретне перетворення Фур'є* полягає в пошуку іншої послідовності $x_0, x_1, x_2, \dots, x_{N-1}$, елементи якої обчислюються за формулою:

$$X_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{-j2\pi kn}{N}}$$

У загальному випадку гармонійне коливання, яке має вигляд, що наведено на рис. 15.9, описується наступним виразом:

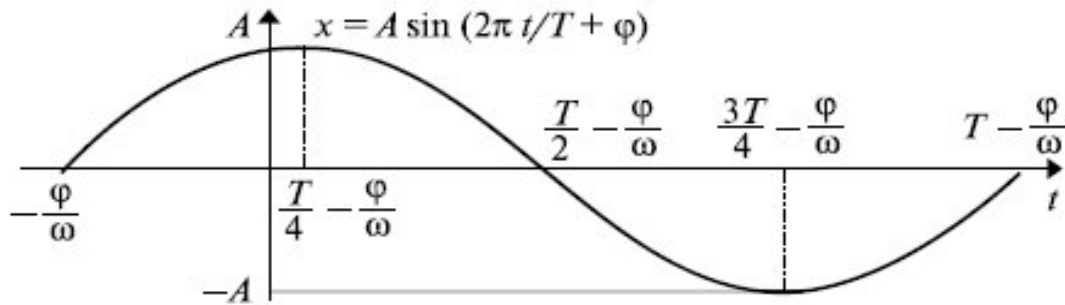


Рис. 15.9. Загальний вид гармонійного коливання

$$x = A \sin\left(\frac{2\pi * t}{T} + \varphi\right),$$

де A_0 – амплітуда сигналу, T – його період, φ – зсув фази сигналу.

Це коливання можна описати також виразом:

$$x = A \cos(\omega t + \varphi)$$

де $\omega = 2\pi / T$ – кругова частота сигналу. Даний вираз будемо називати гармонікою.

Перетворимо останній вираз до вигляду:

$$x = A \cos \varphi \cos(2\pi / T) - A \sin \varphi \sin(2\pi / T)$$

Виділимо елементи, незалежні від t , і позначимо їх як Re і Im . У результаті одержимо:

$$x = Re \cos(2\pi / T) - Im \sin(2\pi / T),$$

де $Re = A \cos \varphi, Im = A \sin \varphi$

По величинах Re і Im можна однозначно відновити амплітуду й фазу вихідної гармоніки:

$$\varphi = \arctg\left(\frac{\text{Im}}{\text{Re}}\right), A = \sqrt{\text{Re}^2 + \text{Im}^2}$$

Розкладемо кожне комплексне число X_k з виразу зворотного перетворення Фур'є на уявну й дійсну складові $X_k = R_{ek} + \varphi \text{Im}_k$, а експоненту – за формулою Ейлера на синус і косинус дійсного аргументу. Одержимо:

$$x_n = \sum_{k=0}^{N-1} \left[\frac{\text{Re}_k}{N} \cos\left(\frac{2\pi kn}{N}\right) - \frac{\text{Im}_k}{N} \sin\left(\frac{2\pi kn}{N}\right) \right] + i \sum_{k=0}^{N-1} \left[\frac{\text{Re}_k}{N} \sin\left(\frac{2\pi kn}{N}\right) - \frac{\text{Im}_k}{N} \cos\left(\frac{2\pi kn}{N}\right) \right]$$

Розглянемо тепер функцію $x = f(t)$, що представляє собою деяке звукове або яке-небудь інше коливання. Нехай це коливання описане графіком на часовому інтервалі $[0, T]$ (рис. 15.10).

Для обробки цього сигналу в комп'ютері потрібно виконати його дискретизацію. Із цією метою часовий інтервал ділиться на $N-1$ частин і зберігаються значення функції $x_0, x_1, x_2, \dots, x_{N-1}$ для N точок на границях інтервалів.

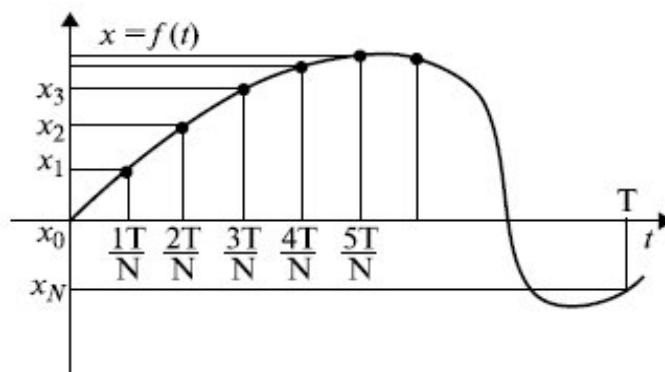


Рис. 15.10. Дискретизація гармонійної функції

У результаті *прямого дискретного перетворення Фур'є* можуть бути отримані N значень для X_k згідно виразу.

Якщо тепер застосувати *зворотне дискретне перетворення Фур'є*, то вийде вихідна послідовність $\{x_n\}$. Вихідна послідовність складалася з дійсних

чисел, а послідовність $\{X_k\}$ у загальному випадку комплексна. Якщо прирівняти нулю її уявну частину, то одержимо:

$$x_n = f(t_n) = \sum_{k=0}^{N-1} \left[\frac{\text{Re}_k}{N} \cos\left(\frac{2\pi kn}{N}\right) - \frac{\text{Im}_k}{N} \sin\left(\frac{2\pi kn}{N}\right) \right] + \sum_{k=0}^{N-1} \left[\frac{\text{Re}_k}{N} \cos\left(\frac{2\pi kn}{N}\right) \right]$$

Зіставивши цю формулу з формулами для гармоніки, побачимо, що вираз являє собою суму з N гармонійних коливань різної частоти, фази й амплітуди. Тобто фізичний зміст *дискретного перетворення Фур'є* полягає в тому, щоб представити деякий дискретний сигнал у вигляді суми гармонік. Параметри кожної гармоніки обчислюються прямим перетворенням Фур'є, а сума гармонік - зворотним.

Тепер, наприклад, операція "фільтр нижніх частот", яка "вирізає" із сигналу всі частоти вище деякої заданої, може просто скинути коефіцієнти, відповідні до частот, які необхідно вилучити. Потім, після обробки, виконується зворотне перетворення.

Особливості *цифрової обробки сигналів* розглянемо на прикладі алгоритму нерекурсивної фільтрації. Структуру пристрою, що реалізує даний алгоритм, наведено на рис. 15.11.

Обробка полягає у формуванні вихідного сигналу $Y[k]$ за значеннями N останніх вхідних відліків $x[k]$, які надходять на вхід пристрою через певний інтервал часу T . Прийняті відліки зберігаються в комірках циклічного буфера. При прийманні чергового відліку вміст усіх комірок буфера переписується в сусідню позицію, самий старий відлік залишає буфер, а новий записується в його молодшу комірку.

Аналітично алгоритм роботи нерекурсивного фільтра записується як:

$$Y[k] = \sum_{i=1}^{N-1} x[k-i] * a_i$$

де a_i – коефіцієнти, обумовлені типом фільтра.

Відліки з виходів елементів буфера надходять на помножувачі, на інші входи яких надходять коефіцієнти a_i . Результати добутків складаються й формують відлік вихідного сигналу $Y[k]$, після чого вміст буфера зсувається на 1 позицію й цикл роботи фільтра повторюється. Вихідний сигнал $Y[k]$ повинен бути обчислений до надходження наступного вхідного сигналу, тобто за інтервал T . У цьому полягає суть роботи пристрою в реальному масштабі часу.

Інтервал часу T задається частотою дискретизації, яка визначається областю застосування фільтра. За наслідком з теореми Котельникова в дискретному сигналі період, що відповідає найвищій уявній частоті, відповідає двом періодам дискретизації.

При обробці звукового сигналу частоту дискретизації можна прийняти в 40 кГц. У цьому випадку якщо необхідно реалізувати цифровий нерекурсивний фільтр 50-го порядку, то за час в $1/40$ кГц = 25 мкс повинно бути виконано 50 множень і 50 накопичень результатів множення. Для обробки відеосигналу інтервал часу, за який повинні бути виконані ці дії, буде на кілька порядків менше.

Якщо виконувати ДПФ вхідної послідовності прямо, суворо по вихідній формулі, то буде потрібно багато часу. Порахувавши за визначенням (N раз підсумувати N доданків), одержуємо величину порядку N^2 .

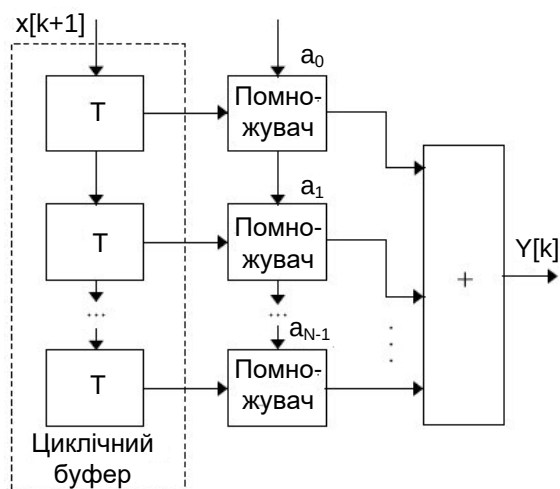


Рис. 15.11. Схема роботи нерекурсивного фільтра

Проте, можна обійтися суттєво меншим числом операцій.

Найбільш популярним з алгоритмів прискореного обчислення ДПФ є метод Лантухи-Т'юки (Cooley-Tukey), що дозволяє обчислити ДПФ для числа відліків $N = 2^k$ за час порядку $N \cdot \log_2 N$ (звідси й назва – *швидке перетворення Фур'є*, ШПФ, або в англійському варіанті FFT – Fast Fourier Transformation). Основна ідея цього методу полягає в рекурсивній розбивці масиву чисел на два підмасиви й відомості обчислення ДПФ від цілого масиву до обчислення ДПФ від підмасивів окремо. При цьому процес розбивки вихідного масиву на підмасиви проводиться по методу побітового зворотного сортування (bit-reversal sorting).

Спочатку вхідний масив ділиться на два підмасиви – з парними й непарними номерами. Кожний з підмасивів перенумеровується й знову ділиться на два підмасиви – з парними й непарними номерами. Це сортування триває доти, поки розмір кожного підмасива не досягне 2 елементів. У результаті (що можна показати математично) номер кожного вихідного елемента у двійковій системі перевертається. Тобто, наприклад, для однобайтових номерів двійковий номер 00000011 стане номером 110000000, номер 01010101 – номером 10101010.

Існують алгоритми ШПФ для випадків, коли N є ступенем довільного простого числа (а не тільки двійки), а також у випадку, коли число N є добутком ступенів простих чисел будь-якого числа відліків. Однак ШПФ, реалізоване по методу Лантухи-Т'юки для випадку $N = 2^k$, одержало найбільш широке поширення. Причина цього в тому, що алгоритм, побудований за цим методом, має ряд дуже гарних технологічних властивостей:

- структура алгоритму і його базові операції не залежать від числа відліків (міняється тільки число прогонів базової операції);
- алгоритм легко розпаралелюється з використанням базової операції й конвеєризується, а також легко каскадується (коефіцієнти ШПФ для $2N$ відліків можуть бути отримані перетворенням коефіцієнтів двох ШПФ по N відліків, отриманих "проріджуванням" вихідних $2N$ відліків через один);
- алгоритм простий і компактний, допускає обробку даних "на місці" і не вимагає додаткової оперативної пам'яті.

Однокристальні мікроконтролери й навіть універсальні мікропроцесори виявляються відносно повільними при виконанні операцій, характерних для ЦОС. До того ж вимоги до якості перетворення аналогових сигналів постійно зростають. У *сигнальних мікропроцесорах* такі операції підтримуються на апаратному рівні й виконуються, відповідно, досить швидко. Робота в реальному масштабі часу вимагає від процесора також підтримки на апаратному рівні таких дій, як обробка переривань, програмних циклів.

Усе це приводить до того, що DSP-процесори, архітектурно включаючи в себе багато рис як універсальних мікропроцесорів, особливо з RISC-архітектурою, так і *однокристальних мікроконтролерів*, у той же час значно відрізняються від них. Універсальний мікропроцесор крім чисто обчислювальних операцій виконує функцію об'єднуючої ланки всієї мікропроцесорної системи, зокрема комп'ютера.

Він повинен управляти роботою різних компонентів апаратного забезпечення, таких як дисководи, графічні дисплеї, мережний інтерфейс, для того щоб забезпечити їхню погоджену роботу. Це приводить до досить складної архітектури, оскільки вона повинна підтримувати поряд із цілочисельною арифметикою й операціями із плаваючою комою такі базові функції, як захист пам'яті, мультипрограмування, обробка векторної графіки й т.п. У підсумку типовий універсальний мікропроцесор з CISC-, а найчастіше й RISC-архітектурою має систему з кілька сотень команд, які забезпечують виконання всіх цих функцій і відповідну апаратну підтримку. Це призводить до необхідності мати в складі такого МП десятки мільйонів транзисторів.

У той же час *DSP-процесор* є вузькоспеціалізованим пристроєм. Його єдине завдання – швидко обробляти потік цифрових сигналів. Він складається головним чином з високошвидкісних апаратних схем, що виконують арифметичні функції, що й маніпулюють бітами, оптимізованими таким чином, щоб швидко обробляти більші об'єми даних. У силу цього набір команд в DSP значно менше, ніж в універсального мікропроцесора: їх число звичайно не перевищує 80. Це значить, що для DSP потрібен полегшений декодер команд і наба-

гато менше число виконавчих пристроїв. Крім того, усі виконавчі пристрої в остаточному підсумку повинні підтримувати високопродуктивні арифметичні операції. Таким чином, типовий *DSP-процесор* складається не більш ніж з кількох сотень тисяч (а не десятків мільйонів, як у сучасних CISC-МП) транзисторів. У силу цього такі МП споживають менше енергії, що дозволяє використовувати їх у продуктах, що працюють від батарей. Вкрай спрощується їхнє виробництво, тому вони знаходять собі застосування в недорогих пристроях. Комбінація низького енергоспоживання й невисокої вартості дозволяє використовувати їх не тільки у високій сфері телекомунікацій, але й у стільникових телефонах і роботах-іграшках.

Відзначимо основні особливості архітектури процесорів цифрової обробки сигналів:

1. *Гарвардська архітектура*, основу якої становить фізичний і логічний поділ пам'яті команд і пам'яті даних. Ключові команди *DSP-процесора* є багатоперандними, і прискорення їх роботи вимагає одночасного читання декількох комірок пам'яті. Відповідно на кристалі є роздільні шини адреси й даних (у деяких типах процесорів – кілька шин адреси й даних). Це дозволяє сполучати в часі вибірку операндів і виконання команд. Використання *модифікованої гарвардської архітектури* припускає, що операнди можуть зберігатися не тільки в пам'яті даних, але й у пам'яті команд разом із програмами. Наприклад, у випадку реалізації цифрових фільтрів коефіцієнти можуть зберігатися в пам'яті програм, а значення даних – у пам'яті даних. Тому коефіцієнт і дані можуть вибиратися в одному машинному циклі. Для забезпечення вибірки команди в тому ж машинному циклі використовується або кеш-пам'ять програм, або дворазове звертання до пам'яті програм за час машинного циклу.

2. Для скорочення часу виконання однієї з основних операцій цифрової обробки сигналу – множення – застосовується апаратний помножувач. У процесорах загального призначення ця операція реалізується за кілька тактів зсуву й додавання й займає багато часу, а в *DSP-процесорах* завдяки спеціалізованому помножувачу потрібний усього один цикл. Вбудована схема апаратного мно-

ження дозволяє виконати за 1 такт основну операцію ЦОС - *множення з накопиченням* (Multiply-accumulate – MAC) для 16- і/або 32-розрядних операндів.

3. Апаратна підтримка циклічних буферів. Наприклад, для фільтра при кожному обчисленні відліку вихідного сигналу використовується новий відлік вхідного сигналу, який зберігається в пам'яті на місці самого старого. Для такого циркулюючого буфера може використовуватися фіксована область ОЗП. При цьому під час обчислень генеруються лише послідовні значення адрес ОЗП незалежно від того, яка операція – запис або читання – виконується в даний момент. Апаратна реалізація циклічних буферів дозволяє встановити параметри буфера (адреса початку, довжина) у програмі поза тілом циклу фільтрації, що дозволяє скоротити час виконання циклічної ділянки програми.

4. Скорочення тривалості командного такту. Це багато в чому забезпечується прийомами, характерними для RISC-процесорів. Головними з них є розміщення операндів більшості команд у регістрах, а також конвеєризація на рівні команд і мікрокоманд. Конвеєр має від 2 до 10 щаблів, що дозволяє на різних стадіях виконання одночасно обробляти до 10 команд. При цьому використовується генерація адрес регістрів паралельно з виконанням арифметичних операцій, а також багатопортовий доступ до пам'яті. Сюди ж можна віднести й прийом, характерний для універсальних мікропроцесорів з *EPIC-архітектурою*, як застосування команд із надвеликою довжиною слова (VLIW), що генеруються на стадії компіляції програми. Цьому ж служить і розглянута вище *гарвардська архітектура* процесора, характерна для однокристальних мікроконтролерів.

5. Наявність на кристалі процесора внутрішньої пам'яті, що споріднює ЦСП із однокристальними МК. Вбудована в процесор пам'ять звичайно має значно більшу швидкодію, ніж зовнішня. Наявність вбудованої пам'яті дозволяє значно спростити систему в цілому, зменшити її розміри, енергоспоживання й вартість. Ємність внутрішньої пам'яті є результатом певного компромісу. Її збільшення веде до подорожчання процесора й збільшує енергоспоживання, а обмежена ємність пам'яті програм не дозволяє зберігати складні алгоритми. Більшість DSP-процесорів з фіксованою крапкою мають малу ємність внутрішньої

пам'яті, звичайно від 4 до 256 Кбайт, і невисоку розрядність зовнішніх шин даних, що зв'язують процесор із зовнішньою пам'яттю. У той же час ЦСП із плаваючою крапкою звичайно передбачають роботу з великими масивами даних і складними алгоритмами й мають або вбудовану пам'ять великої ємності, або більшу розрядність адресних шин для підключення зовнішньої пам'яті (а іноді й те, і інше).

6. Широкі можливості по апаратній взаємодії із зовнішніми пристроями, що включають:

- велику різноманітність інтерфейсів, у тому числі контролери локальної промислової мережі CAN, вбудовані комунікаційні (SCI) і периферійні (SPI) інтерфейси, I2C, UART;

- кілька входів для аналогових сигналів і, відповідно, вбудований АЦП;
- вихідні канали широтно-імпульсної модуляції (ШІМ);
- розвинену систему зовнішніх переривань;
- контролери прямого доступу до пам'яті.

7. У деяких DSP-сімействах передбачені спеціальні апаратні засоби, що полегшують створення мультипроцесорних систем з паралельною обробкою даних для нарощування продуктивності.

8. DSP-процесори широко використовуються в мобільних пристроях, де споживана потужність є основною характеристикою. Для зниження енергоспоживання в сигнальних процесорах застосовується безліч методів, у тому числі зменшення напруги живлення й введення функцій керування споживанням, наприклад, динамічна зміна тактової частоти, перемикання в сплячий або черговий режим або відключення не використовуваної в цей момент периферії. Слід зазначити, що ці заходи впливають на швидкість роботи процесора й при некоректному використанні можуть привести до непрацездатності проектованого пристрою (як приклад можна згадати деякі стільникові телефони, які в результаті помилок у програмах керування енергоспоживанням іноді переставали включатися) або до погіршення його експлуатаційних характеристик (напри-

клад, значному часу відновлення працездатності при виході зі сплячого режиму).

Система команд сигнальних процесорів має багато рис систем команд універсальних мікропроцесорів (особливо з RISC-архітектурою) і однокристальних мікроконтролерів. Вона містить у собі основні арифметичні й логічні операції й команди переходів, але в меншій, ніж в універсальних МП, кількості. Число режимів адресації операндів також відносно невелике. Команда має простий, чітко заданий формат. Довжина команди становить одне, рідше два 16-розрядні слова. Однак поряд з використанням скороченого набору команд, в DSP-процесорах застосовуються й такі характерні для MMX-обробки апаратно підтримувані інструкції, як команди пошуку мінімуму й максимуму, одержання абсолютного значення, додавання з насиченням, при якому у випадку переповнення при додаванні двох чисел результату привласнюється максимально можливе в даній розрядній сітці значення. Це веде до зменшення кількості конфліктів у конвеєрі й підвищує ефективність роботи процесора.

З іншого боку, ЦСП містять ряд команд, наявність яких обумовлено специфікою їх застосування і які внаслідок цього рідко присутні в мікропроцесорах інших типів. Насамперед це, звичайно, команда множення з накопиченням суми MAC, що лежить в основі ЦОС: $A = B * C + A$. У системах команд деяких сигнальних процесорів можна при програмуванні вказати число виконань цієї команди в циклі й правила зміни індексів для адресації операндів B и C. При цьому на відміну від команд повторення звичайних процесорів сигнальний процесор може апаратно підтримувати перевірку умови завершення циклу. Сюди ж можна віднести й команди зрушення (перезаписи) у сусідню комірку ОЗП даних, що підтримують роботу циклічного буфера для підготовки множення в наступному такті.

Для ефективної реалізації алгоритмів ШПФ у систему команд деяких DSP-процесорів включені можливості адресації з реверсуванням біт адреси.

Програмування мікропроцесорів цього класу також має свої особливості. Значна зручність для розроблювача, що звичайно пов'язана з використанням

мов високого рівня, у більшості випадків обертається одержанням менш компактного й швидкого коду. Так як особливості ЦОС передбачають роботу в реальному часі, це приводить до необхідності використання для розв'язку тих же завдань могутніших і дорогих DSP. Така ситуація особливо критична для багатосерійної продукції, де різниця у вартості більш продуктивного DSP або додаткового процесора відіграє важливу роль. У той же час у сучасних умовах швидкість розробки (і, отже, виходу нового виробу на ринок) може принести більше вигід, ніж витрати часу на оптимізацію коду при написанні програми на асемблері.

Компромісним підходом тут служить використання асемблера для написання найбільш критичних з погляду час- і ресурсоемності ділянок програми, у той час як основна частина програми пишеться мовою високого рівня, як правило, С або С++.

Припустимо, що програма на С пишеться в 5 раз швидше, ніж на асемблері, а одержуваний у результаті трансляції код працює приблизно в стільки ж раз повільніше.

Тоді у випадку, коли деяка ділянка програми обсягом 20 % коду займає 80 % часу виконання програми, що буває досить часто ("правило 80/20"), одержимо, що програмування цього фрагмента на асемблері приведе до збільшення часу програмування на 80 %, але зменшить час роботи програми майже втричі (табл. 15.1).

Таблиця 15.1.

Аналіз ефективності різних методів програмування

	Вся програма на С	Вся програма на асемблері	Програма на С із фрагментом на асемблері
Час програмування(в умовних одиницях)	100	500	180
Час виконання програми (в умовних одиницях)	100	20	36

От чому програми для процесорів, що мають критичне значення часу роботи, хоча в більшості випадків і написані на С, але містять асемблерні вставки.

До цікавих особливостей DSP-асемблерів можна віднести наявність двох форм запису багатьох команд - мнемонічної й алгебраїчної. Мнемонічна форма аналогічна запису команд для звичайних мікропроцесорів, наприклад, ADD dst, src. Інша, алгебраїчна форма в асемблерах мікропроцесорів інших класів використовується рідше, у той час як на DSP-асемблері згадана команда може бути записана у вигляді $DSt = DSt + src$.

Звичайно асемблери DSP розуміють обидві форми запису, але, наприклад, асемблери ADI і Lucent застосовують тільки алгебраїчний запис.

З ростом числа областей застосування ЦОС і складності алгоритмів обробки зростають вимоги до сигнальних процесорів у плані підвищення швидкодії й оснащеності інтерфейсними й іншими спеціалізованими вузлами. У цей час на ринку присутні безліч типів ЦСП, як універсальних, так і орієнтованих на досить вузьке коло завдань. Звісно, жоден із процесорів не може бути оптимальним для всіх додатків. Тому перше завдання розроблювача - вибір процесора, найбільш підходящого по продуктивності, ціні, наявності певної периферії, споживаної потужності, простоті використання й іншим критеріям.

Наприклад, для таких портативних пристроїв, як мобільні телефони, портативні цифрові плеєри, першорядними є вартість, ступінь інтеграції й споживана потужність, а максимальна продуктивність найчастіше не потрібна, тому що звичайно спричиняє значне підвищення споживаної потужності, не даючи переваг при обробці відносно низькошвидкісних аудіоданих. У той же час для гідроакустичних або радіолокаційних систем визначальними параметрами є швидкість роботи, наявність високошвидкісних інтерфейсів, а вартість є другим критерієм.

Хоча більшість фірм випускає широку номенклатуру процесорів, які можуть бути використані для різних вживань, серед них спостерігається певна спеціалізація. Так, слід зазначити перевагу процесорів Analog Devices для додатків, що вимагають виконання більших обсягів математичних обчислень (таких

як цифрова фільтрація сигналу, обчислення кореляційних функцій і т.п.), оскільки їх продуктивність на подібних завданнях вище, ніж у процесорів компаній Freescale і Texas Instruments. У той же час для завдань, що вимагають виконання інтенсивного обміну із зовнішніми пристроями (багатопроцесорні системи, різного роду промислові контролери), переважніше використовувати процесори Texas Instruments, що володіють високошвидкісними інтерфейсними підсистемами. Компанія Freescale є лідером за обсягом виробництва щодо дешевих і досить продуктивних 16- і 24-розрядних сигнальних процесорів з фіксованою крапкою.

Розглянемо структуру й основні характеристики DSP-процесора на прикладі *процесора цифрової обробки* сигналів TMS320F2833x фірми Texas Instruments.

Сімейство цих процесорів являє собою 32-розрядні цифрові сигнальні процесори, що поєднують у собі потужні й ефективні технології цифрової обробки сигналів з багатим набором периферійних пристроїв і простотою використання. Дана платформа процесорів споконвічно створювалася для додатків керування електродвигунами й електроприводами.

У процесі розвитку вони здобули більш розвинену периферію й потужне 32-розрядне ядро сигнального процесора, за рахунок чого область їх застосування значно розширилася й у цей час містить у собі також ті області, де раніше традиційно застосовувалися потужні мікроконтролери.

У цей час сімейство містить у собі моделі TMS320F2832, TMS320F2833 і TMS320F2835, що відрізняються частотою синхронізації, ємністю внутрішньої пам'яті й деякими іншими кількісними параметрами.

Надалі ми будемо опиратися на характеристики ЦСП моделі TMS320F2835, структуру котрого наведено на рис. 15.12.

Процесор містить блок обробки чисел із плаваючою крапкою, апаратний помножувач, що дозволяє виконувати за один цикл операцію типу 32х32 біт MAC або дві операції типу 16х16 біт MAC, а також так зване атомарне АЛП, яке забезпечує виконання за 1 цикл такі операції, як читання-модифікація-запис

по кожній з адрес пам'яті, що дозволяє значно підвищити продуктивність і ефективність коду.

До складу процесора входять різноманітні запам'ятовуючі пристрої:

- однопортове ОЗП (SARAM) обсягом 34К*16 слів; воно складається з 8 блоків з організацією 4К 16-розрядних слів кожний і 2 блоки ємністю по 1К*16 байт, у яких може зберігатися таблиця векторів переривань;
- ПЗП (у цьому процесорі позначається як OTP - one-time programmable): 1К*16 розрядів;

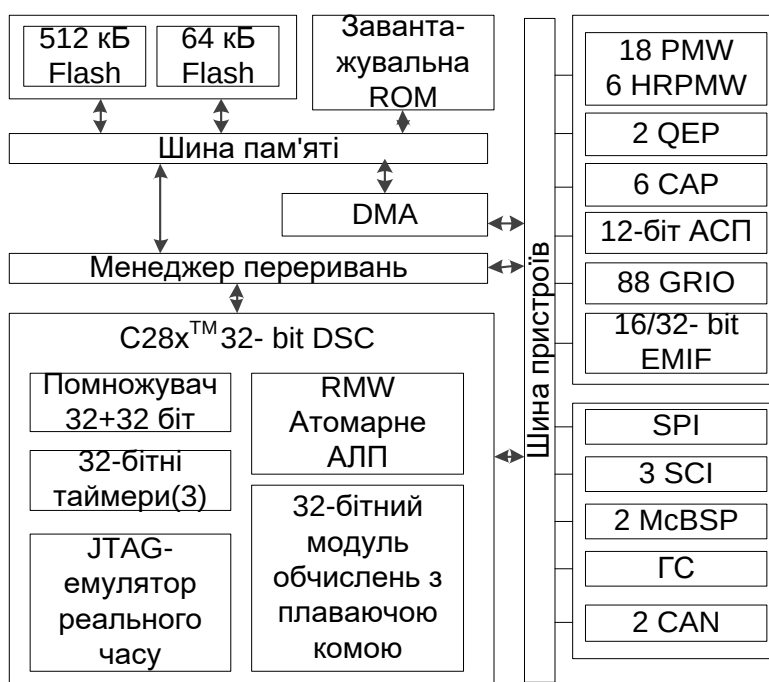


Рис. 15.12. Цифровий сигнальний процесор TMS320F2835

- модуль FLASH-пам'яті ємністю 512 Кбайт, який дозволяє розробнику багаторазово змінювати програму в процесорі, у тому числі й безпосередньо в готовому виробі, що забезпечує максимальну легкість і зручність відновлення програмних версій приладів. FLASH-пам'ять розбита на сектори, що дозволяє користувачеві програмувати лише частину пам'яті, не здійснюючи попереднього повного стирання. Однак слід зазначити, що FLASH-пам'ять працює повільніше, ніж ядро процесора і його основна пам'ять (пам'ять програм і пам'ять да-

них), тому деякі частини коду, такі як функції переривань, а також функції, критичні до швидкості виконання, не можуть виконуватися з FLASH;

- завантажувальна пам'ять Boot ROM обсягом 8К слів по 16 розрядів.

Записана тут мікропрограма виконується ядром процесора щораз при подачі живлення на процесор, а також після скидання. Результатом її виконання є передача функцій завантаження програми, яке ЦСП повинен виконати, певному периферійному пристрою, підключеному по інтерфейсу SPI, I2C і т.п., або внутрішньої або зовнішньої FLASH-пам'яті. Тут же втримуються таблиці нормалізованих значень математичних функцій, таких як синус, косинус, а також *таблиці векторів переривань*.

FLASH -Пам'ять, ПЗП й блоки L0-L7 ОЗП захищені 128-розрядним секретним ключем.

Пам'ять має можливість розширення до 2М*16 слів за допомогою 16/32-розрядної шини EMIF.

Система команд для роботи із числами у форматі із плаваючою крапкою є функціонально повною. Крім звичайних арифметичних команд і команд конвертування форматів (з формату з фіксованою у формат із плаваючою крапкою й назад), система команд містить у собі також команди одержання першого наближення зворотного значення числа у форматі із плаваючою крапкою й квадратного кореня. Тим самим апаратно забезпечується висока ефективність будь-яких операцій із плаваючою крапкою, включаючи розподіл. Компілятор C/C++ автоматично оптимізує програму користувача з урахуванням можливого паралельного виконання команд, додаючи, як правило, команди, що не конфліктують між собою, завантаження операндів, які знадобляться на наступних етапах обчислень (VLIW-підхід). Ці апаратні можливості підтримуються компілятором C/C++. На апаратному рівні підтримуються як команди повторення окремої інструкції, так і команди повторення блоку коду.

Усі процесори сімейства F2833x є високоінтегрованими пристроями й містять:

1. Послідовні інтерфейси, що включають:

- 2 мережних порти CAN, що дозволяють організувати дві незалежно працюючі CAN-мережі - для підключення внутрішніх інтелектуальних пристроїв і зовнішніх, наприклад, для зв'язку із системами керування верхнього рівня;

- 3 порти асинхронного послідовного інтерфейсу SCI (UART), використовуваних для міжпроцесорного або іншого асинхронного зв'язку в дуплексному й напівдуплексному режимах;

- порт синхронного послідовного інтерфейсу (SPI);

- шину I2C;

- 2 буферованих послідовних порти McBSP;

- 88 послідовних портів введення-виведення загального призначення (GPIO) з незалежним тактуванням і можливістю багатоканальних режимів роботи з підтримкою блокової передачі по кожному каналу;

- порт JTAG, що дозволяє налагоджувати систему в реальному часі;

2. 12-розрядний 16-канальний АЦП згодом перетворення сигналу 80 нс, що на сьогоднішній день є кращим результатом серед АЦП, вбудованих у ЦСП.

3. 6 контролерів *прямого доступу до пам'яті*. Передача даних по каналу ПДП виконується пакетами. Довжина кожного пакета - не більше 32 16-розрядних слів. Загальне число пакетів у передачі – до 65535. По завершенню передачі канал ПДП може генерувати запит переривання для нової ініціалізації контролера ПДП.

У процесі передачі даних можлива їхня ортогональна перестановка для наступного прискорення обробки даних центральним процесором. Канали працюють у режимі або у такому, що змінюється циклічно, або фіксованих пріоритетів. В останньому випадку процес передачі даних по низькопріоритетним каналам може перериватися й відновлюватися при завершенні високопріоритетної передачі.

4. Три 32-розрядних таймери.

5. Сторожовий таймер, що запобігає зависанню системи.

6. Шість 32-розрядних каналів захоплення зовнішніх подій (enhanced capture - ECAP), використовуваних у системах, де важливо точне узгодження в часі зовнішніх подій, наприклад, при вимірі періоду проходження сигналів;

7. Два 32-розрядних модуля EQEP(enhanced quadrature encoder pulse), призначених для підтримки вимірів швидкості й частоти, детектування заклинювання ротора й визначення його положення, контролю несправностей. Можуть використовуватися як додаткові модулі CAP;

8. Розширений блок широтно-імпульсної модуляції (enhanced pulse-width modulator - EPWM), що забезпечує функціонування 18 каналів ШІМ, 6 з яких працюють на підвищеній частоті з роздільною здатністю 150 пс. Широкі можливості цього блоку пов'язані з основними областями застосування даного сигнального процесора: автомобільна промисловість, промислова автоматика, вимірювальні прилади, перетворювачі потужності.

Розглянемо роботу цього блоку більш докладно.

Кожний модуль ШІМ-генератора містить базовий таймер, блок порівняння, конструктор вихідних ШІМ-сигналів. Модуль має два виходи EPWMA і EPWMB, які можна використовувати або незалежно один від одного, або в парі.

Базовий таймер модуля працює в режимах нереверсивного й реверсивного лічильника. Період ШІМ задається в регістрі періоду, а початковий стан лічильника апаратно завантажується з регістру фазового зсуву. Тим самим забезпечується можливість формування різними каналами ШІМ-сигналів, зсув одного щодо другого на будь-який заданий кут.

Відмінною рисою блоку є два незалежні канали з регістрами завдання двох уставок порівняння CMPA і CMPB. Конструктор ШІМ-сигналів для кожного із двох виходів дозволяє по кожній з п'яти вхідних подій згенерувати наступні вихідні події: установити високий або низький рівень сигналу; перемкнути з високого на низький рівень або назад; залишити стан виходу незмінним. Щоб сконструювати вихідні періодичні сигнали, розробнику потрібно розставити на "опорній цифровій пилці" графічні позначення необхідних вихідних

подій і відповідно до них проініціалізувати ШІМ-генератор. На рис. 15.13 як приклад наведено формування двох ШІМ-сигналів.

Вхідні події в порівнянні А нагору (CA↑) і вниз (CA↓) управляють фронтами першого ШІМ-сигналу, а в порівнянні В нагору (CB↑) і вниз (CB↓) - фронтами другого ШІМ-сигналу.

Є можливість по потрібній події (наприклад, " по періоду") генерувати запит переривання й у процедурі обслуговування цього переривання змінювати параметри ШІМ-сигналів (період, скважність).

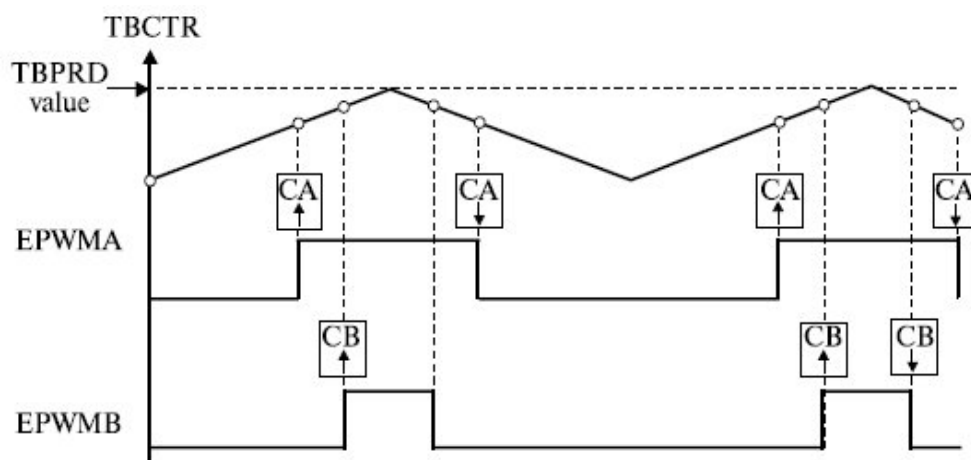


Рис. 15.13. Конструювання вихідних ШІМ-Сигналів

Структура процесора забезпечує швидку реакцію на переривання і їх обробку. Запити переривань генеруються як зовнішніми пристроями, так і модулями самого процесора. Блок обробки зовнішніх переривань (Peripheral Interrupt Expansion - PIE) використовується для проведення арбітражу *запитів переривань* від зовнішніх пристроїв і підтримує до 8 запитів зовнішніх переривань, що маскуються, які можуть надходити по кожній з 64 ліній портів загального призначення GPIO0-GPIO63, а також запит по зовнішньому входу XNMI.

Кожне переривання може бути налаштоване на спрацьовування по високому або низькому рівню сигналу й дозволене або заборонене до обслуговування.

Блок обробки переривань для кожного із зовнішніх запитів містить 16-розрядний лічильник, який може застосовуватися для оцінки кількості переривань даного типу.

У якості джерел внутрішніх переривань виступають таймери, канали ПДП і інші модулі процесора.

Кожне з переривань має свій *вектор переривань*, який може бути змінений. Пріоритети переривань встановлюються як на апаратному, так і на програмному рівнях.

Напруга живлення процесора становить 1,9 В, а напруга зовнішніх портів 3,3 В.

У сімействі процесорів 320F28x більше уваги приділяється зниженню енергоспоживання (до 0,05 мВт на 1 млн операцій у секунду). Це досягається за рахунок відключення неактивних пристроїв і можливістю керування енергоспоживанням з боку користувача (до 64 режимів енергоспоживання).

Наявність модуля підтримки обчислень із плаваючою комою здорожчує процесор, тому для розробників, що працюють винятково у форматі з фіксованою крапкою, пропонуються крім процесорів серії 2833x точно такі ж вироби, але без цього модуля: 28235, 28234, 28232.

Цифровий сигнальний процесор ADSP BF561

У ряді завдань, наприклад, у завданнях обробки відеозображень із високою роздільною здатністю, потужності одного, навіть високопродуктивного процесора виявляється недостатньо. Розв'язок цієї проблеми йде по двом напрямкам.

По-перше, це створення *мультипроцесорних систем*. У деяких DSP-сімействах для нарощування продуктивності передбачені спеціальні апаратні засоби, що полегшують створення мультипроцесорних систем з паралельною обробкою даних.

По-друге, це розробка багатоядерних цифрових сигнальних процесорів, аналогічно такому ж напрямку в розвитку архітектури універсальних МП. Розг-

ляємо другий підхід на прикладі цифрового сигнального процесора ADSP BF561 фірми Analog Devices.

Основні функціональні блоки процесора наведено на рис. 15.14. Основу процесора становлять два обчислювальні ядра BLACKFIN, кожне з яких має свій набір ресурсів, що включає обчислювальний блок, блок керування й набір регістрів.

Основу обчислювального блоку становлять два 16-розрядних MAC пристрої. Так як процесор працює на частоті 600 МГц, це забезпечує виконання MAC-операцій з еквівалентною частотою 2,4 ГГц (по 2 MAC-пристрої в кожному із двох ядер).

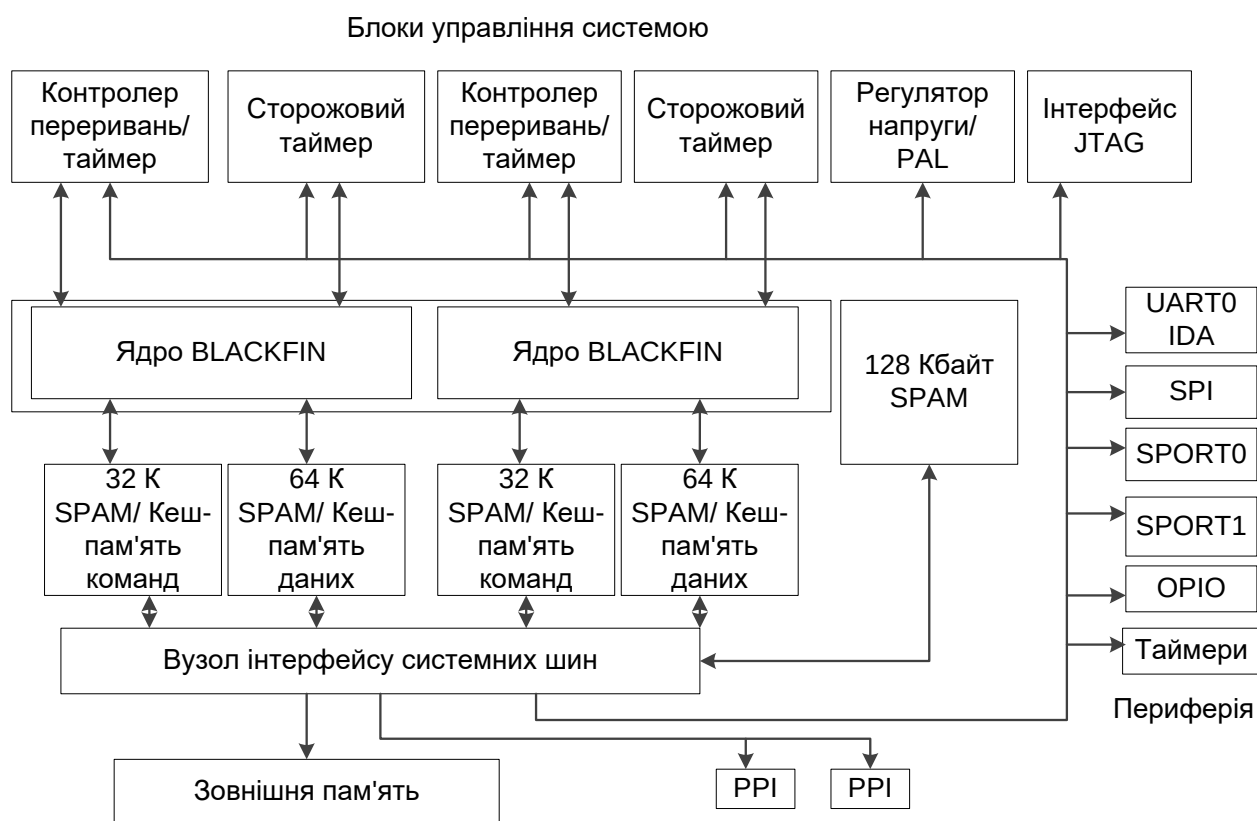


Рис. 15.14. Структура процесора ADSP BF561

Крім цього в обчислювальний блок кожного з ядер входять наступні пристрої:

- два 40-розрядних АЛП для виконання звичайних арифметичних і логічних операцій над 16- і 32-розрядними даними;
- 40-розрядний пристрій зсуву, що виконує крім власно зсуву різних типів також нормалізацію й добування експоненти, обробку окремих біт або наборів біт;
- чотири 8-розрядних відео-АЛП, які підтримують деякі операції над 8-розрядними даними, характерні для завдань обробки відеозображень;
- блок регістрів загального призначення, що включає вісім 32-розрядних регістри, кожний з яких залежно від конкретної операції може розглядатися як одне 32-розрядне слово або як два 16-розрядних.

Кожне ядро процесора має власну пам'ять ємністю 100 Кбайт, доступну тільки йому й працюючу на частоті ядра: 32 Кбайт пам'яті команд, з яких 16 Кбайт можуть бути сконфігуровані як кеш команд, 64 Кбайт пам'яті даних, з яких 32 Кбайт можуть бути сконфігуровані як кеш даних, і 4 Кбайта спеціального ОЗП для збереження контексту (scratchpad).

На кристалі також є менш швидкодіюча (працююча на половині частоти ядра) пам'ять другого рівня обсягом 128 Кбайт, доступна обом ядрам. У ній можуть зберігатися як команди, так і дані.

До процесора через 32-розрядну шину даних може бути підключене до чотирьох банків зовнішньої пам'яті ємністю від 16 до 128 Мбайт кожний із загальною ємністю до 512 Мбайт.

Окремого адресного простору введення/виведення в процесорі немає, і адресація до всіх ресурсів проводиться за допомогою єдиної 32-розрядної адреси.

Процесор має 10-рівневий конвеєр команд.

Процесор ADSP BF561 має дуже багатий набір інтегрованих периферійних вузлів, частина з яких входить до складу кожного з ядер, а інші є загальними ресурсами процесора. Сюди відносяться:

- 14 таймерів (32-розрядних), 12 з яких є загальними й можуть також програмуватися як ШІМ-генератори, а ще два використовуються кожним з ядер;
- два сторожові таймери (по одному в кожному з ядер);
- 4-канальний внутрішній контролер ПДП для прискорення обміну між блоками пам'яті усередині процесора;
- 32- і 16-розрядний контролери ПДП; кожний із цих контролерів має по 12 каналів для пересилань між внутрішньою пам'яттю процесора й периферією й по 4 канали для пересилань між внутрішньою й зовнішньою пам'яттю;
- контролер переривань системи, який перетворює усі переривання від периферійних вузлів, контролерів ПДП і зовнішні переривання (усього 61 джерело) у переривання ядра загального призначення;
- по одному контролеру переривань у кожному з ядер, які підтримують 9 переривань загального призначення й виділені переривання: скидання, емуляції, переривання, що не маскуються, переривання від таймерів і т.д.;
- два 16-розрядних паралельних порти PPI, які дозволяють без використання додаткової логіки підключати до процесора багато стандартних АЦП, ЦАП, відеокодера й декодери;
- 48 програмувальних портів введення-виведення загального призначення (GPIO);
- набір стандартних послідовних інтерфейсів - SPI, SPORT і UART (з підтримкою IrDA) і т.д.

Процесор підтримує режими прямої адресації, непрямої адресації з пред- і постінкрементом і специфічні для завдань цифрової обробки сигналів режими циклічної адресації й адресації з реверсуванням біт адреси.

Система команд процесора, аналогічно RISC-мікропроцесорам, має структуру, орієнтовану на регістри й підтримку мов високого рівня.

Підсумки. Розглянуто особливий клас мікропроцесорних ВІС - цифрові сигнальні процесори. Наведені приклади цифрової обробки сигналів і використання при цьому дискретного перетворення Фур'є. Виділені особливості

ЦСП, що впливають із області їх застосування, що й суттєво впливають на їх архітектуру.

Розглянута архітектура сигнальних мікропроцесорів TMS320F2833x фірми Texas Instruments і ADSP BF561 фірми Analog Devices, що мають найкращі характеристики, у цей час, у цьому класі мікропроцесорів.

КОНТРОЛЬНІ ПИТАННЯ

1. В чому полягає ідеологія RISC-архітектури?
2. Перелічіть, на які принципи опирається RISC-архітектура?
3. Які переваги RISC-архітектури має в порівнянні з CISC-архітектурою?
4. Зазначте, які 8 конвеєризованих виконавчих пристроїв (executive unit - EU) є у складі кожного ядра POWER 4?
5. Які класи архітектури команд виділяються на основі числа потоків команд і потоків даних?
6. Охарактеризуйте клас архітектури команд SISD.
7. Охарактеризуйте клас архітектури команд MISD.
8. Охарактеризуйте клас архітектури команд SIMD.
9. Охарактеризуйте клас архітектури команд MIMD.
10. Наведіть класифікацію багатопроцесорних і багатомашинних систем на основі ступеня поділу обчислювальних ресурсів системи.
11. Що являє собою SMP-система?
12. Дайте визначення поняттю «кластер». Наведіть характеристику кластерних систем.
13. Що таке трансп'ютер? Для чого використовуються трансп'ютерні системи?
14. Що таке цифрова обробка сигналів?
15. Що являють собою процесори цифрової обробки сигналів?
16. Що вам відомо про перетворення Фур'є? Дайте визначення, основні характеристики.
17. Дайте визначення дискретному перетворенню Фур'є.
18. Дайте визначення прямому і зворотному дискретному перетворенню Фур'є.

19. Визначте основні особливості архітектури процесорів цифрової обробки сигналів.

20. Які запам'ятовуючі пристрої входять до складу цифрового сигнального процесора.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Приходько В. М. Комп'ютерна схемотехніка / В. М. Приходько, С. П. Євсєєв, К. В. Садовий. – Х. : Вид. ХНЕУ, 2011. – 299 с.
2. Євсєєв С. П. Архітектура мікропроцесорів та компонентів ЕОМ / С. П. Євсєєв, О. А. Смірнов, О. Г. Король та ін. // Кіровоград: Вид. Лисенко В. Ф. – 2015. – 550 с.
3. Бабак В. П. Обробка сигналів : підручник / В. П. Бабак, В. С. Хандецький, Е. Шрюфер. – 2-ге вид., перероб. і доп. – К.: Либідь, 1999. – 496 с.
4. Бабич М. П. Комп'ютерна схемотехніка: навч. посібн. / М. П. Бабич, І. А. Жуков. – К.: МК-Прес, 2004. – 412 с.
5. Васильєва Л. Д. Напівпровідникові прилади/ Л. Д. Васильєва, Б. І. Медведєнко, Ю. І. Якименко. – К.: ІВЦ Видавництво "Політехніка", 2003. – 256 с.
6. Приходько В. М. Комп'ютерна схемотехніка: навчально-практичний посібник/ В. М. Приходько, В. Ф. Третьак, С. В. Осієвський. –Харків: Вид. ХНЕУ, 2008. – 208.
7. Соловей О. І. Інженерна графіка: схеми електричні: навч. посібн. для вищ. закл. освіти / О. І. Соловей, О. С. Хмеленко. – К.: Кондор, 2005. – 187 с.
8. Каганюк О.К., Поліщук М.М. Комп'ютерна схемотехніка: Навчальний посібник. – Луцьк: РРВ Луцького НТУ, 2016. – 236 с
9. Воробйова О.М. Цифрові пристрої: навч. посіб. – Ч. 1 / О.М. Воробйова, М.П. Савицька, Ю.В. Флейта. – Одеса: ОНАЗ ім. О.С. Попова, 2016. – 112 с.
10. Комп'ютерна схемотехніка : підручник / [Азаров О. Д., Гарнага В. А., Клятченко Я. М., Тарасенко В. П.]. – Вінниця : ВНТУ, 2018. – 230 с.
11. Абель П. Асемблер: мова і програмування для IBM PC: підручник. – К.: Век+, 2003. – 734 с.
12. Корнєв В. П. Курс лекцій з дисципліни «Обчислювальні та мікропроцесорні засоби електронних апаратів». Частина 1. «Архітектура та програмування процесорів Intel». К.: НТУУ «КПІ» , 2006 – 2010 р.р. – Електронне навчально-методичне видання на сервері кафедри КЕОА.
13. Поворознюк А. І. Архітектура комп'ютерів ч.1: навч. посібник. – Х.: Торнадо, 2004. – 355 с.
14. Якименко Ю. І., Терещенко Т. О. Мікропроцесорна техніка : Підручник. –К.: Видавництво "Політехнік", 2003. – 440 с.

15. Архітектура обчислювальної системи. Класифікація комп'ютерів. – [Електронний ресурс]. – Режим доступу до ресурсу: <http://www.uatur.com/html/informatika/lecture3.htm>.

16. Багатоядерні процесори: теорія (на прикладі Intel, AMD). [Електронний ресурс]. – Режим доступу до ресурсу: <http://novikov.ua/bnews/2007/07/20/5945/>

17. Гетерогенна мультиядерна архітектура мікропроцесорів Intel. – [Електронний ресурс]. – Режим доступу до ресурсу: <http://www.ixbt.com/news/hard/index.shtml?06/40/96>.

18. Sutter H. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software / / Dr. Dobbs's Journal, 30 (3), March 2005. [Електронний ресурс]. – Режим доступу до ресурсу: <http://www.gotw.ca/publications/concurrency-DDJ.htm>.

ЗМІСТ

Вступ	2
1. ФОРМИ ЗОБРАЖЕННЯ ІНФОРМАЦІЇ.....	5
1.1. Історія розвитку засобів комп'ютерної техніки.....	5
1.2. Характеристика електричних сигналів	6
1.3. Проходження прямокутних імпульсів через RC-ланцюжки.....	9
2. ІНТЕГРОВАНІ СИСТЕМИ ЕЛЕМЕНТІВ	15
2.1. Принципи схемотехніки цифрових інтегральних схем.....	15
2.2. Транзистор як технічна основа реалізації логічних функцій	15
2.2.1. Ключ із спільним емітером	16
2.2.2. Ключі на МОН-транзисторах.....	24
2.2.3. Перемикач струму.....	28
2.3. Класифікація й основні параметри базових логічних елементів	30
2.4. Базовий логічний елемент транзисторно-транзисторної логіки.....	34
2.5. Базовий логічний елемент емітернозв'язаної логіки.....	44
2.6. Елементи на МОН-транзисторах.....	46
2.6.1. Елементи на однотипних МОН-транзисторах	46
2.6.2. Елементи на комплементарних МОН-транзисторах	47
2.7. Базовий логічний елемент інжекційної логіки.....	52
2.8. Порівняльна оцінка базових логічних елементів.....	56
3. ЛОГІЧНІ ОСНОВИ ПОБУДОВИ ЕЛЕМЕНТІВ.....	61
3.1. Логічні основи побудови елементів	61
3.2. Логічні елементи. Взаємна відповідність булевих функцій та логічних елементів	64
3.3. Основні аксіоми і закони булевої алгебри.....	66
3.4. Мінімізація логічних функцій за допомогою карт Карно.....	71
3.5 Синтез комбінаційних пристроїв.....	82
4. ЦИФРОВІ ПРИСТРОЇ КОМБІНАЦІЙНОГО ТИПУ	91
4.1. Дешифратори.....	91
4.2. Шифратори	98
4.3. Мультиплексори.....	102
4.4. Демультиплексори	109
4.5. Суматори.....	116
4.6. Компаратори.....	123
4.7. Програмовані логічні матриці	127
5. СХЕМОТЕХНІКА ЦИФРОВИХ ЕЛЕМЕНТІВ	134
5.1. Поняття про цифрові елементи з пам'яттю.....	134
5.2. Тригери на логічних елементах	136

5.2.1. RS-тригери.....	137
5.2.2. Універсальні тригери (JK-тригери).....	142
5.2.3. D-тригери.....	147
5.2.4. Т-тригери.....	152
5.3. Синтез тригерів із заданими умовами функціонування.....	153
6. СХЕМОТЕХНІКА ЦИФРОВИХ ВУЗЛІВ.....	158
6.1. Регістри.....	158
6.2. Регістри зсуву.....	160
6.3. Лічильники імпульсів.....	164
6.3.1. Лічильники з послідовним перенесенням.....	166
6.3.2. Лічильники з наскрізним перенесенням.....	169
6.3.3. Лічильники з паралельним перенесенням.....	171
6.4. Дільники імпульсів.....	173
7. ПЕРЕТВОРЕННЯ АНАЛОГОВИХ СИГНАЛІВ У ЦИФРОВІ.....	178
7.1. Загальні відомості про перетворювачі.....	178
7.2. Цифроаналогові перетворювачі.....	183
7.2.1. Основні параметри та класифікація.....	183
7.2.2. ПКН на матриці резисторів з ваговими опорами.....	187
7.2.3. ПКН на матриці резисторів $R - 2R$	189
7.2.4. Особливості застосування ЦАП.....	192
7.3. Аналого-цифрові перетворювачі.....	193
7.3.1. Основні параметри та класифікація.....	193
7.3.2. Перетворювачі напруги в код послідовного рахунку.....	196
7.3.3. Перетворювачі напруги в код порозрядного кодування.....	200
7.3.4. Паралельні перетворювачі напруги в код.....	205
7.4. Основні напрями розвитку перетворювачів інформації.....	208
8. СХЕМОТЕХНІКА КОМБІНАТОРНИХ ВУЗЛІВ.....	211
8.1. Комбінаторні вузли.....	211
8.1.1. Кодування чисел для виконання арифметичних операцій.....	211
8.1.2. Виконання операцій над двійковими числами.....	214
8.1.3. Реалізація множення та ділення.....	216
8.1.4. Арифметико-логічні пристрої.....	219
8.2. Таймери.....	222
9. ПРОГРАМУВАЛЬНІ ЛОГІЧНІ ІНТЕГРАЛЬНІ СХЕМИ (ПЛІС).....	228
9.1. Загальні відомості.....	228
9.2. ПЛІС типу CPLD.....	229
9.3. ПЛІС типу FPGA.....	232
9.4. ПЛІС типу SOC.....	236

9.5. Проектування цифрових пристроїв на ПЛІС	238
9.5.1. Засоби описування проекту	238
9.5.2. Мова опису апаратури AHDL	239
9.5.3. Мова опису апаратури VHDL	240
9.5.4. Мова опису апаратури Verilog HDL.....	241
9.5.5. Система проектування MAX+PLUSII.....	242
10. ЦИФРОВІ КОМП'ЮТЕРИ	246
10.1. Арифметико-логічний пристрій (АЛП)	246
10.2 Пристрій керування	249
10.3 Структура мікропроцесорної системи	257
10.4 Введення-виведення інформації	270
11. ЗАПАМ'ЯТОВУЮЧІ ПРИСТРОЇ.....	278
11.1. Запам'ятовуючі пристрої	278
11.3. Система керування пам'яттю в персональній ЕОМ.....	295
11.4. Захист пам'яті в мультипрограмах ЕОМ	300
12. ПРОЦЕСОРИ.....	309
12.1. Режими адресації й формати команд 16-розрядного процесора	309
12.2 Кодування команд.....	316
12.3 Архітектура процесора. Взаємодія основних вузлів і пристроїв персонального комп'ютера при автоматичному виконанні команди. Архітектура 32-розрядного мікропроцесора.	321
13. УНІВЕРСАЛЬНІ ПРОЦЕСОРИ	338
13.1. Регістрова структура універсального мікропроцесора	338
13.2. Основні напрямки розвитку архітектури універсальних мікропроцесорів.....	381
14. ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ	387
14.1. Паралельні структури обчислювальних систем.....	387
14.2 Мікропроцесорні системи й способи розпаралелювання	406
14.3. Розпаралелювання у ОР на рівні виконавчих пристроїв.....	422
14.4. Паралельна обробка стека й статичне розпаралелювання у вирішальному полі	434
14.5. Апаратна підтримка мови користувача – основна концепція мультипроцесорних систем.....	445
15. RISC-ПРОЦЕСОРИ	458
15.1. Мікропроцесори з RISC- архітектурою	458
15.2. Багатопроесорні й багатомашинні обчислювальні системи.....	468
15.3. Процесори цифрової обробки сигналів	482
Використана література.....	508

Наукове видання

Євсєєв С. П., Дженюк Н.В., Охрименко М.Ю.,
Головашич С.О., Кудій Д.А.

ЦИФРОВА СХЕМОТЕХНІКА ТА АРХІТЕКТУРА МІКРОПРОЦЕСОРІВ

Навчальний посібник

Відповідальний за випуск С.В. Піча
Дизайн та верстка К.А. Краснощок

Підписано до друку 26.07.2023 р.

Формат 60X84 1/16. Папір офсетний.
Цифровий друк. Ум. друк. арк. 32,0625.
Тираж 100.

Зам. № 2023 – 33.

Видавець ФОП Піча С.В.
а/с 5026, м. Львів-53, 79053, Україна
e-mail: novsv2016@ukr.net
068-978-94-42 / 050-33758-46

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру
видавців, виготівників і розповсюджувачів видавничої продукції:
серія ДК № 5069 від 22.03.2016 року, видане Державним комітетом
інформаційної політики, телебачення та радіомовлення України