3. ЦИФРОВІ ПРОЦЕСОРИ ОБРОБКИ СИГНАЛІВ – ТЕХНІЧНІ ЗАСОБИ КОМП'ЮТЕРІЗАЦІЇ СПЕЦІАЛІЗОВАНИХ СЕРЕДОВИЩ

3.1. Загальна характеристика цифрових процесорів обробки сигналів (ЦПОС) та область їх застосування

Цифрова обробка сигналу (ЦОС) – це арифметична обробка в реальному масштабі часу послідовності значень амплітуди сигналу, визначених через рівні часові проміжки. Прикладами цифрової обробки є:

- фільтрація сигналу;
- згортка двох сигналів (змішування сигналів);
- обчислення значень кореляційної функції двох сигналів;
- підсилення, обмеження або трансформація сигналу;
- пряме/зворотне Фур'є-перетворення сигналу.

Аналогова обробка сигналу, що традиційно використовується в багатьох радіотехнічних пристроях, є у багатьох випадках дешевшим способом досягнення необхідного результату. Проте тоді, коли потрібна висока точність обробки, мініатюрність пристрою, стабільність його характеристик в різних температурних умовах функціонування, цифрова обробка виявляється єдиним прийнятним рішенням [58].

Приклад аналогової фільтрації сигналу наведено на рис. 3.1.

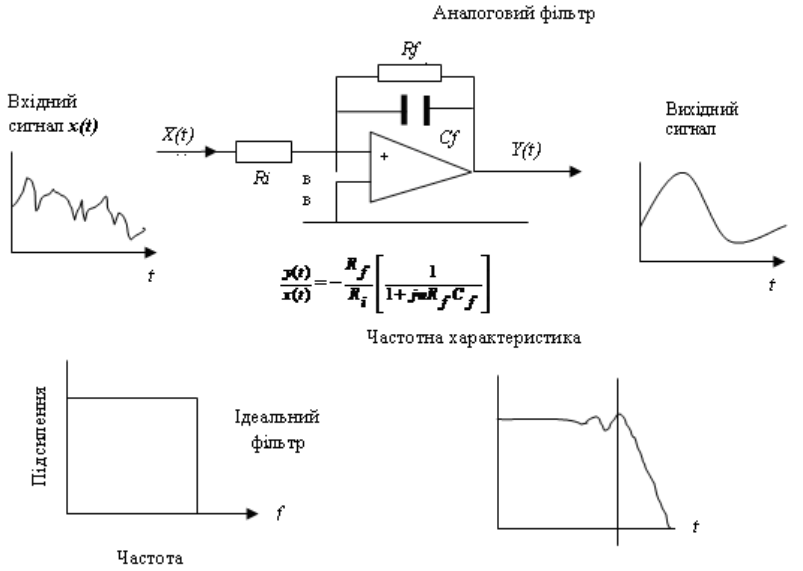

Рис. 3.1. Аналогова обробка сигналу

Використаний у фільтрі операційний підсилювач дозволяє розширити динамічний діапазон сигналів, що обробляються. Форма амплітудно-частотної характеристики фільтра визначається значеннями величин *Rf, Cf* . Для аналогового фільтра складно забезпечити високе значення добротності, характеристики фільтра сильно залежать від температурного режиму. Компоненти фільтра вносять додатковий шум в результуючий сигнал. Аналогові фільтри важко перебудовувати в широкому діапазоні частот.

Аналогічні результати обробки сигналу можуть бути отримані за допомогою цифрової схеми, показаної на рис. 3.2 [58].

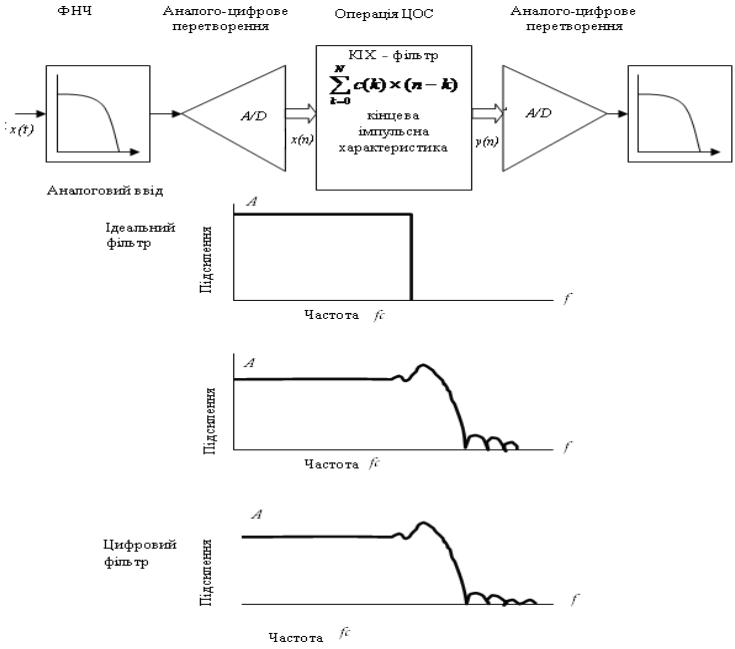

Рис. 3.2. Цифрова обробка сигналу

Компонентами схеми є фільтри низької частоти (ФНЧ), які виконують попереднє і подальше видалення з частотного спектру додаткових гармонік сигналу, аналого-цифровий (АЦП) та цифро-аналоговий (ЦАП) перетворювачі сигналу і власне цифровий фільтр з кінцевою імпульсною характеристикою. Амплітудно-частотна характеристика фільтра визначається значеннями коефіцієнтів фільтра С(к). Змінюючи кількість коефіцієнтів (довжину фільтра) і їх значення, можна отримати фільтр з будь-якою амплітудно-частотною характеристикою. Шуми, що вносяться (шуми

квантування), залежать від частоти і розрядності АЦП і ЦАП, а також від точності обчислень.

Схема перетворення, що виконується над послідовністю відліків сигналу, яка задається математичною формулою, може бути також зображена графічно у вигляді структурної схеми цифрового фільтра.

Існує класифікація фільтрів за виглядом імпульсної характеристики: фільтри з кінцевою імпульсною характеристикою КІХ (FIR) і фільтри з безкінечною імпульсною характеристикою БІХ (IIR).

Структура цифрових фільтрів типу FIR і IIR наведена на рис.3.3 і 3.4 відповідно. На цих рисунках прийняті такі позначення: T – блок затримки на 1 такт; X – блок множення; + – блок складання [58].

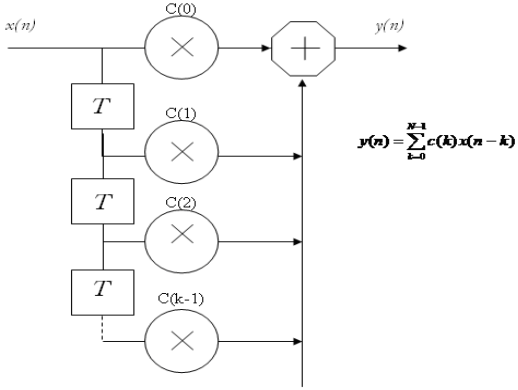

Рис. 3.3. Структура каскаду фільтра типу FIR

$$w(n) = x(n) - a_i1 \times w(n-1) - a_i2 \times w(n-2) \quad (3.1)$$

$$y(n) = w(n) + b_i1 \times w(n-1) + b_i2 \times w(n-2) \quad (3.2)$$

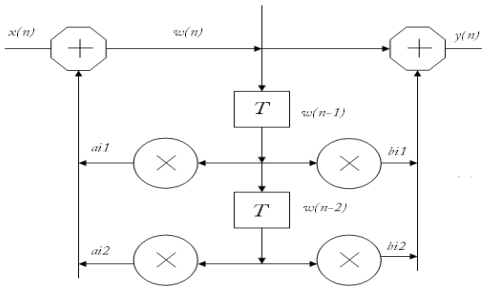

Рис. 3.4. Структура каскаду фільтра типу IIR

Для ефективної реалізації алгоритмів цифрової фільтрації необхідна апаратна підтримка базових операцій: множення з накопиченням (МАС); модульної адресної арифметики; нормування результатів арифметичних операцій.

Іншим перетворенням сигналу, часто виконуваним, є Фур'є-перетворення (пряме і зворотне).

Будь-який сигнал може бути зображений як в часовій області (сукупність графіків в координатах час-амплітуда), так і в частотній області (послідовність графіків в координатах частота-амплітуда). Залежно від складності реалізації обробки може бути вибране або частотне, або часове подання сигналу. Фур'є-перетворення дозволяє здійснювати перенесення сигналу з однієї форми подання в іншу.

Дискретне перетворення Фур'є (ДПФ) в аналітичному вигляді задається формулою [59]

$$X(f) \approx \widetilde{X}(f) = T \sum_{n=-\infty}^{+\infty} \chi(nT) e^{-j2\pi fnT}, \qquad (3.3)$$

де χ(nT) – послідовність відліків сигналу.

Існує велика різноманітність реалізацій дискретного перетворення Фур'є. У ряді алгоритмів використовуються прийоми, що дозволяють скоротити обсяг необхідних обчислень. Ці алгоритми відомі під загальною назвою «швидке перетворення Фур'є» (ШПФ).

На практиці інтервал підсумовування обмежений деяким числом часових відліків N, залежним від необхідної точності перетворення. В цьому випадку формула набуває вигляду [59]

$$X(f) \approx \widetilde{X}(f) = T \sum_{n=0}^{N-1} \chi(nT) e^{-j2\pi fnT}; \qquad (3.4)$$

N – називають числом точок перетворення.

Для зменшення числа операцій множення при виконанні ДПФ використовується метод, що отримав назву «проріджування за часом».

Суть даного методу полягає у тому, що перетворення Фур'є за послідовністю з N точок може бути виражене через перетворення, що виконані за підпослідовностями цієї послідовності, кожна з яких має довжину $N/2$ точок. Оскільки число множень пропорційне числу точок перетворення, то процедура двократного перетворення по $N/2$ точках (з подальшим об'єднанням результатів) уявляється вигіднішою з погляду часу обчислення. Застосувавши до послідовності відліків процедуру проріджування рекурсивно, отримаємо схему обчислень, зображену на рис. 3.5.

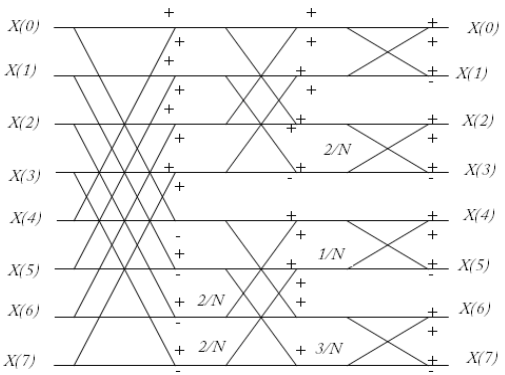

Рис.3.5. Схема ШПФ по 8 точках

На цьому рисунку рис *к/N* означає множення на коефіцієнт $e^{-jk\frac{2\pi}{N}}$.

При реалізації даної схеми перетворення разом з операціями множення і складання використовуються бітові операції. Як видно з рис 3.5, результуючі відліки розташовані в біт-реверсивному порядку, тобто такому, коли позиція елемента визначається реверсією двійкового подання індексу елемента. Для їх переупорядковування необхідно виконати або перестановку елементів масиву, або операцію бітової реверсії індексу при зверненні до елемента масиву. Другий підхід є більш економічним з погляду часу доступу, проте вимагає можливості маніпулювання адресами даних.

У більшості реальних додатків розглянуті базові алгоритми цифрової обробки сигналів повинні виконуватися в режимі реального часу, що ставить підвищені вимоги до продуктивності процесора, що їх реалізує. Апаратна підтримка базових операцій алгоритмів цифрової обробки сигналів є характерною особливістю сигнальних процесорів. Нижче будуть розглянуті основні сімейства сигнальних мікропроцесорів, представлені на світовому ринку.

3.1.1. Організація обчислень в ЦПОС. Особливості архітектури

Процесори ЦПОС можна для зручності розділити на дві категорії: універсальні і спеціалізовані [58].

Існує два типи спеціалізованих пристроїв.

1. Апаратне забезпечення, розроблене для ефективного виконання спеціальних алгоритмів ЦОС, таких, як цифрова фільтрація, швидке перетворення Фур'є. Пристрої даного типу іноді називають алгоритмічними процесорами ЦОС.

2. Апаратне забезпечення, розроблене для спеціального додатку, наприклад, у сфері контролю, телекомунікацій або цифрового аудіо. Пристрої даного типу іноді називають процесорами ЦОС спеціального призначення (спеціалізованими) .

В більшості випадків спеціалізовані процесори виконують такі алгоритми ЦОС, як кодування-декодування. Крім того, вони повинні виконувати інші операції, що відображають специфіку додатку.

Всі універсальні і спеціалізовані процесори можна побудувати за допомогою окремих мікросхем або блоків помножувачів, арифметико-логічних пристроїв (АЛП), елементів пам'яті та ін.

Розглянемо архітектурні особливості процесорів ЦОС, які дозволили застосувати цифрову обробку у реальному часі в багатьох областях.

Більшість доступних зараз універсальних процесорів заснована на архітектурі фон Неймана, при якій операції виконуються послідовно. При обробці інструкції в такому процесорі блоки процесора, не задіяні в кожній фазі інструкції, перебувають у стані очікування до передачі їм управління. Підвищення швидкості процесора досягається за рахунок прискорення роботи окремих блоків.

Якщо пристрій має працювати у реальному часі, то архітектуру процесора ЦОС потрібно оптимізувати під виконання функцій ЦОС. Наприклад, на рис. 3.6 показана загальна апаратна архітектура цифрової обробки сигналів у реальному часі. Вона характеризується такими особливостями:

• містить багатошинну структуру з роздільною пам'яттю для даних і інструкцій програми. Звичайно пам'ять для зберігання даних містить вхідні дані, проміжні значення і вихідні вибірки, а також фіксовані коефіцієнти, наприклад, для цифрової фільтрації або ШПФ. Команди програми зберігаються у спеціально відведених елементах пам'яті;

• порт вводу-виводу дозволяє обмінюватися даними із зовнішніми пристроями (АЦП, ЦАП) або передавати цифрові дані іншим процесорам. Прямий доступ до пам'яті (Direct Memory Access — DMA), якщо він є, дозволяє швидко обмінюватися блоками даних з пам'яттю (ОЗП) для зберігання даних, причому звичайно це відбувається під зовнішнім управлінням;

• містить арифметичні пристрої для логічних і арифметичних операцій, до числа яких виходять АЛП, апаратні помножувачі і схеми зсуву (або помножувачі-накопичувачі) [58].

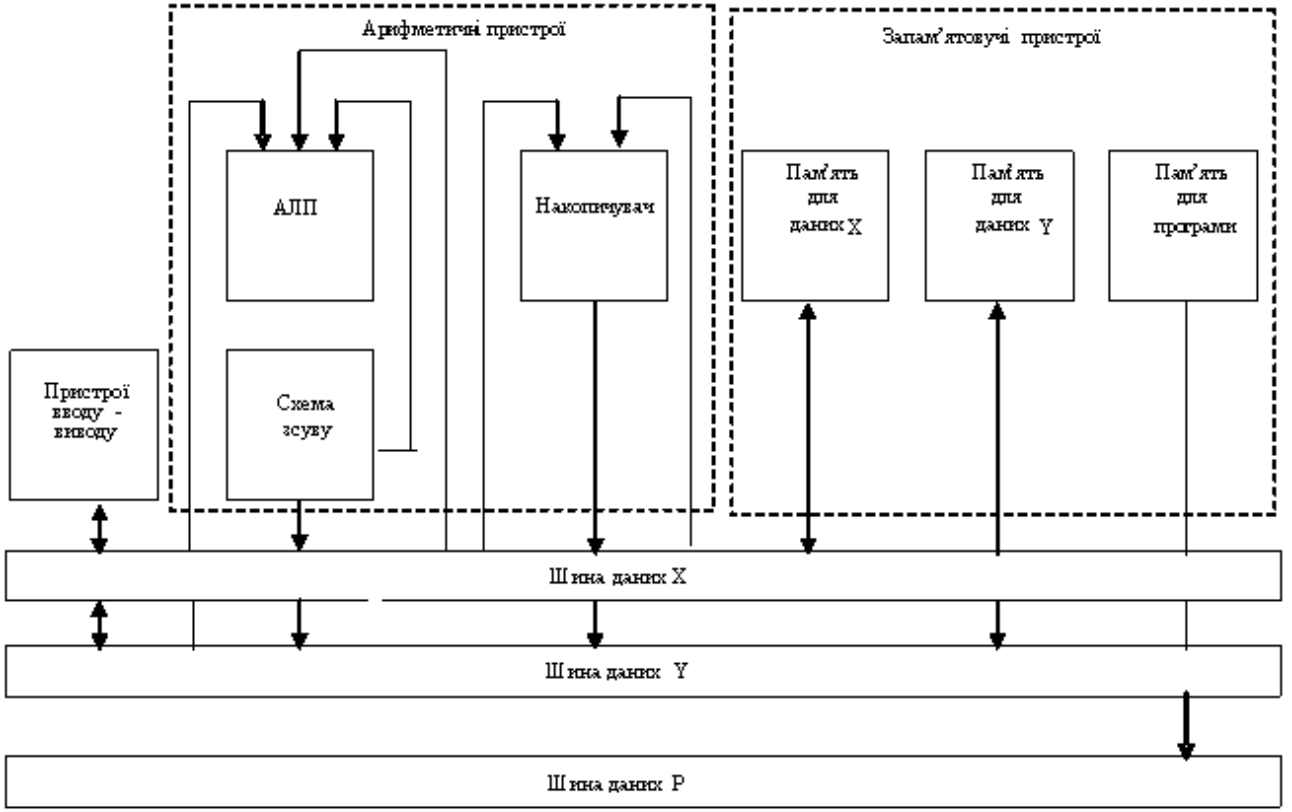

Рис. 3.6. Стандартна універсальна апаратна архітектура обробки сигналів

Для цифрової обробки сигналів використовуються так звані сигнальні мікропроцесори. До їх особливостей можна віднести малорозрядну (40 розрядів і менш) обробку чисел з плаваючою точкою, переважне використання чисел з фіксованою точкою розрядності 32 і менш, а також орієнтацію на нескладну обробку великих масивів даних.

Відмітною особливістю задач цифрової обробки сигналів є потоковий характер обробки великих обсягів даних в реальному режимі часу, який вимагає від технічних засобів високої продуктивності і забезпечення можливості інтенсивного обміну із зовнішніми пристроями. Відповідність до даних вимог досягається завдяки специфічній архітектурі сигнальних процесорів і проблемно-орієнтованій системі команд.

Сигнальні процесори мають високий ступінь спеціалізації. У них широко використовуються методи скорочення тривалості командного циклу, характерні і для універсальних RISC-процесорів, такі, як конвеєризація на рівні окремих мікроінструкцій і інструкцій, розміщення операндів більшості команд в регістрах, використання тіньових регістрів для збереження стану обчислень при перемиканні контексту, розділення шин команд і даних (гарвардська архітектура). У той же час для сигнальних процесорів характерною є наявність апаратного помножувача, що дозволяє виконувати множення двох чисел за один командний такт. В універсальних процесорах множення звичайно реалізується за декілька тактів, як послідовність операцій зсуву і складання. Іншою особливістю сигнальних процесорів є включення до системи команд таких операцій, як множення з накопиченням MAC (C:=AxB+C з вказаним в команді числом виконань в циклі і з правилом зміни індексів елементів масивів A і B), інверсія біт адреси, різноманітні бітові операції. У сигнальних процесорах реалізується апаратна підтримка програмних циклів, кільцевих буферів. Один або декілька операндів витягуються з пам'яті в циклі виконання команди.

Реалізація однотактного множення і команд, що використовують як операнди вміст елементів пам'яті, обумовлює порівняно низькі тактові частоти роботи цих процесорів. Спеціалізація не дозволяє підіймати продуктивність за рахунок швидкого виконання коротких команд типу «регістр-регістр», як це робиться в універсальних процесорах. Цих команд просто немає в програмах обробки сигналів.

3.1.2 MAC-операція. Схеми реалізації в ЦПОС. Точність обчислень

Основними чисельними операціями в цифровій обробці сигналів є множення і складання. Множення в програмній формі сумно відоме своєю трудомісткістю, а якщо використовується арифметика з плаваючою точкою, складання виявляється навіть ще більш трудомісткою операцією. Щоб

максимально прискорити цифрову обробку сигналів у реальному часі, рекомендується використовувати спеціалізовані апаратні помножувачі-накопичувачі для арифметики з плаваючою або фіксованою точкою. Таке апаратне забезпечення тепер стандартно використовується у всіх цифрових процесорах сигналів. У процесорі з фіксованою точкою апаратний помножувач за один такт (звичайно 25 нс) приймає два 16-бітові дробові числа, подані у формі доповнення до двох, і обчислює їх 32-бітовий добуток. Середній час виконання команди множення-накопичення можна значно зменшити, якщо використовувати спеціальні команди повторення.

Типова конфігурація апаратного помножувача-накопичувача ЦОС зображена на рис. 3.7. У такій конфігурації помножувач має пару вхідних регістрів (Рг X та Рг Y), які містять входи помножувача, і 32-бітовий регістр добутку (Рг P), який містить результат множення. Вихід регістра P (product - добуток) з'єднується з накопичувачем подвійної точності, в якому накопичуються добутки.

Подібна конфігурація застосовується в апаратних помножувачах-накопичувачах з плаваючою точкою, за винятком того, що входи і добутки – це нормовані числа, що подані у формі з плаваючою точкою. Помножувачі-накопичувачі з плаваючою точкою дозволяють швидко обчислювати результати алгоритмів ЦОС з мінімальним помилками [58].

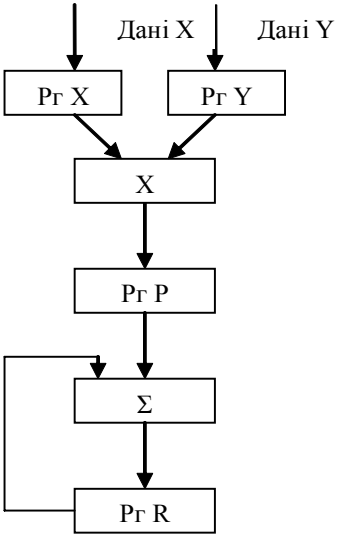

Рис. 3.7. Типова конфігурація апаратного помножувача-накопичувача ЦОС

Операції з плаваючою точкою дозволяють використовувати великий динамічний діапазон і знижують арифметичні помилки, хоча в багатьох додатках достатньо динамічного діапазону, який пропонує подання з фіксованою точкою.

Двома найпоширенішими типами арифметики, що використовуються в сучасних ЦПОС, є арифметики з фіксованою і плаваючою точками. Арифметика з плаваючою точкою – це природний вибір в додатках з широкими і змінними вимогами до динамічного діапазону (динамічний діапазон можна визначити як різницю між найбільшим та найменшим рівнем сигналу, який можна представити, або як різницю між найбільшим сигналом і мінімальним рівнем шуму, що вимірюються в децибелах). Процесори з фіксованою точкою переважні з погляду низької вартості, вони підходять для масового виробництва (наприклад, стільникові телефони і комп'ютерні дисководи). При використанні арифметики з фіксованою точкою виникають питання, пов'язані з обмеженнями динамічного діапазону. Взагалі процесори з плаваючою точкою дорожчі за процесори з фіксованою точкою, хоча останніми роками різниця в ціні істотно зменшилася. Більшість існуючих ЦПОС з плаваючою точкою також підтримує арифметику з фіксованою точкою.

Використання даних у форматі з плаваючою точкою в сигнальній обробці обумовлене декількома причинами. Для багатьох задач, пов'язаних з виконанням інтегральних і диференціальних перетворень, особливе значення має точність обчислень, забезпечити яку дозволяє експоненціальний формат подання даних. Алгоритми компресії, декомпресії, адаптивної фільтрації в ЦОС пов'язані з визначенням логарифмічних залежностей і досить чутливі до точності подання даних в широкому динамічному діапазоні.

Робота з даними у форматі з плаваючою точкою істотно спрощує і прискорює обробку, підвищує надійність програми, оскільки не вимагає виконання операцій округлення і нормалізації даних, відстежування ситуацій втрати значущості і переповнювання.

3.1.3. Фізична організація пам'яті в ЦПОС. Гарвардська архітектура та багатопортова пам'ять

Фізично пристрої пам’яті в ЦПОС можна розподілити на внутрішні та зовнішні. До внутрішніх належать:

• оперативний запам’ятовуючий пристрій (ОЗП), присутній у всіх типах ЦПОС;

• постійний запам’ятовуючий пристрій (ПЗП), присутній у деяких типах ЦПОС;

• напівпостійний запам’ятовуючий пристрій (НПЗП), присутній у деяких типах ЦПОС.

Ємність ОЗП, ПЗП та НПЗП залежить від типу ЦПОС і буде розглянута нижче для кожного типу процесора окремо.

Зовнішню пам'ять можна розподілити:

• на пам'ять для збереження даних;

• пам'ять для збереження програм.

Принципова особливість гарвардської (двошинної) архітектури полягає у тому, що пам'ять для зберігання даних і пам'ять для зберігання програми розташовуються в різних місцях, припускаючи повне поєднання в часі операцій виклику команди з пам'яті і її виконання [58]. Стандартна гарвардська архітектура наведена на рис. 3.8. Звичайні мікропроцесори, такі, як Intel 6502, характеризуються одношинною структурою, через яку передаються і дані, і команди. У гарвардській архітектурі, де команди програми і дані зберігаються в різних областях пам'яті, виклик наступної команди може співпадати з виконанням поточної команди. Як правило пам'ять програми містить програмний код, тоді як пам'ять для зберігання даних включає змінні, наприклад, вибірки вхідних даних.

Стандартна гарвардська архітектура використовується лише в декількох процесорах ЦОС (наприклад, Motorola DSP56000), в більшості пристроїв застосовується модифікована гарвардська архітектура (наприклад, сімейство процесорів TMS320). У модифікованій архітектурі також виділяються роздільні області пам'яті для зберігання програми і даних, але на відміну від строгої гарвардської архітектури тут дозволений зв'язок між двома областями пам'яті.

Для взаємодії із зовнішніми пристроями (АЦП, ЦАП та ін.) або з іншими процесорами в ЦПОС передбачена система портів вводу-виводу. За способом передачі даних (послідовний чи паралельний) та за розрядністю номенклатура портів вводу-виводу залежить від конкретного типу ЦПОС.

3.1.4. Логічна організація пам’яті та режими адресації

Комбінація гарвардської архітектури з швидкодіючою та багатопортовою пам’яттю дозволяє реалізувати вимоги до ефективної організації взаємодії з пам’яттю, якої потребують алгоритми ЦОС.

Деякі ЦПОС мають спеціальний кеш команд, який є пам’яттю невеликого розміру, що входить до складу ядра процесора та призначається для збереження команд. Попереднє завантаження команд до кеша дозволяє звільнити шини процесора для доступу до даних. Найпростіші процесори мають кеш на одну команду. В більш складних ЦПОС ємність кеша складає 1К слів.

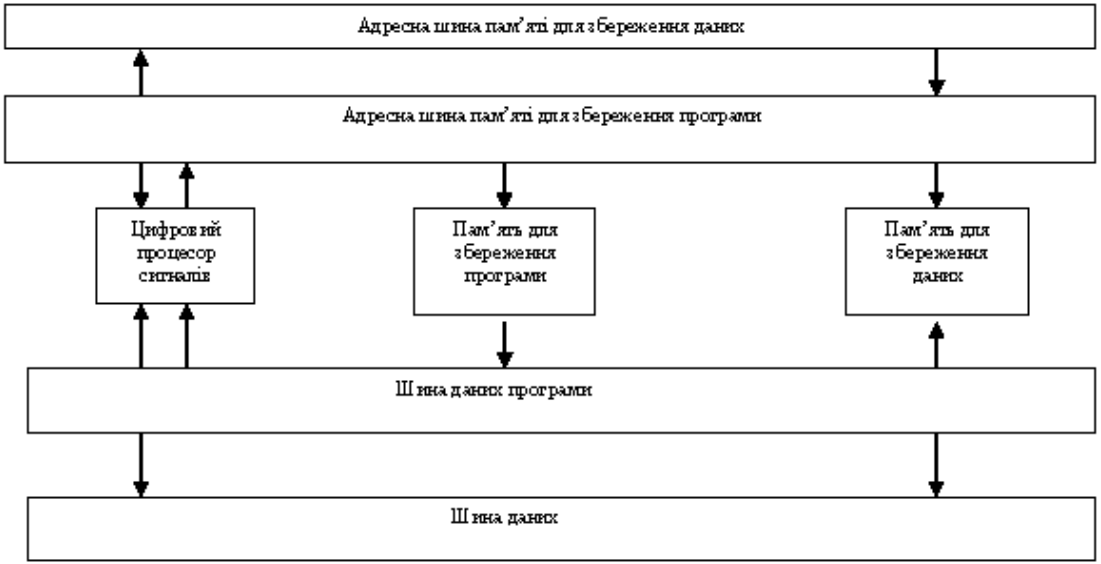

Рис. 3.8. Стандартна гарвардська архітектура

При виконанні будь-якої команди процесор звертається до пам'яті програми або до пам'яті даних. Спосіб визначення адреси операнда в команді або адреси передачі управління називається режимом адресації. У ЦПОС використовують режими таких видів адресації: прямої, безпосередньої та різних видів непрямої адресації [60].

Пряма адресація. Абсолютна адреса операнда міститься в кодовому слові команди. Наприклад, команда ЦПОС ADSP-21xx

AX0=DM(0800)

завантажує значення, розташоване за абсолютною адресою 0800 пам'яті даних до регістру AX0.

Безпосередня адресація. Значення операнда міститься в команді. Наприклад, команда ЦПОС ADSP-21xx

AX0=1357

Значення константи 1357 завантажується до регістру AX0. Якщо значення константи перевищує довжину одного кодового слова команди, воно розміщується в наступному кодовому слові. Це призводить до читання з пам'яті двох слів, що підвищує час виконання програми.

Непряма адресація. Використовує регістри-покажчики, вміст яких є реальною адресою розміщення даних у пам'яті. У цьому випадку в команді міститься тільки посилання на регістр-покажчик. Кількість регістрів для непрямої адресації в ЦПОС невелика, тому довжина команди непрямої адресації дорівнює одному слову. У ЦПОС використовуються такі види непрямої адресації: непряма регістрова адресація, непряма регістрова адресація з інкрементуванням, непряма регістрова адресація з модульною арифметикою, біт-інверсна адресація. Проілюструємо ці види адресації за допомогою операторів мови С++.

Непряма регістрова адресація. Відповідає наступній формі оператора присвоєння:

A=A+*R;

Відповідно до цього оператора значення, що зберігається за адресою пам'яті, яка міститься в регістрі R, додається до значення акумулятора A. Регістр R розглядається як покажчик.

Непряма регістрова адресація з інкрементуванням та декрементуванням. Відповідає наступним операторам:

A=A+(*R)++;

A=A+(*R)--;

Значення, що зберігається за адресою пам'яті, яка міститься в регістрі R, додається до значення акумулятора A. Після отримання значення з пам'яті вміст регістру-покажчика збільшується (++) або зменшується (--) на одиницю. Деякі ЦПОС дозволяють збільшувати або зменшувати вміст

регістра-покажчика не на одиницю, а на іншу величину, значення якої зберігається в додатковому регістрі.

Непряма регістрова адресація з модульною арифметикою. Використовується при організації в пам'яті кільцевого буфера. Буфер – послідовність однотипних елементів даних, що розташовані у суміжних комірках пам'яті. Якщо елементи даних обробляються в порядку їх запису у пам'ять, то буфер відповідає структурі даних, яка називається чергою. Для доступу до значень, що зберігаються в буфері, використовуються покажчики. Кожен раз після звертання до буфера значення покажчика змінюється з визначеним кроком. При досягненні кінця буфера значення покажчика змінюється на початкове. Цей механізм можна описати за допомогою умовного оператора

```
if(pointer+step<BkEnd);
  pointer=pointer+step;
else pointer=BkBegin;
```

Змінна BkBegin визначає початкову адресу, а BkEnd – кінцеву адресу області пам'яті, в якій розміщено буфер. Змінна pointer – покажчик, а змінна step відповідає кроку, з яким змінюється значення покажчика. Ця схема адресації використовується, наприклад, в ЦПОС TMS320C5x.

Можливе використання іншого механізму адресації кільцевого буфера, який застосовується в ЦПОС ADSP21xx, ADSP21xxx, TMS320C3x, TMS320C4x. У цих ЦПОС не задається кінцева адреса області пам'яті, відведеної під буфер, а в спеціальному регістрі фіксується довжина буфера. Значення, що відповідає довжині буфера, називається модулем. При збільшенні початкової адреси на величину модуля здійснюється перехід до початкової адреси, що і складає суть непрямої адресації з модульною арифметикою.

Біт-інверсна адресація. Дозволяє змінювати впорядкованість вхідних або вихідних даних, що потрібно, наприклад, в алгоритмах ШПФ. Біт-інверсний порядок задається шляхом «дзеркального» відображення двійкових розрядів вхідної чи вихідної послідовності. Приклад реалізації наведено в табл.. 3.1 [60].

Таблиця 3.1 – Біт-інверсний порядок

N	Двійковий номер	Біт-інверсія	Біт-інверсний номер
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

3.1.5. Архітектура й призначення спеціалізованих модулів ЦПОС: таймера, лічильника числа повторень, генераторів адреси

Призначення та роботу спеціалізованих модулів розглянемо на прикладі ЦПОС сімейства TMS320C2x [60].

У складі ЦПОС є 16-розрядний таймер (TIM) та регістр періоду таймера (PRD). Таймер управляється відповідним сигналом (CLKOUT1). При скиданні таймера в нього та регістр періоду передається максимальне число FFFFh. Після дії сигналу скидання вміст таймера зменшується на одиницю при кожному надходженні сигналу CLKOUT1. Початкове значення таймера визначається вмістом регістру періоду. Як тільки значення в таймері стане дорівнювати нулю, формується переривання від таймера (TINT) і він перезавантажує те значення, що знаходиться в регістрі періоду. Це відбувається під час першого машинного циклу після формування переривання TINT. Таким чином, переривання будуть відбуватися через проміжки часу, що дорівнюють $[(PRD)+1]*T_{CLKOUT1}$, де $T_{CLKOUT1}$ – період сигналу CLKOUT1.

Зазвичай переривання TINT використовується для зчитування вибірок вхідних даних, що обробляються процесором. Таймер та регістр періоду можуть бути проініціалізовані в будь-якому машинному циклі. Нульове значення регістру періоду не допускається.

Якщо переривання TINT не використовуються, то необхідно застосувати маску або заборонити переривання командою DINT. У цьому випадку регістр періоду може використовуватися як звичайна комірка пам'яті. Якщо знову виникне потреба у використанні переривання TINT, то регістр PRD та таймер необхідно попередньо ініціалізувати, а потім дозволити переривання TINT.

Лічильник числа повторень RPTC містить 8-розрядне число N, яке визначає повтори деякої команди. Кількість повторів дорівнює N+1. За допомогою команди RPT або RPTK лічильник числа повторень може бути завантажений значенням з діапазону від 0 до 255. Це дозволяє повторювати команду, що знаходиться за RPT або RPTK до 256 разів.

Лічильник числа повторень може використовуватися з командами: множення з накопиченням, пересилання блоків, вводу-виводу, читання-запису таблиць. Ті команди, що потребують для виконання декількох циклів, при запису їх після команди RPT або RPTK виконуються за один цикл.

Для формування адрес при звертанні до пам'яті в ЦПОС застосовуються генератори адреси, які можуть формувати одну чи декілька адрес даних за один цикл команди, не використовуючи для цього основного арифметичного пристрою, що займається обробкою даних. Це дозволяє

обчислювати необхідні адреси даних паралельно з виконанням арифметичних операцій, що підвищує продуктивність ЦПОС

3.1.6. Архітектурні особливості основних сімейств ЦПОС (Motorola, Texas Instruments, Analog Devices)

У числі найпоширеніших сигнальних процесорів можна відзначити вироби таких компаній: Motorola (56002,96002), Intel (i960), Texas Instruments (TM5320Cxx), Analog Devices (21xx, 210xx). Велика продуктивність, що вимагається при обробці сигналів у реальному часі, спонукала дві останні з перелічених компаній випустити трансп'ютерні сімейства мікропроцесорів TMS320C4x і ADSP2106x, що орієнтовані на використання в мультипроцесорних системах [60].

Вибір того або іншого процесора для реалізації конкретного проекту – багатокритеріальна задача, але слід зазначити перевагу процесорів компанії Analog Devices для додатків, що вимагають виконання великих обсягів математичних обчислень (таких, як цифрова фільтрація сигналу, обчислення кореляційних функцій і т.ін.), оскільки їх продуктивність на подібних задачах вища, ніж у процесорів компаній Motorola і Texas Instruments. У той же час для задач, що вимагають виконання інтенсивного обміну із зовнішніми пристроями (багатопроцесорні системи, різного роду контроллери), перевагу можна віддати використанню мікропроцесорів компанії Texas Instruments, які обладнані високошвидкісними інтерфейсними підсистемами.

3.1.7. Інструментальні засоби розробки систем на основі ЦПОС

У число специфічних чинників, які слід розглянути при виборі процесора ЦОС для даного додатку, входять архітектурні особливості, швидкість виконання, тип арифметики і довжина слова [58].

Архітектурні особливості. Більшість доступних зараз ЦПОС має хорошу архітектуру. Ключовими характеристиками процесорів є розмір вбудованої пам'яті, наявність спеціальних команд і можливості вводу-виводу. Наявність вбудованої пам'яті – необхідна вимога в більшості додатків ЦОС реального часу, оскільки це означає швидкий доступ до даних і швидке виконання програми. Для додатків із суворими вимогами до пам'яті (цифрове аудіо, система Dolby, факс-модем, кодування-декодування MPEG) розмір внутрішньої пам'яті ОЗП може стати вирішальним чинником при виборі процесора. Якщо внутрішньої пам'яті недостатньо, її можна доповнити високошвидкісною зовнішньою пам'яттю, хоча це може призвести до збільшення вартості системи. Для додатків, що вимагають швидких і ефективних обчислень або обміну потоками даних із зовнішнім світом,

досить важливі такі засоби вводу-виводу, як інтерфейси АЦП і ЦАП, можливість прямого доступу до пам'яті і підтримка багатопроцесорної обробки. Залежно від додатку важливий багатий набір спеціальних команд підтримки операцій ЦОС, наприклад, можливість організації циклів з нульовими службовими витратами, спеціалізовані команди ЦОС і колова адресація.

Швидкість виконання. Оскільки більшість задач ЦОС вимагає термінового розв'язання, то важливою мірою продуктивності є швидкість процесорів ЦОС. Традиційно двома основними одиницями вимірювання цієї величини є тактова частота процесора в мегагерцах або гігагерцах і число команд, що виконуються, в мільйонах команд за секунду (Million Instructions Per Second - MIPS) або, якщо використовуються процесори ЦОС з плаваючою точкою, в мільйонах операцій з плаваючою точкою за секунду (million floating-point operations per second - MFLOPS). Втім подібні методи вимірювання можуть в деяких випадках не надавати об'єктивної картини через значні відмінності в принципах роботи різних ЦПОС, більшість з яких може виконувати декілька операцій в одній машинній команді. Різні процесори також відрізняються числом операцій, що виконуються в кожному такті. Отже, порівнювати швидкості роботи процесорів на основі зазначених вище методів не коректно. Альтернативна міра порівняння заснована на швидкості виконання контрольних алгоритмів, наприклад основних алгоритмів ЦОС, таких, як ШПФ, КІХ- та БІХ-фільтрація. Вивчаючи та порівнюючи чи ці дані, можна отримати уявлення про відносну продуктивність різних популярних ЦПОС.

Тип арифметики. Критерії вибору арифметики з фіксованою або з плаваючою точками розглядалися вище.

Довжина слова. Ще одним важливим параметром в ЦОС є довжина слова даних процесора, оскільки вона може істотно впливати на якість сигналу. Цей параметр визначає, наскільки точно можна представити параметри і результати операцій ЦОС. Взагалі чим довше слово даних, тим менша помилка при цифровій обробці сигналу. У аудіообробці з фіксованою точкою, наприклад, для підтримки CD-якості довжина слова процесора повинна бути не менше 24 біт, що дозволить підтримати найменший рівень сигналу, достатньо вищий за мінімальний рівень шуму, що генерується обробкою сигналу. У процесорах ЦОС з фіксованою точкою використовуються різноманітні довжини слів процесорів, залежно від додатку. У процесорах ЦОС з фіксованою точкою, націлених на ринок телекомунікацій, звичайно використовуються слова 16-бітової довжини (наприклад, TMS320C54x), тоді як у процесорах, націлених на аудіододатки високої якості, використовуються слова довжиною 24 біта (наприклад, DSP56300). Останніми роками автори відзначають тенденцію до

використання більшого числа бітів для АЦП і ЦАП (наприклад, 24-бітовий аудіокодек Cirrus, CS4228), оскільки вартість подібних пристроїв постійно знижується. Таким чином, автори передбачають підвищення попиту на процесори для аудіообробки з великими довжинами слів. У накопичувачах процесорів з фіксованою точкою також можуть бути потрібні захисні біти (звичайно 1 – 8 біт) для запобігання арифметичному переповненню в процесі операцій множення і накопичення підвищеної точності. Додаткові біти ефективно розширюють динамічний діапазон, доступний для процесорів ЦОС. У більшості процесорів ЦПОС під арифметикою звичної точності розуміють використання 32-бітових даних (24-бітова мантиса і 8-бітова експонента). Більшість процесорів ЦОС з плаваючою точкою також дозволяє виконувати операції з фіксованою точкою і часто підтримує арифметику з фіксованою точкою із змінним розміром даних.

На практиці при виборі процесора можуть враховуватися і такі чинники, як досвід роботи з конкретним сімейством процесорів ЦОС, легкість використання, термін присутності на ринку і вартість.

Контрольні питання

1. Наведіть приклади найбільш поширених методів ЦОС.
2. Які особливості апаратної архітектури ЦОС?
3. Від яких чинників залежить точність обчислень ЦОС?
4. З яких основних структурних елементів складається ЦПОС?
5. Які типи арифметики застосовуються в ЦПОС? У чому їх особливості?
6. Як апаратно реалізується операція множення з накопиченням (MAC)?
7. Що таке гарвардська архітектура?
8. Які основні принципи вибору ЦПОС для розробки конкретного додатку?
9. ЦПОС яких фірм набули найбільшого поширення?
10. За якими критеріями порівнюються ЦПОС щодо швидкодії?

3.2. Архітектура й особливості функціонування ЦПОС із фіксованою точкою

3.2.1. ЦПОС із фіксованою точкою фірми Analog Devices (ADSP 21xx)

Мікропроцесори компанії Analog Divices представлені двома сімействами: ADSP21xx і ADSP210xx [60].

Сімейство ADSP21xx – набір однокристальних 16-розрядних мікропроцесорів із загальною базовою архітектурою, що оптимізована для виконання алгоритмів цифрової обробки сигналів та інших додатків, що вимагають високошвидкісних обчислень з фіксованою точкою. Представники сімейства відрізняються один від одного, в основному, розташованими на кристалі периферійними пристроями, такими як кеш-пам'ять, таймери, порти і т. ін.

Інше сімейство мікропроцесорів ADSP210xx, яке об'єднує 32-розрядні мікропроцесори, орієнтовані на сигнальні алгоритми, що вимагають виконання обчислень з плаваючою точкою, буде розглянуте у відповідному розділі.

Мікропроцесори сімейства ADSP21xx успішно конкурують з аналогічною продукцією інших компаній-виробників сигнальних процесорів завдяки порівнянній продуктивності при нижчій ціні, а також розвиненій системі технічних і програмних засобів розробки прикладних систем.

Висока продуктивність процесорів на сигнальних алгоритмах досягається завдяки багатофункціональній і гнучкій системі команд, апаратній реалізації більшості типових для даних додатків операцій, високому ступеню паралелізму процесів в мікропроцесорі, скороченню командного такту. Мікропроцесори ADSP21xx мають модифіковану Гарвардську архітектуру, в рамках якої передбачається можливість доступу в пам'ять команд, при її фізичному розділенні з пам'яттю даних. (Аналогічну архітектуру, що стала для DSP-процесорів стандартом де-факто, мають багато інших процесорів, наприклад TMS320xxx виробництва кампанії Texas Instruments).

Узагальнена структура мікропроцесора ADSP21xx наведена на рис. 3.9.

Кожен мікропроцесор цього сімейства містить три незалежні повнофункціональні обчислювальні пристрої: АЛП, MAC-помножувач з накопиченням, пристрій барабанного зсуву. Кожен пристрій безпосередньо оперує з 16-розрядними даними і забезпечує апаратну підтримку обчислень з різною точністю.

Мікропроцесор містить генератор адрес команд (PS) і два генератори

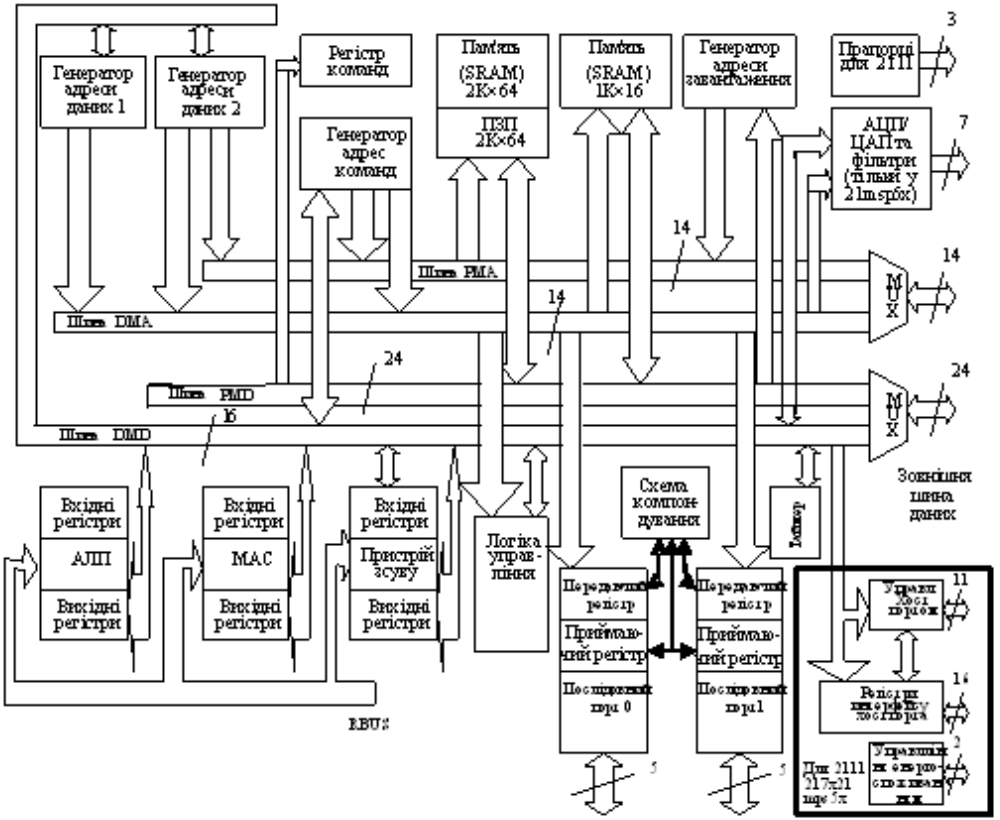

Рис. 3.9. Узагальнена структура мікропроцесора ADSP21xx

адрес даних (DAG), що забезпечують адресацію до даних і команд, розташованих як у внутрішній, так і в зовнішній пам'яті. Паралельне функціонування генераторів скорочує тривалість виконання команди, дозволяючи за один такт вибирати з пам'яті команду і два операнди.

Таймер/лічильник мікропроцесора забезпечує періодичну генерацію переривань.

Послідовні порти (SPORTs) забезпечують послідовний інтерфейс з більшістю стандартних послідовних пристроїв, а також з апаратними засобами стиснення-відновлення даних, що використовують А- і μ-закони компандування.

Порт інтерфейсу з хост-процесором (HIP) дозволяє без додаткових інтерфейсних схем взаємодіяти з головним процесором системи, в якості якого може використовуватися як процесор даного сімейства, так і інший мікропроцесор, наприклад Motorola 68000 або Intel 8051.

Мікропроцесор ADSP-2181 містить внутрішній порт ПДП (IDMA) і байтовий порт ПДП (BDMA), що забезпечує швидкий обмін з внутрішньою пам'яттю. Порт IDMA підтримує асинхронний обмін з пам'яттю програм, а порт BDMA дозволяє записувати і читати як команди, так і дані.

Мікропроцесори компанії Analog Devices відрізняє високий ступінь паралелізму внутрішніх операцій. За один такт процесор може:

- генерувати адресу наступної команди;
- завантажити з пам'яті наступну команду;
- виконати 1 або 2 пересилання даних;
- відновити 1 або 2 покажчика на дані;
- виконати операцію.

Мікропроцесор, що має відповідний пристрій, може в цьому ж такті:

- прийняти та/або передати дані через послідовні порти;
- прийняти та/або передати дані хост-процесору;
- прийняти та/або передати дані через аналоговий інтерфейс.

Основні характеристики мікропроцесорів сімейства ADSP-21xx зведені в табл.. 3.2 [60].

Загальне для сімейства ADSP-21xx мікропроцесорне ядро зображене на рис. 3.10. Арифметико-логічний пристрій мікропроцесора виконує стандартний набір арифметичних і логічних операцій, включаючи розподіл. Пристрій MAC виконує операції множення із складанням (відніманням) за один такт. Пристрій зсуву здійснює арифметичні і логічні зсуви операндів, нормалізацію і експоненціальні операції. Функціональні пристрої мікропроцесора можуть обмінюватися результатами виконання операцій по шині результатів (R).

Внутрішні функціональні модулі зв'язані між собою за допомогою п'яти шин: шини адрес пам'яті даних (DMA), шини адрес пам'яті команд

Таблиця 3.2 – Основні характеристики мікропроцесорів сімейства 21xx

Можливості	2101	2103	2105	2115	2111	2171	2173	2181	2183	21msp58
АЛП	+	+	+	+	+	+	+	+	+	+
Блок МАС	+	+	+	+	+	+	+	+	+	+
Зсув	+	+	+	+	+	+	+	+	+	+
Генератор адрес даних	+	+	+	+	+	+	+	+	+	+
Генератор адрес команд	+	+	+	+	+	+	+	+	+	+
ОЗП даних	1К	1К	512	512	1К	2К	2К	16К	16К	2К
ОЗП команд	2К	2К	1К	1К	2К	2К	2К	16К	16К	2К
Таймер	+	+	+	+	+	+	+	+	+	+
Багатоканальний послідовний порт	+	+		+	+	+	+	+	+	+
Послідовний порт 1	+	+	+	+	+	+	+	+	+	+
Порт хост-інтерфейсу	-	-	-	-	+	+	+	-	-	+
Порт ПДП	-	-	-	-	-	-	-	+	+	-
Аналоговий інтерфейс	-	-	-	-	-	-	-	-	-	+
Напруга живлення, В	5	3.3	5	5	5	5	3.3	5	3.3	5
Продуктивність (MIPS)	20	10	13.8	20	20	33	20	33	33	26

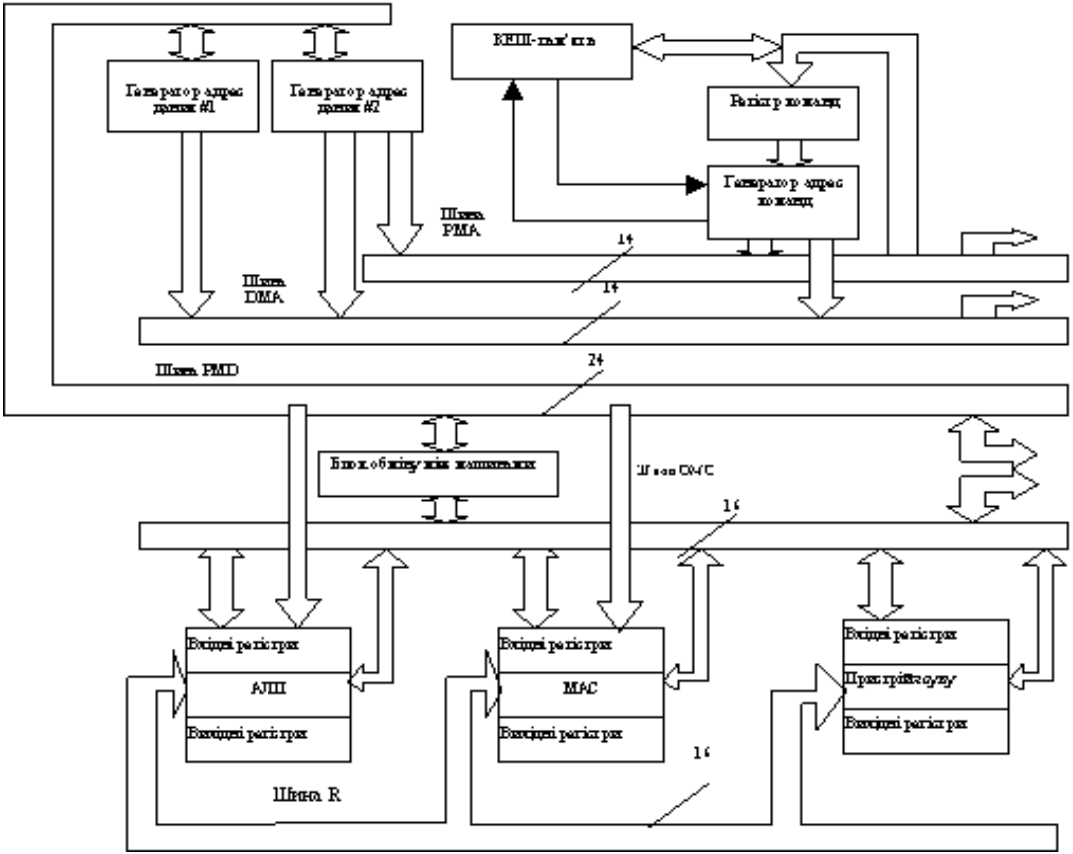

Рис 3.10. Структура мікропроцесорного ядра ADSP 21xx

(PMA), шини даних пам'яті даних (DMD), шини даних пам'яті команд (PMD) і шини внутрішніх результатів (R). Перші чотири шини мають мультиплексований зовнішній інтерфейс у вигляді шини адреси і шини даних (див. рис 3.10).

Система команд мікропроцесорів цього сімейства оптимізована для алгоритмів цифрової обробки сигналів. За системою команд всі мікропроцесори сумісні від верху до низу. Окремі представники сімейства – ADSP-2171, 2181, 21msp5x – мають додаткові і розширені команди. Кожна команда виконується за один такт. Багатофункціональні команди мікропроцесора об'єднують декілька пересилань даних з арифметико-логічною обробкою. Всі пристрої мікропроцесора є 16-розрядними і оперують з даними у форматі з фіксованою точкою. Числа подаються або як беззнакові, або в додатковому коді. Логічні операції виконуються над бітовими рядками.

Вдосконалення даного сімейства мікропроцесорів йде у напрямі підвищення тактової частоти, зниження енергоспоживання і розширення комунікаційних можливостей процесора.

3.2.2. ЦПОС із фіксованою точкою фірми Motorola (DSP 5600xx)

Сигнальні мікропроцесори компанії Motorola підрозділяються на сімейства 16- і 24-розрядних мікропроцесорів з фіксованою точкою – DSP 560xx, -561xx, -563xx, -566xx, -568xx і мікропроцесори з плаваючою точкою – DSP960xx.

Лінія 24-розрядних мікропроцесорів компанії Motorola включає два сімейства: DSP560xx і DSP563xx [60]. Основні принципи, покладені в основу архітектури сигнальних мікропроцесорів Motoro1a, були розроблені і втілені в сімействі DSP560xx. Подальші роботи для вдосконалення сигнальних процесорів проводилися по трьох напрямах:

• нарощування продуктивності 24-розрядних процесорів за рахунок конвейєризації функціональних модулів і підвищення тактової частоти;

• створення дешевих 16-розрядних мікропроцесорів з розширеними засобами взаємодії з периферією;

• розробка високопродуктивних процесорів, що включають блок обчислень з плаваючою точкою.

Далі послідовно будуть розглянуті всі три напрями на прикладі найпопулярніших представників мікропроцесорних сімейств. Будуть вказані також найістотніші відмінності процесорів в рамках одного сімейства.

Мікропроцесори DSP56000/DSP56001 є першими представниками лінії сигнальних процесорів компанії Motoro1a [60]. Архітектура мікропроцесорів орієнтована на максимізацію пропускної спроможності в додатках 08Р з

інтенсивним обміном даними. Це забезпечується завдяки розширеній архітектурі із складною вбудованою периферією і універсальній підсистемі вводу/виводу. Дані властивості, а також низьке енергоспоживання мінімізують складність, вартість і терміни розробки прикладних систем на базі мікропроцесорів DSP56000/DSP56001.

Мікропроцесори працюють на частотах до 33 Мгц і забезпечують продуктивність близько 16 MIPS, що дозволяє виконувати швидке перетворення Фур'е по 1024 відліках за 3,23 мс.

Відмінність між даними процесорами полягає в типі їх внутрішньої пам'яті. З метою мінімізації вартості прикладних систем мікропроцесор DSP56000 орієнтований на роботу під управлінням програми, що зберігається в НПЗП (ROM), ємністю 3,75 К слів. Існує також варіант процесора DSP56000, який володіє властивостями захисту від несанкціонованого доступу до програми, що зберігається у внутрішній пам'яті. DSP56001 містить на кристалі пам'ять довільного доступу (RAM), ємністю 512 слів, 32 слова пам'яті (ROM) програми початкового завантаження процесора із зовнішнього джерела, а також два модулі пам'яті, заздалегідь запрограмованих як таблиці функцій експандування за А- і μ-законами, і таблиці синусоїдального перетворення.

Структура мікропроцесорів DSP56000 і DSP56001 зображена на рис. 3.11 і 3.12 відповідно.

Призначення основних компонентів структури мікропроцесорів буде описане нижче, при розгляді загальної структури мікропроцесорного сімейства DSP560xx. Подальший розвиток сімейства мікропроцесорів DSP560xx здійснювався в рамках концепції процесорного ядра, загального для всіх представників сімейства, до складу якого входять 24-розрядні мікропроцесори з фіксованою точкою DSP 56002, 4, 7, 9, 11 [60].

Процесори даного сімейства характеризуються високою пропускною спроможністю, розширеною розрядністю, що забезпечує високу точність обчислень, і широким динамічним діапазоном даних, що обробляються, підтримкою енергозберігаючого режиму роботи. Представники сімейства відрізняються один від одного конфігураціями пам'яті і периферійними пристроями.

Типова структура представника сімейства DSP560xx мікропроцесора наведена на рис. 3.13. Основними компонентами мікропроцесора є:

- шини даних;
- шини адрес;
- АЛП даних (ALU);
- пристрій генерації адрес (AGU);
- пристрій програмного управління (PCU);
- розширення пам'яті (порт А);

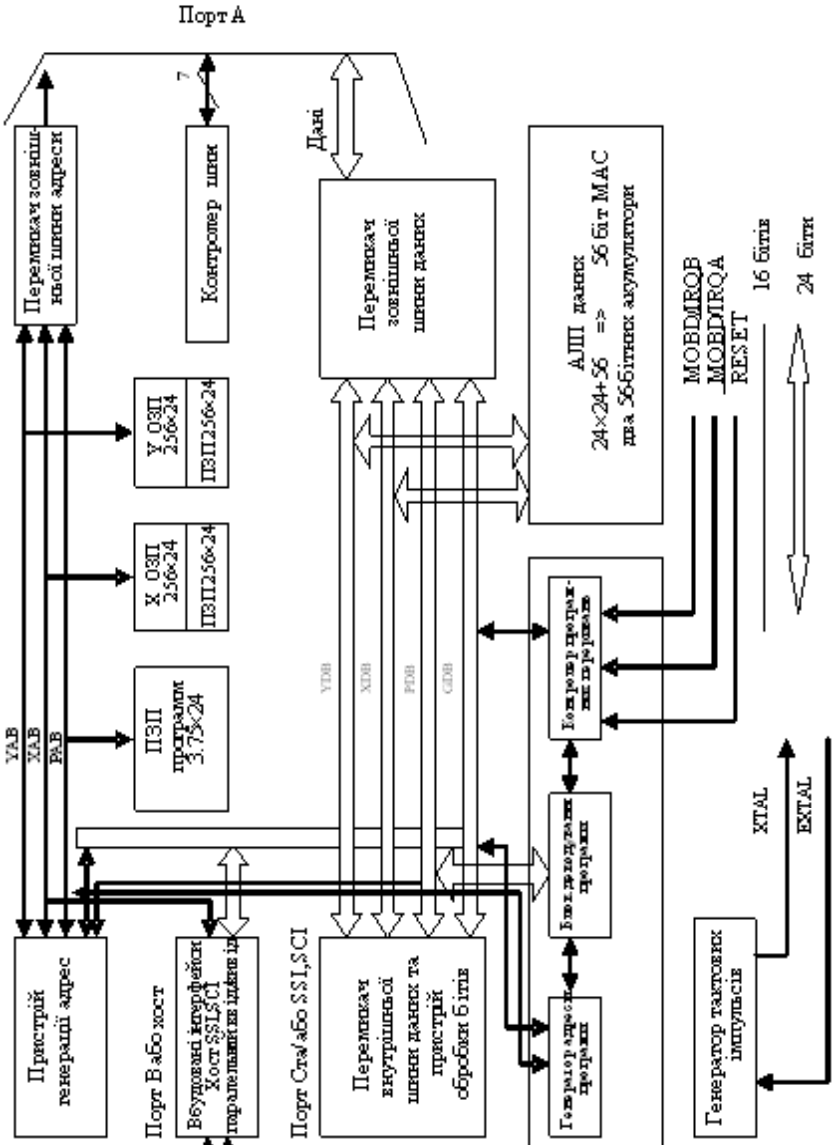

Рис. 3.11. Структура мікропроцесора DSP 56000

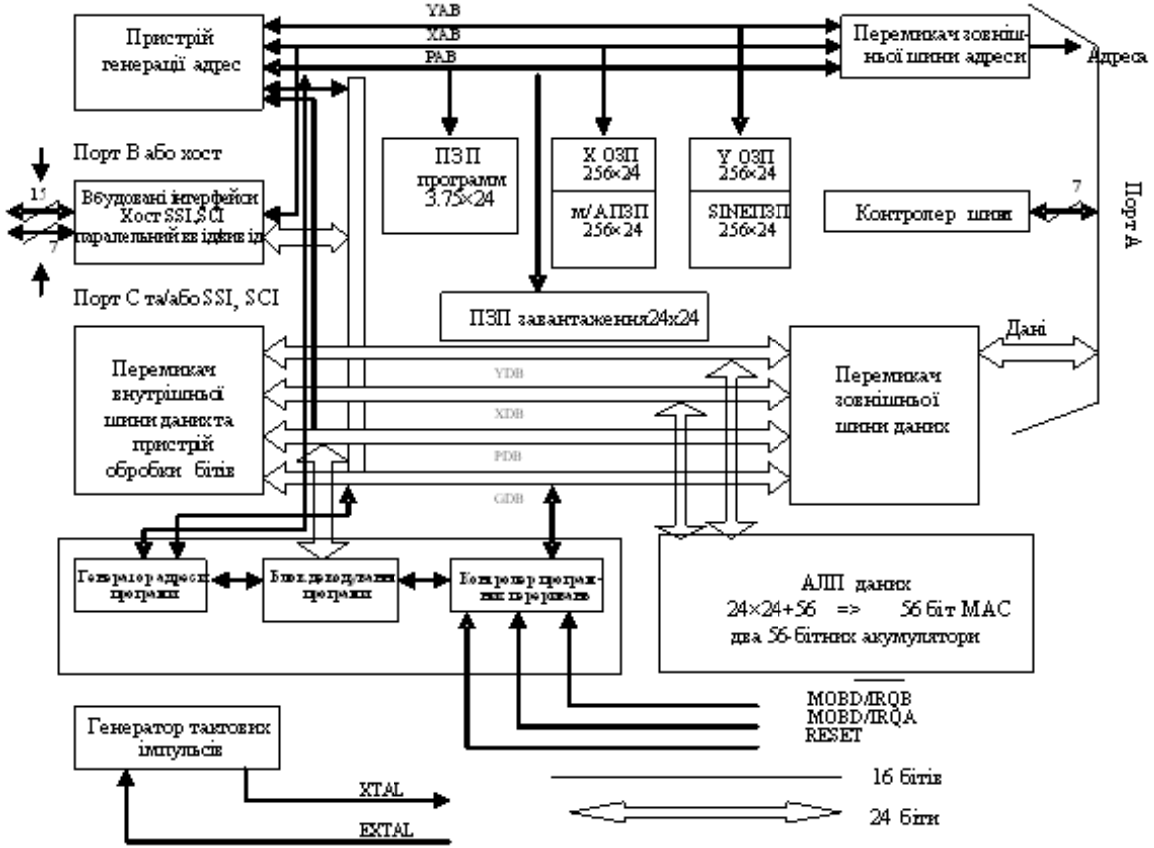

Рис. 3.12. Структура мікропроцесора DSP 56001

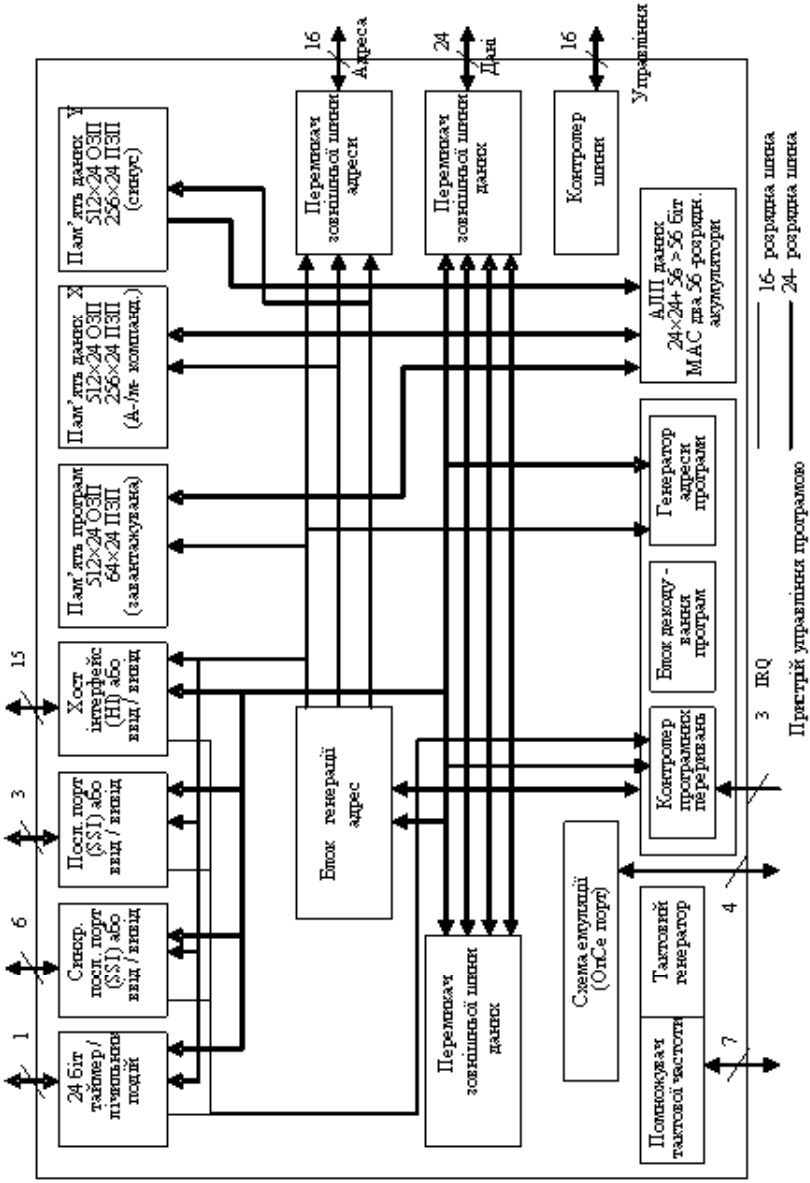

Рис. 3.13. Структура мікропроцесора сімейства DSP560xx

• внутрішньокристальна схема емуляції (OnCEдд);

• схема множення частоти.

Процесор містить три незалежні виконавчі пристрої: PCU, AGU і АЛП даних. Пересилання даних між регістрами виконавчих пристроїв здійснюється по двоспрямованих 24-розрядних шинах: шині даних X (XDB), шині даних У (YDB), програмній шині даних (PDB) і глобальній шині даних (GDB). Деякі команди використовують шини даних X і У як єдину 48-розрядну шину. Для підвищення швидкості вибірки операнди команди завантажуються в АЛП з модулів пам'яті X і У по незалежних шинах XDB і YDB, а команда – по програмній шині даних PDB. Обмін даних з периферійними пристроями здійснюється по шині GDB.

Шинна структура підтримує основні пересилання даних типу регістр-регістр, регістр-пам'ять, пам'ять-регістр. За один такт можуть бути передані два 24-бітових і одне 56-бітове слова. Обмін між шинами здійснюється через внутрішній комутатор – матрицю, що дозволяє з'єднати будь-які дві внутрішні шини без додавання тактів затримки. Адреси для внутрішніх X- і Y- пам'яті даних передаються по двоспрямованих 16-розрядних шинах XAB і YAB, а адреси пам'яті команд – по двоспрямованій програмній шині (PAB). Зовнішня пам'ять адресується за допомогою односпрямованої шини, що є виходом мультиплексора шин з трьома входами: XAB, YAB, PAB.

Пристрій бітових операцій фізично розташований в блоці комутатора, що забезпечує йому доступ до будь-якої області пам'яті і дозволяє виконувати бітові операції над даними в пам'яті, регістрах, вмістом адресних і управляючих регістрів.

АЛП даних мікропроцесора виконує над даними всі арифметичні і логічні операції і містить чотири 24-бітові регістри-джерела, два 48-бітові регістри-акумулятори, два 8-бітові регістри розширення акумуляторів, пристрій зсуву акумулятора, дві схеми зсуву/обмеження даних і паралельний (не конвеєризований) однотактовий пристрій множення з накопиченням (MAC).

Акумулятори A і B служать як буферні регістри шин XDB і YDB, а їх 8-бітові регістри розширення використовуються схемою зсуву/обмеження для фіксації і обробки ситуацій переповнення в результаті арифметичних операцій або зсуву.

АЛП даних дозволяє виконувати множення в режимі подвоєної точності (задається установленням відповідного біта в регістрі стану процесора). Результат множення двох 48-бітових операндів має 96 розрядів і міститься в 4-х 24-бітових регістрах.

Пристрій генерації адреси (AGU) працює паралельно з іншими компонентами процесора, забезпечуючи обчислення необхідних адрес даних в пам'яті за один такт за допомогою двох однакових 16-бітових

арифметичних пристроїв, кожен з яких може виконувати лінійні, модульні і циклічні арифметичні операції.

З кожним адресним АЛП зв'язані три набори з 4-х регістрів: адресних – R0-R3 і R4-R7, зсувів – N0 - N3, N4 - N7 і модифікаторів M0 -M3 і M4 - M7. Регістри використовуються трійками: RO:NO:MO, R1:N1:M1, R2:N2:M2, R3:N3:M3, R4:N4:M4, R5:N5:M5, R6:N6:M6 і R7:N7:M7. Адреса формується з вмісту адресного регістру і регістру зсуву з урахуванням типу арифметики, який визначається вмістом регістра модифікатора.

Пристрій програмного управління генерує адреси програми (попередня вибірка команд), декодує, апаратно обробляє команди циклічного переходу, внутрішні і зовнішні переривання або виняткові ситуації. Він містить 15-рівневий 32-бітовий системний стек (SS) і 6 регістрів, що безпосередньо адресуються: лічильник команд (PC), лічильник циклу (LC), регістр адреси циклу (LA), регістр стану (SR), регістр режиму (OMR) і покажчик стека (SP); 16-бітовий регістр PC може адресувати до 65536 команд. SS зберігає PC і SR при виклику процедур, обробці переривань і виконанні програмних циклів.

Команди процесора виконуються в триетапному (передвибірка, декодування, виконання) конвеєрі з подальшим аналізом 5 можливих станів процесора: "нормальний", "виключення", "скидання", "очікування" і "останов".

До складу PCU входять три блоки: блок декодування програми (PDC), генератор адреси програми (PGA) і програмний контролер переривань (PIC). PDC декодує команди, завантажені в командний буфер, і генерує всі необхідні для виконання команди управляючі сигнали. Вміст командного буфера дублюється для ефективнішого виконання команд повторення (REP) і переходу.

Основне призначення блока PGA – апаратне формування адрес циклів. При ініціалізації циклу адреса його початку поміщається в стек, значення змінної циклу міститься в регістрі LC, адреса кінця циклу – в LA. При завершенні чергової ітерації адреса переходу витягується із стека, тобто не формується програмно, що істотно підвищує швидкість обробки.

PIC одержує всі запити на переривання, класифікує їх і генерує адресу вектора переривань. Переривання можуть бути маскованими – що дорівнюють 0 (нижній рівень), 1, 2, і немаскованими – рівень 3 (вищий рівень).

Порт розширення пам'яті A забезпечує синхронний обмін даними з різними типами пам'яті і зовнішніми пристроями по 24-розрядній шині даних. Порт працює з високо- і низькошвидкісною пам'яттю, а також іншими універсальними і сигнальними процесорами в режимі master/slave.

Внутрішньокристальний емулятор – це схема, що дозволяє інтерактивно аналізувати стан регістрів, пам'яті, периферійних пристроїв і управляти процесом відлагодження програми – дозволяти відлагодження для розробника системи або забороняти іншим користувачам доступ до внутрішніх ресурсів процесора.

Помножувач частоти дозволяє процесору працювати на підвищеній внутрішній тактовій частоті, забезпечуючи синхронізацію внутрішніх і зовнішніх тактових імпульсів, а також зниження частоти в енергозберігаючому режимі.

Програмна модель мікропроцесора подається у вигляді трьох діючих паралельно функціональних пристроїв: ALU, AGU і PCU. Система команд орієнтована на ефективну підтримку мови С і організована так, щоб забезпечити зайнятість цих пристроїв протягом кожного такту, досягаючи при цьому максимальної швидкості виконання програми.

Команди мікропроцесора мають змінну довжину: 1 або 2 24-бітових слова. Типова команда мікропроцесора містить поле коду операції, що визначає відповідну дію ALU, AGU або PCU, поле операндів і два поля, що задають пересилання, які виконуються паралельно з основною операцією по шинах XDB і YDB. Приклад команди MAC наведено в табл. 3.3.

Таблиця 3.3 – Структура команди мікропроцесора DSP-560xx

Opcode	Operands	XDB	YDB
MAC	XO.YO.A	X:(RO)+,XO	У:(R4)+,YO

3.2.3. ЦПОС із фіксованою точкою фірми Texas Instruments (TMS 320 C2x)

Сигнальні процесори компанії Texas Instruments [60] поділяються на два класи: це процесори для обробки чисел з фіксованою та плаваючою точками (рис. 3.14.). Перший клас представлений трьома сімействами процесорів, базовими моделями яких є відповідно TMS320.10, .20, .50. Другий клас включає процесори TMS320.30, .40, TMS320C80, які підтримують операції з плаваючою точкою і які є мультипроцесорною системою, виконаною на одному кристалі, а сімейство TMS320C6x включає процесори як з фіксованою так і з плаваючою точками.

Процесори старших поколінь одного сімейства успадковують основні архітектурні особливості і є сумісними від низу до верху за системою команд (чого не можна сказати про процесори, що входять в різні сімейства).

Модифікована гарвардська архітектура TMS 320 C2x.

Перший процесор сімейства TMS320C10 був випущений в 1982 р. і завдяки ряду вдалих технічних рішень набув великого поширення [60].

Структура типового представника сімейства – мікропроцесора TMS320C15 наведена на рис. 3.15.

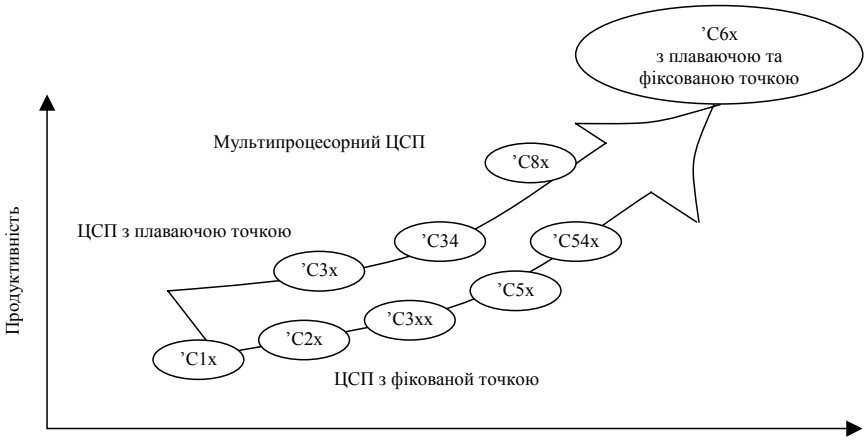

Рис 3.14. Сімейства мікропроцесорів компанії TI

У основі мікропроцесора лежить модифікована гарвардська архітектура, відмінністю якої від традиційної гарвардської архітектури є можливість обміну даними між пам'яттю програм і пам'яттю даних, що підвищує гнучкість пристрою.

TMS320C10 є 16-розрядним процесором. Його адресний простір складає 4К 16-розрядних слів пам'яті програм і 144 16-розрядних слів пам'яті даних. Тривалість командного такту процесора складає 160 – 200 нс.

Арифметичні функції в процесорі реалізовані апаратно. Він має апаратний помножувач (MULT), пристрій зсуву (SHIFTER), апаратурну підтримку автоінкремента/декремента адресних регістрів даних (AR0, AR1)

Із зовнішніми пристроями процесор взаємодіє через 8 16-розрядних портів вводу/виводу. Передбачена обробка зовнішнього переривання.

Інші мікропроцесори даного сімейства (C14-C17) мають аналогічну архітектуру і відрізняються тривалістю командного такту, конфігурацією пам'яті, наявністю (або відсутністю) додаткових периферійних пристроїв (наприклад, в C17-кодек даних за μ-/A-законом, перетворювач логарифмічної імпульсно-кодової модуляції (ІКМ) в лінійну ІКМ).

Мікропроцесори сімейства TMS320C2x аналогічні за архітектурою, але мають підвищену продуктивність і ширші функціональні можливості [60]. Всі процесори цього сімейства можуть використовувати по 64К слів пам'яті

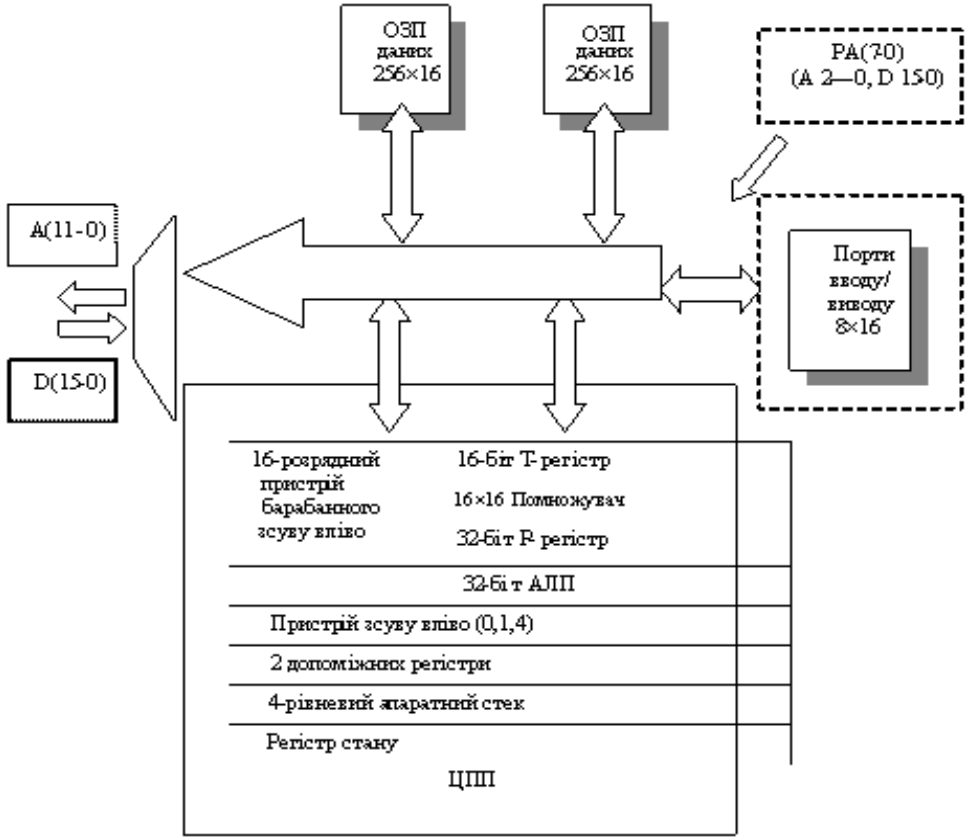

Рис. 3.15. Структура мікропроцесора сімейства TMS320C1x

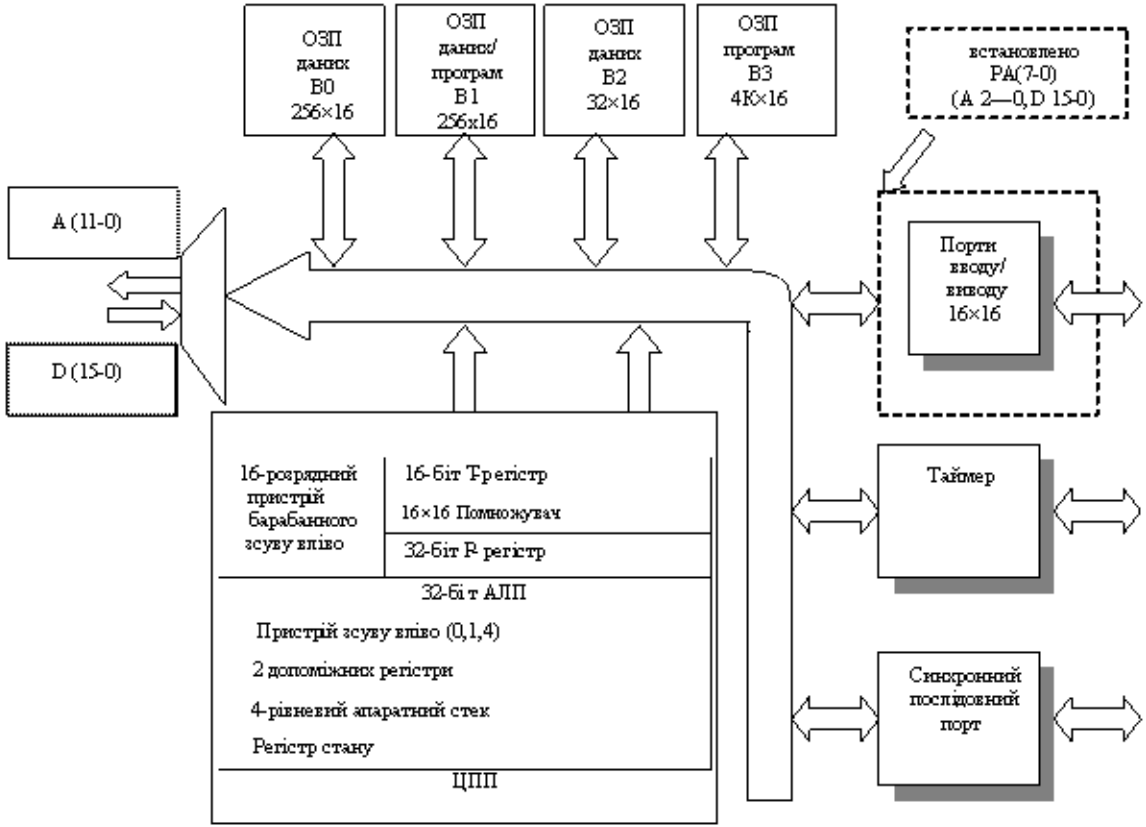

Рис.3.16. Структура мікропроцесора TMS320C2x

Таблиця 3.4 – Основні характеристики процесорів сімейства TMS320C2x

Мікропроцесор	Технологія	Командний цикл	Внутрішня пам'ять			Зовнішня пам'ять		Ввід/вивід		
		(нс)	ОЗП	ПЗП	НПЗП	Дані	Програма	Послідовний порт	Паралельний порт	Прямий доступ до пам'яті (ПДП)
TMS32020	NMOS	200	544			64K	64K	1	16x16	Так
TMS320C25	CMOS	100	544	4K		64K	64K	1	16x16	Так
TMS320C25-50	CMOS	80	544	4K		64K	64K	1	16x16	Так
TMS320C25	CMOS	100	544		4K	64K	64K	1	16x16	Так
TMS320C26	CMOS	100	1568	256		64K	64K	1	16x16	Так

програм і даних, мають по 16 16-розрядних портів вводу/виводу і послідовний порт.

Структура мікропроцесора TMS320C2x наведена на рис. 3.16.

Процесори сімейства TMS320C2x мають можливість використовувати зовнішній контролер ПДП. Помножувач мікропроцесорів, крім операцій множення, дозволяє виконувати за один такт зведення в квадрат. У процесори включена апаратна підтримка кратного виконання команди, реалізований режим двійкової непрямої адресації, призначений для ефективної реалізації ШПФ.

Основні технічні характеристики процесорів 2-го покоління наведено в табл.. 3.4.

Основні відмінності архітектури процесорів TMS3202x від TMS3201x полягають в такому:

• виконання множення і збереження результатів в TMS3202x здійснюється за один командний цикл;

• набір команд підтримує обчислення з плаваючою точкою;

• є внутрішній маскований ПЗП програм (ROM), розміром 4К слів для TMS320C25 або ПЗП з ультрафіолетовим стиранням (EPROM) 4К слів для TMS320E25;

• виконання програм здійснюється з пам'яті програм RAM, розташованої на кристалі. Обсяг пам'яті програм RAM становить 544 слова, з яких 256 можуть бути використані як пам'ять даних;

• розширена зовнішня пам'ять може мати обсяг 128К слів (64К слів – пам'ять програм, 64К – пам'ять даних);

• TMS3202x містить зовнішній інтерфейс для організації багатопроцесорних зв'язків і засобу синхронізації для доступу до пам'яті, що розділяється;

• можливість обміну пам'яті даних і програм блоками;

• можливість організації циклів очікування при доступі до повільної зовнішньої пам'яті або до повільних периферійних пристроїв;

• TMS3202X містить на кристалі таймер і послідовний порт;

• наявність п'яти (TM532020) або восьми (TM5320C25) допоміжних регістрів і спеціального арифметичного пристрою для них;

• наявність апаратного стека розміром 4 слова для TM832020 або 8 слів для TM5320C25 і можливості програмного розширення стека в пам'яті даних;

• наявність команд обробки бітових даних;

• наявність трьох маскованих користувачем переривань;

• наявність режиму прямого доступу до пам'яті (ПДП) (тільки для TMS320C25).

Архітектура й призначення основних модулів TMS320C2x - ЦАЛП, ДАП, лічильника команд. Центральний арифметико-логічний пристрій (ЦАЛП) містить 16-розрядний регістр із масштабуванням та зсувом, паралельний помножувач 16x16, 32-розрядний арифметико-логічний пристрій (ALU), 32-розрядний акумулятор (ACC) та додаткові зсувні пристрої на виходах помножувача та акумулятора.

Виконання команди в ЦАЛП звичайно складається з таких дій:

• отримання даних з пам'яті через шину даних;

• обробка даних в регістрі з масштабуванням та зсувом, помножувачі, ALU;

• збереження результатів в акумуляторі.

Один з виходів ALU з'єднаний з виходом акумулятора ACC, інший може бути підключений до виходу регістру із масштабуванням та зсувом, або до виходу помножувача. Регістр із масштабуванням та зсувом зв'язує шину даних з ALU та забезпечує зсув даних вліво на 16 розрядів (SFL(0-16). Молодші біти слова, що зсувається, заповнюються нулями або бітом знака (SX), якщо встановлено біт режиму розширення знака регістру стану ST1.

ЦАЛП містить додатковий пристрій зсуву на виході помножувача, який дозволяє виконати зсув вправо на шість розрядів (SFR(6) з урахуванням біта знака SX, або вліво на 1 або 4 розряди (SFL(1,4). Крім цього вміст акумулятора може бути зсунуто вліво на 7 розрядів при виводі даних.

ALU забезпечує виконання великого набору арифметичних та порозрядних логічних операцій за один командний цикл, оперуючи при цьому 16-розрядними словами, що надходять з пам'яті даних чи безпосередньо з команди.

ALU підтримує операції з плаваючою точкою. До системи команд процесора включено команду NORM, яка дозволяє нормалізувати число з фіксованою точкою, що міститься в акумуляторі. Це досягається шляхом вилучення зайвих знакових бітів. Команда NORM реалізується за допомогою зсуву вліво. Команда LAST дозволяє виконати зворотну операцію – денормалізацію числа з плаваючою точкою за рахунок зсуву мантиси. В цьому випадку кількість зсувів задається чотирма молодшими бітами Т-регістру. Операція зсуву виконується в регістрі із масштабуванням та зсувом. Вміст Т-регістра визначає значення експоненти.

Програмування режиму переповнення акумулятора може виконуватися командами SOVM та ROVM, які встановлюють або скидають відповідно біт режиму переповнення OVM регістру стану ST1. Якщо OVM=1 та відбувається переповнення, то акумулятор звантажується максимальним додатним числом (7FFF FFFFh) або мінімальним від'ємним числом (8000 0000h) залежно від напрямку переповнення. При OVM=0

результати в акумуляторі не модифікуються. Логічні операції не скидають прапор переповнення.

ALU та акумулятор дозволяють виконувати широкий набір команд безумовного та умовного переходів, наприклад команду переходу при виникненні переповнення BV, команду BZ, яка перевіряє, чи дорівнює нулю вміст акумулятора, команду переходу за адресою, що міститься в акумуляторі, та ін.

Акумулятор процесора складається з двох 16-розрядних регістрів: ACCH (старше слово акумулятора) і ACCL (молодше слово акумулятора). Пристрій зсуву на виході акумулятора забезпечує зсув на 7 розрядів вліво. Зсув виконується під час передачі даних до пам'яті і тому не впливає на вміст акумулятора.

Акумулятор містить біт переносу C, який дозволяє більш ефективно виконувати арифметичні операції з підвищеною точністю. Для цього можуть використовуватися команди ADDC (складання з урахуванням переносу) та SUBB (віднімання з урахуванням позики). Біт переносу може змінювати значення під впливом багатьох арифметичних команд та команд зсуву. На значення біта переносу не впливають логічні команди, команди завантаження акумулятора, команди множення MPY, MPYK, MPIU. Значення біта переносу C аналізується в командах умовного переходу BC та BNC. Біт переносу C бере участь в операціях зсуву акумулятора SFL, SFR, ROL, ROR.

У ЦПОС використано апаратний помножувач 16x16 біт, який дозволяє отримувати 32-розрядний результат з урахуванням чи без урахування знака операндів команди множення. Помножувач може обчислювати добуток двох операндів, що надійшли з пам'яті даних та пам'яті програм або тільки з пам'яті даних. Результат виконання операції множення поміщається в P-регістр. При передачі вмісту P-регістру в АЛП чи пам'ять даних може виконуватися операція зсуву добутку вліво на 1 або 4 біта та вправо на 6 біт. Конкретна кількість розрядів, на яку зсувається добуток, залежить від значення 2-розрядного поля PM регістру стану ST1. Зсув вправо на 6 біт дозволяє реалізувати 128 команд множення з накопиченням результатів в акумуляторі, що усуває небезпеку виникнення переповнення.

Для управління помножувачем використовуються команди: LT – завантаження T-регістру вмістом завданої комірки пам'яті даних; MPY – множення операндів з T-регістру та пам'яті даних; MPYK – обчислення добутку вмісту T-регістру та константи, що задана в команді. Крім цього до системи команд ЦПОС входять команди MAC, MACD, MPYA, MPYS, які управляють як помножувачем, так і АЛП з акумулятором. Ці команди дозволяють виконати множення з одночасним накопиченням результатів в акумуляторі. При виконанні команд MAC та MACD операнди отримуються з пам'яті програм за один цикл. Це дозволяє командам MAC та MACD

виконуватися також за один цикл, якщо вони використовуються сумісно з командами RPT або RPTK.

ЦПОС також містить додатковий арифметичний пристрій (ДАП), до якого входять: допоміжні регістри AR0 – AR7, покажчик допоміжного регістру ARP, буфер допоміжного регістру ARB, арифметичний пристрій допоміжних регістрів (ARAU), покажчик сторінок DP, мультиплексори MUX.

Дев'ятирозрядний регістр сторінок DP використовується при прямій адресації пам'яті даних. Для цього сім молодших розрядів слова команди об'єднуються із вмістом DP.

Допоміжні регістри AR0 – AR7 використовуються при непрямій адресації пам'яті даних. При цьому вміст допоміжних регістрів розглядається як 16-розрядна адреса пам'яті даних. Для вибору одного з допоміжних регістрів застосовується 3-розрядний регістр-покажчик ARP, до якого завантажується значення від 0 до 7. Процесор має спеціальні команди для первинного завантаження регістрів AR0 – AR7 та ARP. Покажчик ARP може бути завантажений значенням або з пам'яті даних (через шину даних), або з трьох молодших розрядів команди (через шину команд). Кожного разу, коли ARP завантажується новим значенням, старе значення зберігається в буфері ARB (за виключенням команди LST).

Пристрій ARAU дозволяє після первинного завантаження виконувати індексацію допоміжного регістру, на який вказує покажчик ARP. Індексація виконується шляхом збільшення (зменшення) вмісту ARn на одиницю, або шляхом складання (віднімання) вмісту AR0 з вмістом ARn. Індексація здійснюється паралельно з операціями, що виконуються в ЦАЛП.

Хоча пристрій ARAU розроблено для підтримки маніпуляцій з адресами, він може також використовуватися як арифметичний пристрій загального призначення. ARAU оперує 16-розрядними аргументами без знака, а також забезпечує реалізацію команди передачі управління (BANZ), що заснована на порівнянні вмісту AR0 з допоміжним регістром, на який вказує ARP. Це дозволяє реалізовувати за допомогою ARAU циклічні алгоритми. У цьому випадку один з допоміжних регістрів застосовується як лічильник циклів, а AR0 містить його кінцеве значення.

Управління обчислювальним процесом в ЦПОС здійснюється за допомогою лічильника команд (PC), апаратного стека, сигналів переривань та скидань, регістрів станів, таймера лічильника числа повторень.

ЦПОС містить 16-розрядний лічильник команд (PC) та 8-рівневий апаратний стек. Лічильник команд адресує пам'ять програм. Стек використовується під час обробки переривань та виклику підпрограм.

Лічильник команд формує адресу чергової команди та передає її на шину адреси команд. Команди з пам'яті програм завантажуються до регістру

команди (IR). В момент завантаження регістра IR лічильник команд містить адресу наступної команди. Лічильник команд може також адресувати пам'ять даних. Це відбувається під час виконання команди BLKD, яка виконує пересилання блоків даних до пам'яті даних.

Лічильник команд зв'язаний з шиною даних через мультиплексор (MUX) та може бути завантажений вмістом акумулятора. Це використовується в командах передачі управління за адресою, що знаходиться в акумуляторі (BACC, CALA). На початку нового циклу вибірки команди вміст лічильника команд збільшується на одиницю, або в нього завантажується адреса, за якою слід передати управління.

Якщо відбувається виклик підпрограм або обробка переривань, то вміст лічильника команд зберігається у стеку. Для роботи із стеком використовуються команди PUSH (занести до стека) та POP (вилучити із стека). Коли вміст лічильника команд заноситься у верхівку стека, старе значення верхівки переміщується на рівень нижче; значення, що знаходилося внизу стека, втрачається. Старе значення верхівки стека може бути втрачене, якщо до вилучення його із стека команда PUSH буде виконана 8 разів. При вилученні значень із стека може статися зворотне переповнення стека, якщо виконати команду POP більш ніж 7 разів. В цьому випадку на всіх рівнях стека буде знаходитися те саме значення.

ЦПОС містить дві додаткові команди (PUSHD та POPD), які дозволяють зберегти вміст верхівки стека у пам'яті даних та вилучати його з пам'яті даних. Ці команди забезпечують організацію стека довільної глибини у пам'яті даних.

У ЦПОС TMS320C25 кожна команда виконується впродовж трьох машинних циклів: попередньої вибірки, декодування та виконання. Лічильник попередньої вибірки (PFC) містить адресу наступної команди, яка буде обрана. Ця команда завантажується до регістру команд (IR). Якщо він зайнятий командою, що виконується, то попередньо обрана команда зберігається в регістрі черги команд (QIR). Далі інкрементується вміст PFC, і після виконання поточної команди вміст QIR завантажується в IR. Таким чином, лічильник команд, хоча і містить адресу наступної команди, але безпосередньо в операціях вибірки не використовується, а по суті є покажчиком поточної команди програми. Вміст PC інкрементується після того, як завершиться виконання команди.

Наявність 3-етапного конвеєра дозволяє в кожному машинному циклі обробляти три команди, що знаходяться на етапах виконання. Конвеєр стає 2-етапним, якщо програма, що виконується, зчитується із внутрішнього ОЗП. При звертанні до внутрішнього ОЗП вибірка та декодування команди виконуються в тому ж самому машинному циклі.

Організація пам'яті TMS320C2x і розподіл адресного простору. Процесори сімейства TMS320C2x забезпечують роботу з трьома адресними просторами: пам'яті даних, пам'яті програм та вводу-виводу. Процесор може виконувати дії над вмістом внутрішньої (розташованої на кристалі) пам'яті або над вмістом зовнішньої пам'яті (рис. 3.16).

Внутрішня пам'яті даних представлена трьома блоками ОЗП (B0, B1, B2) та шістьма регістрами (DRR, DXR, TIM, PRD, IMR, GREG). Ємність ОЗП складає 544 16-розрядних слова. З них 256 слів (блок B0) можуть використовуватися або як пам'ять даних, або як пам'ять програм. Інші 288 слів (блоки B1, B2) завжди використовуються як пам'ять даних. Загальна ємність пам'яті даних складає 64К 16-розрядних слова. Адреси, що відповідають внутрішній пам'яті даних, не перевищують 1024 (0400h). Адреси регістрів, що відображаються на пам'ять даних наведено в табл.. 3.5.

Таблиця 3.5 – Адреси регістрів

Регістр	Адреса	Призначення
DRR(16)	0	Регістр прийому послідовного порту
DXR(16)	1	Регістр передачі послідовного порту
TIM(16)	2	Таймер
PRD(16)	3	Регістр періоду таймера
IMR(6)	4	Регістр маски переривань
GREG(8)	5	Регістр розподілу глобальної пам'яті

Внутрішня пам'ять програм представлена ПЗП ємністю 4Кх16 та блоком B0, якщо його розподілено на пам'ять програм командою CNFP. Внутрішня пам'ять програм дозволяє процесору функціонувати з максимальною швидкодією, при цьому зовнішня шина даних D15 – D0 залишається вільною для звертання до пам'яті даних. Після надходження сигналу $\overline{RS}$ блок B0 відображається на пам'ять даних.

Розподіл перших 4К слів пам'яті програм задається сигналом $MP/\overline{MS}$. Якщо сигнал $MP/\overline{MS}$=1, то зазначений адресний простір відводиться під зовнішню пам'ять програм. Якщо $MP/\overline{MS}$=0, то адреси від 32 до 4015 відводяться під ПЗП програм. Звертання до зовнішньої пам'яті програм/даних здійснюється при активних сигналах $\overline{PS}/\overline{DS}$, $\overline{STRB}$. Під час звертання до внутрішньої пам'яті ці сигнали неактивні.

Дії, що виконуються за тією чи іншою командою, суттєво залежать від пам'яті, що використовується. При цьому можливі чотири комбінації видів пам'яті: PI/DI, PI/DE, PE/DI, PE/DE. З урахуванням того, що внутрішня пам'ять може бути представлена ПЗП (PI) чи блоком B0 (PR), отримуємо шість різних комбінацій. Ці обставини необхідно мати на увазі при визначенні кількості циклів, що потрібні для виконання команди.

Реалізація конвеєра команд і апаратної адресації в TMS320C2x. У ЦПОС TMS320C2x з метою скорочення тривалості командного циклу та підвищення продуктивності використано модифіковану гарвардську архітектуру, яка дозволяє здійснювати обмін даними між пам'яттю програм і пам'яттю даних та конвеєрний режим роботи, що дає можливість обробляти одночасно декілька команд.

В архітектурі процесорів цього сімейства використано 3-етапний конвеєр. Процесор може одночасно обробляти 3 команди, які знаходяться на різних етапах виконання: попередня вибірка, декодування, виконання. Наприклад, коли здійснюється вибірка *n*-ї команди, то попередня команда (*n*-1) знаходиться на етапі декодування, команда (*n*-2) – на етапі виконання. Конвеєрний режим роботи процесора залишається для користувача непомітним, за виключенням команд передачі управління, коли необхідне перезавантаження конвеєра.

При звертанні до зовнішньої пам’яті програм чи даних адресація або зчитування даних виконується за допомогою однієї зовнішньої шини адреси (A15 – A0) та однієї зовнішньої шини даних (D15 – D0) (рис. 3.16). Тому звертання до зовнішньої пам’яті програм та даних можуть виконуватися тільки послідовно. З метою підвищення швидкодії процесора та можливості паралельного доступу до пам’яті програм та пам’яті даних до складу процесора включено внутрішньокристалічні ОЗП даних/програм та ПЗП програм.

Адреси внутрішньокристалічного ОЗП, що відповідають блоку (B0), можуть бути розподілені на пам’ять даних, або на пам’ять програм. Для цього відповідно використовуються команди CNFD/CNFP. Програми, що зберігаються у блоці B0, виконуються з максимально можливою швидкістю. Блоки B1 та B2 завжди використовуються як пам’ять даних. До внутрішньокристалічного ОЗП даних відносять також шість регістрів, розподілених на адресний простір пам’яті даних, регістри послідовного порту (DRR та DXR), таймер (TIM), регістр періоду таймера (PRD), регістр маски переривань (IMR) та регістр розподілу глобальної пам’яті (GREG).

Контрольні питання

1. Що таке модифікована гарвардська архітектура?
2. Перелічити основних виробників ЦПОС з фіксованою точкою та основні сімейства цих процесорів.
3. У чому полягають основні відмінності між ЦПОС сімейств ADSP21xx та ADSP210xx?
4. Склад та призначення основних модулів ЦПОС сімейства ADSP21xx.

5. Склад та призначення основних модулів мікропроцесорного ядра ЦПОС сімейства ADSP21xx.

6. Склад та призначення основних модулів ЦПОС сімейств DSP56000 та DSP56001.

7. Склад та взаємодія між основними модулями ЦПОС сімейства DSP560xx.

8. Архітектура та призначення центрального арифметичного пристрою ЦПОС TMS320C2x.

9. Склад та призначення додаткового арифметичного пристрою ЦПОС TMS320C2x.

10. Організація пам'яті ЦПОС TMS320C2x і розподіл адресного простору.

11. Призначення та принцип дії конвеєра команд ЦПОС TMS320C2x.

12. Організація апаратної адресації в ЦПОС TMS320C2x.

3.3. Архітектура ЦПОС із плаваючою точкою ADSP-2106x

3.3.1. Супергарвардська архітектура (Super Harvard Architekture) ADSP-2106x

Представники сімейства ADSP 21xxx – ADSP 21060 і ADSP-21062 – високопродуктивні сигнальні процесори, що працюють на частоті 40 Мгц і що мають швидкодію до 80 MIPS і 120 MFLOPS [61]. Будучи схожими з раніше розглянутими ADSP-210xx за структурою мікропроцесорного ядра і сумісні знизу-вгору за системою команд, ці процесори мають деякі дуже істотні доповнення, орієнтовані перш за все на розширення комунікаційних можливостей.

Мікропроцесори сімейства ADSP-2106x мають другу назву – SHARC, пов'язану з особливостями їх архітектури (Super Harvard Architecture Computer), яка задає новий стандарт інтеграції сигнальних процесорів в мультипроцесорну систему. У SHARC-мікропроцесорі об'єднані високоефективне процесорне ядро, що виконує обробку даних у форматі з плаваючою точкою, інтерфейс з хост-процесором, контролер ПДП, послідовні порти і шина, що розділяється. Узагальнена архітектура ADSP-2106x наведена на рис. 3.17 [60].

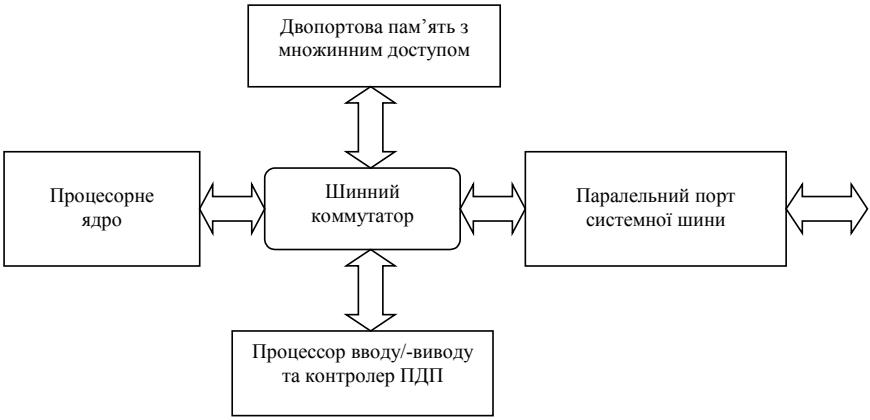

Рис. 3.17. Узагальнена архітектура ADSP-2106x (SHARC-архітектура)

Архітектуру SHARC створено на основі ядра цифрового сигнального процесора ADSP -21000. Для формування однокристальної мікро-ЕОМ до нього додаються двопортова пам’ять (статичне ОЗП), що знаходиться на кристалі, процесор вводу/виводу, контролер ПДП, паралельний порт системної шини та ряд інших пристроїв.

SHARC-архітектура є прикладом гармонійного поєднання принципів побудови розподілених і зв'язаних систем, що об'єднують в собі простоту і ефективність масштабування розподілених систем із зручністю програмування систем з пам'яттю, що розділяється.

Обчислювальні модулі на базі мікропроцесорів ADSP-2106x випускаються у вигляді мікропроцесорних кластерів, що містять від 3 до 8 вузлів у вигляді плат з шинним інтерфейсом ISA, PCI або VME, а також у вигляді модулів SHARCPAC і TRANSPAC – мезонінної плати, що встановлюється в спеціальні роз'єми материнських плат сигнальних процесорів.

Розроблено два ADSP SHARC-процесори: ADSP-21060 та ADSP-21062. Перший містить 4 Мегабіти ОЗП на кристалі, другий – 4 Мегабіти. У всьому іншому ці процесори однакові і є функціонально сумісними з ADSP -21020. Розширена структурна схема мікропроцесора ADSP-21060 SHARC наведена на рис. 3.18. До його складу входять: цифровий сигнальний процесор (ядро), двопортовий статичний ОЗП, процесор вводу/виводу та інтерфейсний процесор (зовнішній порт).

3.3.2. Архітектура мікропроцесорного ядра в ADSP-2106x

Мікропроцесорне ядро ADSP-2106x складається з таких блоків:

• 32-розрядного обчислювального пристрою для виконання операцій з плаваючою точкою відповідно до стандарту IEEE, який складається з помножувача, АЛП та багаторегістрового пристрою циклічного зсуву;

• блока регістрів загального призначення (файла даних);

• двох генераторів адреси: ГА-1 та ГА-2;

• контролера мікропрограм;

• кеш-пам'яті команд;

• таймера;

• комутатора шин.

ADSP-2106x SHARC містить три незалежних пристрої, що здійснюють обчислювальні операції: АЛП, помножувач з акумулятором та пристій зсуву. Обчислювальні пристрої обробляють дані у трьох форматах: 32-бітні слова з фіксованою точкою, 32-бітні слова та 40-бітні слова з плаваючою точкою. Операції над числами з плаваючою точкою є IEEE-сумісними; 32-розрядний формат з плаваючою точкою є стандартним IEEE-форматом; 40-розрядний формат з плаваючою точкою є IEEE-форматом з розширеною точністю, який має вісім додаткових бітів мантиси для забезпечення більшої точності обчислень.

АЛП виконує стандартний набір арифметичних та логічних операцій як у форматі з фіксованою точкою, так і у форматі з плаваючою точкою. Помножувач здійснює множення з плаваючою та фіксованою точками, а також сумісне множення/додавання та множення/віднімання з фіксованою точккою. Пристрій зсуву здійснює арифметичні та логічні зсуви, маніпуляції з бітами, копіювання полів, операції виділення (маскування) та нормалізацію 32-бітних операндів. Обчислювальні пристрої виконують операції за один цикл без використання конвеєра. Вихід якого-небудь обчислювального пристрою може бути входом будь-якого іншого в наступному циклі. При множенні АЛП та помножувач виконують незалежні операції одночасно.

Файл регістрів даних загального призначення використовується для пересилання даних між обчислювальними пристроями та шинами даних і для збереження проміжних результатів. Файл регістрів даних має два набори

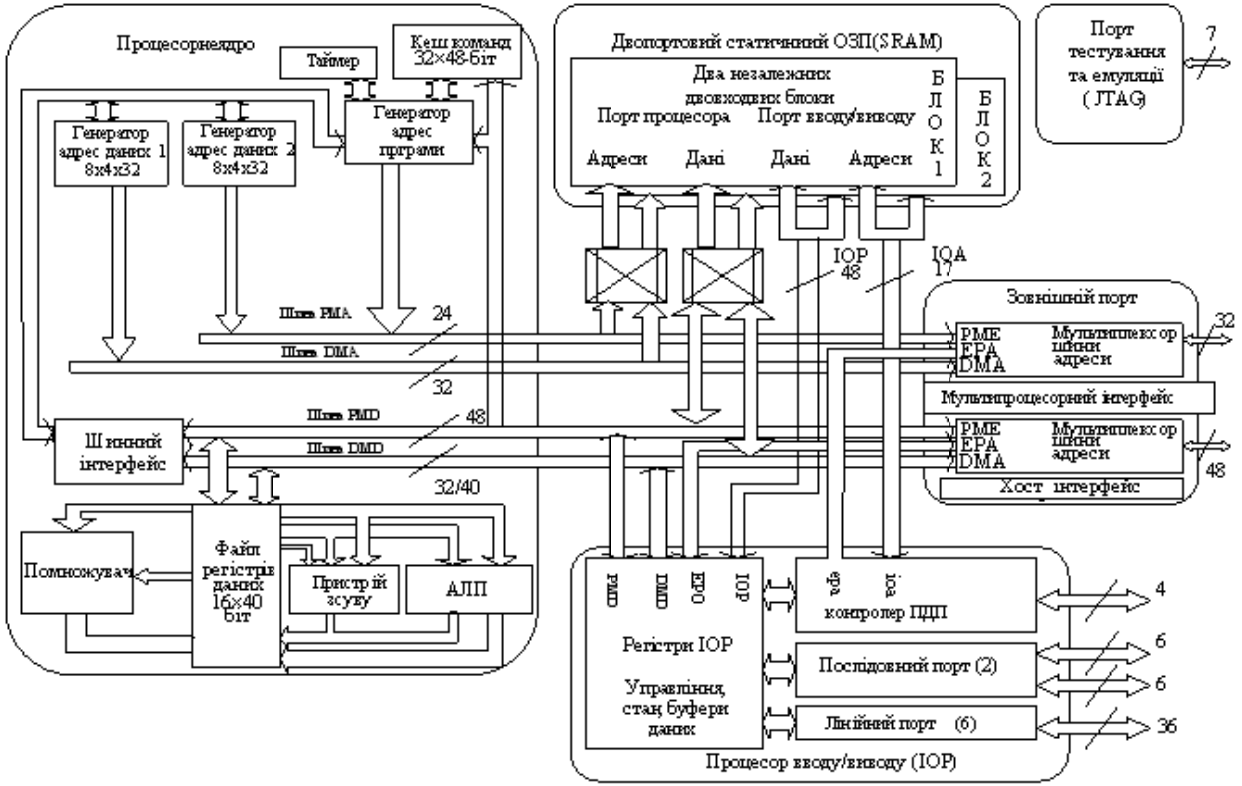

Рис. 3.18. Розширена структурна схема ADSP-21060 SHARC

регістрів (основний та додатковий), до 16 регістрів у кожному, для використання швидкого контекстного переключення при застосуванні мультизадачного режиму. Всі регістри 40-розрядні.

3.3.3. Організація управління в ADSP-2106x. Призначення й архітектура контролера мікропрограм, генератора адреси й кеша команд

Два генератора адреси (ГА-1, ГА-2) (рис. 3.18) та контролер мікропрограм (КМП) генерують адресу для доступу до пам'яті. Разом КМП та генератори адреси даних дозволяють обчислювальним операціям виконуватися з найбільшою ефективністю, тому що пристрої обчислення використовуються виключно для обробки даних. Маючи у своєму складі кеш команд, ADSP-2106x може одночасно вибирати інструкцію з кеш-пам'яті та 2 операнди з пам'яті. Генератори адреси даних апаратно реалізують циклічні буфери даних. КМП передає адресу команди до програмної пам'яті. Він управляє ітераціями та оцінює результати виконання умовних команд. Завдяки внутрішньому лічильнику циклів та стеку циклів, ADSP-2106x виконує циклічні операції з нульовими непродуктивними затратами. При цьому не потребується явних інструкцій переходу для зациклювання або декрементації та перевірки лічильника.

Висока швидкодія ADSP-2106x досягається за рахунок конвеєрного режиму роботи, який передбачає вибірку, декодування та виконання команди. Генератори адреси забезпечують формування адреси пам'яті при передачі даних між пам'яттю та регістрами. Пара генераторів адрес даних дозволяє процесору отримувати адреси для одночасного виконання двох операцій: читання або запису операндів. ГА-1 посилає 32-розрядну адресу до пам'яті даних, а ГА-2 – 24-розрядну адресу до пам'яті програм для доступу до даних програмної пам'яті. Кожен з генераторів адрес забезпечує управління 8-ма покажчиками адрес, 8-ма модифікаторами адрес та 8-ма регістрами значень довжини. Покажчик, що використовується для непрямої адресації, при необхідності модифікується значенням, розташованим у спеціальному регістрі. Модифікація може виконуватися до або після здійснення доступу до пам'яті. З кожним покажчиком може бути пов'язаний визначений обсяг даних, що адресуються. Наприклад, щоб виконати автоматичну адресацію кільцевих буферів, кожен з регістрів обох генераторів адрес має дублюючий регістр, який активується при здійсненні швидкого контекстного переключення. Кільцеві буфери звичайно використовуються в цифрових фільтрах та перетвореннях Фур'є.

Контролер мікропрограм містить кеш команд на 32 слова. До кеш-пам'яті надсилаються тільки ті інструкції, які конфліктують при доступі до

даних програмної пам’яті. Це дозволяє ядру функціонувати з повною швидкістю при циклічному виконанні багатьох операцій.

В ADSP-2106x використовуються чотири зовнішніх переривання. Три з них – загального призначення (IRQ2-0) та спеціальне переривання для скидання. Процесор також має внутрішні переривання таймера, арифметичних виключень, мультипроцесорні векторні переривання, та переривання, що визначаються програмним забезпеченням користувача. При зовнішніх перериваннях в стеку автоматично зберігаються арифметичні стани та вміст регістру режиму (MODE1). Допускаються переривання з чотирма рівнями вкладеності.

Таймер, що програмується, забезпечує періодичну генерацію переривань. Якщо переривання дозволені, таймер декрементує 32-розрядний регістр підрахунку на кожному тактовому циклі. При переході цього регістру в нульовий стан ADSP-2106x генерує переривання, активізуючи свій вихід TIMEXP. Регістр підрахунку автоматично перезавантажується з 32-розрядного регістру періоду, і підрахування поновлюється.

3.3.4. Організація ОЗП в ADSP-2106x

На кристалі ADSP-21060 розташований ОЗП ємністю 4 Мбіт, що організований у вигляді двох незалежних блоків по 2 Мбіт кожний, які можуть бути конфігуровані як для збереження даних, так і кодів. ADSP-21062 містить статичний ОЗП ємністю 2 Мбіт, розподілений на блоки по 1 Мбіт.

Кожен блок пам’яті є двопортовим (рис.3.18). Доступ до ОЗП здійснюється за один машинний цикл як для ядра процесора та процесора вводу/виводу, так і для контролера ПДП. Двопортова пам’ять та розподілені внутрішньокристалічні шини дозволяють на протязі одного машинного циклу одночасно пересилати дані з ядра та одного з портів вводу/виводу. В ADSP-21060 пам’ять може мати дві конфігурації: максимум 128К 32-розрядних слів даних або 80К 48-розрядних слів інструкцій (або 40-розрядних даних) загальним обсягом до 4 Мбіт. Вміст пам’яті може бути обрано словами по 16, 32 або 48 розрядів. Крім цього може бути забезпечений 16-розрядний формат збереження з плаваючою точкою, при котрому подвоюється кількість даних, які можуть зберігатися на кристалі. Перетворення між 32-розрядним форматом з плаваючою точкою та 16-розрядним з плаваючою точкою здійснюється за допомогою однієї команди. Незважаючи на те, що кожен блок може зберігати комбінації кодів та даних, доступ до пам’яті найбільш ефективний тільки у випадку, коли один блок зберігає дані, використовуючи для пересилання шину даних (ШД) DMD, а інший – інструкції та дані, передаючи їх по шині даних PMD. Тільки

при такому використанні шин, коли одна шина виділена одному, а інша – другому блоку, гарантується виконання двох пересилань впродовж одного машинного циклу. В цьому випадку інструкція, що виконується, повинна знаходитися у кеш-пам'яті. Виконання передачі даних протягом одного машинного циклу зберігається також в тому випадку, коли один з операндів передається у внутрішню пам'яті із зовнішнього пристрою через зовнішній порт ADSP-2106x або у зворотному напрямку.

3.3.5. Організація зовнішніх зв'язків в ADSP-2106x. Архітектура процесора вводу-виводу й портів зв'язку

Зовнішні порти забезпечують зв'язок процесора ADSP-2106x з пам'яттю, що розташована поза кристалом, та зовнішніми пристроями. При цьому в об'єднаний адресний простір може бути включено до 4Г слів адресного простору позакристалічної пам'яті. Роздільні шини всередині мікросхеми для адрес пам'яті програм і пам'яті даних (PMA та DMA) та даних пам'яті програм і пам'яті даних (PMD та DMD), адреса вводу/виводу та даних вводу/виводу мультиплексуються в інтерфейсному процесорі для створення зовнішньої системної шини з однією 32-розрядною шиною адреси (ША) та однієї 48-розрядної шини даних (ШД). Зовнішній статичний ОЗП може бути 16, 32, або 48-розрядним. При цьому інтегрований в ЦПОС контролер ПДП автоматично упаковує зовнішні дані в слово прийнятної довжини (48 розрядів для інструкцій або 32 розряди для даних).

Адресація зовнішніх пристроїв пам'яті спрощується за рахунок декодування всередині кристала старших розрядів ліній адреси та генерації сигналу вибору банка пам'яті. Для спрощення організації сторінкової адресації є можливість активізації окремих управляючих ліній процесора. ADSP-2106x дозволяє програмувати стан очікування пам'яті та контроль підтвердження зовнішньої пам'яті та зовнішніх пристроїв.

Інтерфейс головного (host) процесора дозволяє здійснювати простий зв'язок із стандартними 16- або 32-розрядними процесорами. Для цього потребується невелике додаткове програмне забезпечення. При цьому підтримується асинхронна передача із швидкістю, що відповідає максимальній частоті тактових імпульсів ЦПОС. Інтерфейс головного процесора доступний через зовнішній порт ADSP-2106x, що відображається на пам'ять в об'єднаному адресному просторі. Чотири канали ПДП підтримують пересилання кодів та даних з мінімальними програмними затратами.

Мікросхема містить засоби для роботи в багатопроцесорних системах ЦОС. Об'єднаний адресний простір дозволяє здійснювати прямий доступ до пам'яті кожного з ADSP-2106x, які включені до багатопроцесорної системи.

В мікросхему інтегрована логіка шинного арбітражу, яка забезпечує управління системою, що складається з одного провідного та до шести підпорядкованих ADSP-2106x. Зміна провідного процесора потребує тільки одного циклу. Шинний арбітраж може функціонувати на основі фіксованого або циклічного пріоритету. Дозволяється захоплення шин процесора для здійснення послідовностей читання-зміна-запис, а також забезпечується виконання векторних переривань для міжпроцесорних команд. Максимальна швидкість пересилань даних через порти зв'язку або зовнішній порт складає 240 Мбіт/с.

Процесор вводу/виводу містить два послідовних порти, шість 4-бітових лінійних портів зв'язку та контролер ПДП. Послідовні порти ADSP-2106x є синхронними та забезпечують інтерфейс з різними типами зовнішніх пристроїв. Послідовні порти функціонують з тактовою частотою процесора, що забезпечує максимальну швидкість передачі даних 40 Мбіт/с. Тактування портів може бути внутрішнім або зовнішнім. Незалежність процесів прийому та передачі забезпечує велику гнучкість послідовного зв'язку. Обмін через послідовні порти також може здійснюватися в режимі ПДП.

ADSP-2106x має шість 4-розрядних портів зв'язку, які забезпечують додаткові можливості вводу/виводу. Їх можна використовувати в мультипроцесорних системах для міжпроцесорної взаємодії за схемою двох точок (point-to-point).

Порти зв'язку можуть функціонувати паралельно з максимальною швидкістю 240 Мбіт/с. Дані портів зв'язку можуть бути упаковані у 32- або 48-розрядні слова та безпосередньо зчитані ядром процесора або передані до внутрішньокристалічної пам'яті через ПДП. Кожен порт зв'язку має власні вхідні та вихідні регістри синхронізації.

3.3.6. Система команд і програмно доступні регістри в ADSP-2106x

Набір команд ЦПОС сімейства ADSP-2106x забезпечує широкий вибір можливостей програмування. Багатофункціональні команди дозволяють здійснювати обчислення одночасно з передачею або модифікацією даних, а також одночасні операції у помножувачі та АЛП. Кожна команда виконується впродовж одного машинного циклу. Для полегшення кодування та розуміння програм асемблер процесора використовує алгебраїчний синтаксис.

Команди ЦПОС SHARC можна розподілити на чотири групи:

• команди обчислення та передачі модифікованих даних, які специфікують обчислювальну операцію з паралельною передачею одного чи двох полів даних або модифікацією індексного регістру;

• команди, що управляють виконанням програми, які специфікують різні типи розгалужень, циклів, викликів підпрограм та повернень;

• команди безпосереднього пересилання даних, в яких поля інструкцій використовуються як операнди, або безпосередньо для адресації;

• інші команди, зокрема, перевірка та модифікація окремих бітових полів, відсутність операції та ін.

В багатьох командах містяться поля специфічних операцій обчислення, які використовуються в АЛП, помножувачі або в пристрої зсуву.

Процесор містить велику групу програмно доступних регістрів. Вони є універсальними, тобто дозволяють як читати, так і модифікувати їх вміст.

Усі регістри ЦПОС ADSP-2106x можна розподілити на 6 груп [60]:

• регістри загального призначення (табл. 3. 6);

• регістри управління послідовністю команд (табл. 3. 7);

• адресні регістри (табл. 3. 8);

• регістри управління шинами (табл. 3. 9);

• регістри таймера (табл. 3. 10);

• системні регістри (табл. 3. 11).

Таблиця 3.6 – Регістри загального призначення

Позначення	Назва та призначення
R15 – R0	Регістри для операцій з фіксованою точкою
F15 – F0	Регістри для операцій з плаваючою точкою

Таблиця 3.7 – Регістри управління послідовністю команд

Позначення	Назва та призначення
PC	Програмний лічильник
PCSTK	Верхівка стека
PCSTKP	Покажчик стека
FADDR	Регістр адреси вибірки
DADDR	Адреса, що декодується
LADDR	Адреса закінчення циклу
CURLCNTR	Лічильник поточного циклу
LCNTR	Лічильник циклів для вкладених циклів

Таблиця 3.8 – Адресні регістри

Позначення	Назва та призначення
I7 – I0	Індексні регістри ГА-1
M7 – M0	Регістри модифікації ГА-1
L7 – L0	Регістри довжини ГА-1
B7 – B0	Базові регістри ГА-1
I15– I8	Індексні регістри ГА-2
M15– M8	Регістри модифікації ГА-2
L15– L8	Регістри довжини ГА-2
B15– B8	Базові регістри ГА-2

Таблиця 3.9 – Регістри управління шинами

Позначення	Назва та призначення
PX1	Обмін шин 1. Шина даних пам'яті програм - Шина даних пам'яті даних (16 біт)
PX2	Обмін шин 2. Шина даних пам'яті програм - Шина даних пам'яті даних (32 біта)
PX	48-бітова комбінація PX1 та PX2

Таблиця 3.10 – Регістри таймера

Позначення	Назва та призначення
TPERIOD	Період таймера
TCOUNT	Лічильник таймера

Таблиця 3.11 – Системні регістри

Позначення	Назва та призначення
MODE1	Управління режимом у стані 1
MODE2	Управління режимом у стані 2
IRPTL	Фіксатор переривань
IMASK	Маскування переривань
IMASP	Покажчик маски переривань (для вкладеності)
ASTAT	Прапори арифметичного стану
STKY	Прапори стану помилок
USTAT1	Регістр 1 стану користувача
USTAT2	Регістр 1 стану користувача

Для локального збереження даних використовуються регістри загального призначення (РЗП), які, за термінологією розробника, мають назву «файл регістрів». Файл регістрів (ФР) складається з 16 40-бітних основних та 16 альтернативних регістрів. Номери регістрів позначаються символом R при обчисленні з фіксованою точкою, та символом F в операціях з плаваючою точкою. АЛП може обирати два операнди (x та y) з будь-якого РЗП. Результат операції вміщується в будь-якому з регістрів файла регістрів. Залежно від результату операції встановлюються відповідні прапори регістра стану ОЗП – ASTAT.

Команди АЛП на мові асемблеру ADSP-2106x записуються таким чином:

```
Rn=Rx+Ry;          /* Сума вмісту регістрів у форматі з фіксованою точкою*/
Fn=Fx+Fy;          /* Сума вмісту регістрів у форматі з плаваючою точкою*/
Rn=(Rx+Ry)/2;      /*Визначення середнього значення*/
Rn=MIN(Rx,Ry);     /*Визначення мінімального операнда*/
Fn=RSQRTS Fx;      /*Обчислення зворотного значення квадратного кореня із вмісту Fx у форматі з плаваючою точкою */
```

```
Ra=Rx+Ry, Rs=Rx-Ry;     /*Одночасне складання та віднімання */
                        /* Rx, Fx, Ry, Fy – будь-які РЗП для операцій з */
Fa=Fx+Fy, Fs=Fx-Fy;     /*фіксованою або плаваючою точами */
```

Для виконання операцій множення з фіксованою та плаваючою точками, а також операцій множення з накопиченням застосовується апаратний помножувач, який помножує два вхідних операнди, позначених X та Y, з файла регістрів. Акумулювання результату відбувається в двох 80-розрядних регістрах (MRB та MRF), які допускають контекстне перемикання. Обидва регістри мають однакову структуру, що складається з трьох частин: MR0, MR1, MR2. Кожен з регістрів може бути незалежно зчитаний чи записаний у файл регістрів. Регістр MR2 має тільки 16 розрядів, але при читанні його вміст розширюється до 32 розрядів. Залежно від результату множення встановлюються чотири прапори регістра ASTAT та чотири прапори регістра стану помилки STKY.

На додаток до обчислень, що виконуються кожним обчислювальним пристроєм окремо, ADSP-2106x дозволяє здійснювати багатофункціональні обчислення, при яких має місце паралельне функціонування помножувача та АЛП. Дві операції виконуються таким чином, як вони виконувалися б у відповідних однофункціональних обчисленнях. Кожен з чотирьох вхідних операндів багатофункціональних команд обмежується визначеним місцем розташування у файлі регістрів. У всіх інших операціях вхідними операндами можуть бути будь-які регістри файла регістрів.

На мові асемблеру ADSP-2106x команди множення та багатофункціональні команди записуються таким чином:

```
Rn=Rx*Ry;
MRF= Rx*Ry;
MRB= Rx*Ry;
Fn=Fx*Fy;
Rn=MRF+ Rx*Ry;
Rn=MRB-Rx*Ry;
MRF=MRF-Rx*Ry;
```

Синтаксис команд є простим, тому не потребує додаткових пояснень. При виконанні однофункціональних команд в якості регістрів Rn, Rx та Ry може бути використаний будь-який регістр файла регістрів, а при багатофункціональних – тільки регістри відповідно до схеми, що зображена на рис. 3.19 [60].

Регістри управління послідовністю команд розташовані в контролері мікропрограм та є універсальними, тобто доступ до них здійснюється так як і для інших універсальних регістрів. ADSP-2106x виконує команди на протязі трьох тактів: вибірки, декодування, виконання. За рахунок конвеєризації,

поки здійснюється вибірка поточної команди, виконується декодування інструкції, обраної у попередньому циклі, та виконання команди, обраної два такти тому. Тобто, продуктивність процесора складає одну команду за один цикл. Будь-яке розгалуження програми знижує продуктивність.

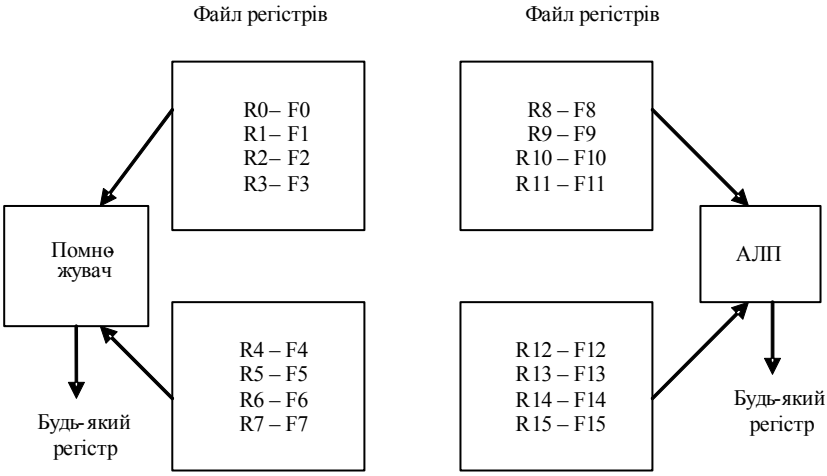

Рис. 3.19. Схема закріплення регістрів при багатофункціональних операціях

Регістр адреси вибірки FADDR, регістр адреси декодування DADDR та програмний лічильник PC містять відповідно адреси команди, що обирається, та команди, що декодується та виконується в даний момент часу. Покажчик стека програмного лічильника PCSTKP являє собою регістр, доступний для читання та запису, і містить адресу верхівки стека. При пустому стеку маємо PCSTKP=0, а PCSTKP=1, 2, …,30 за наявності в стеку даних. У випадку PCSTKP=31 виникає переповнення стеку.

Контролер мікропрограм оперує з двома окремими лічильниками циклів: лічильником поточного циклу (CURLCNTR), який відстежує ітерації поточного циклу, та зі звичайним лічильником циклів (LCNTR), який задає кількість повторів. Наявність двох лічильників необхідна для підтримки вкладених циклів. Регістр LADDR зберігає адресу верхівки стека адрес циклів. Він доступний для читання/запису через шину даних пам'яті даних. У випадку, коли стек адрес циклів пустий, він набуває значення 0xFFFFFFFF.

ADSP-2106x підтримує програмні цикли з інструкцією DO UNTIL. Ця команда здійснює повтор послідовностей інструкцій доти, поки умови, що перевіряються, не стануть істинними (true). Контролер мікропрограм аналізує для визначення моменту припинення виконання циклу шляхом перевірки

бітів регістрів стану ASTAT, управління режимом MODE1, лічильника циклів CURLCNTR та вхідних прапорів. Кожна умова, яку оцінює ADSP-2106x, має мнемоніку мови асемблеру та унікальний код, який використовується в коді операції. Для більшості умов контролер може перевіряти обидва стани: true та false. Умова LCE (закінчення підрахунку лічильника циклів) найчастіше використовується в інструкціях DO UNTIL.

Інструкція DO UNTIL забезпечує ефективний цикл без непродуктивних втрат (додаткових команд переходу, перевірки стану або зменшення лічильника). Наприклад:

```
        LCNTR=30,      DO label UNTIL LCE;
        R0=DM(10,M0), F2=PM(18,M8);
        R1=R0-R15;
label:  F4=F2+F3;
```

У процесі виконання команди DO UNTIL контролер мікропрограм заносить до стека адреси циклу адресу останньої команди та умову завершення для виходу з циклу. Він також заносить до стека програмного лічильника адресу верхівки циклу, яка являє собою адресу інструкції, що безпосередньо йде за рядком DO UNTIL.

Адресні регістри розташовані в ГА-1 та ГА-2. Обидва генератори адреси мають чотири групи регістрів по вісім однотипних регістрів кожної з груп: індексні (I), модифікації (M), бази (B) та довжини (L). У 32-розрядному ГА-1 регістри нумеруються 0 – 7, а 24-розрядному ГА-2 мають номери 8 – 15. I – регістр виконує роль покажчика на пам'ять, M – регістр містить значення приросту покажчика. Регістри B та L використовуються для адресації кільцевих буферів даних. Зокрема, в регістрі B розташована начальна (базова) адреса кільцевого буфера, а в регістрі L – довжина буфера. Кожен з регістрів ГА має альтернативний набір регістрів, який використовується при контекстному переключенні.

Пристрій зсуву працює з 32-розрядними операндами з фіксованою точкою та виконує такі функції:

• зсуви та циклічні зсуви;

• операції маніпулювання бітами;

• маніпулювання полями бітів, включаючи виділення та копіювання до зовнішньої пам'яті;

• підтримка операцій сумісності процесорів сімейства ADSP-2100 (фіксована/плаваюча точка).

Пристрій зсуву приймає від одного до трьох операндів: X – операнд, над яким проводиться дія; Y – операнд, що визначає величину зсуву s, довжини полів бітів або положення бітів; Z – операнд, над яким проводиться дія та корекція. Пристрій зсуву повертає значення результату в один з регістрів ФР.

Вхідні операнди обираються із старших 32 розрядів регістра ФР (біти 39 – 8), або як безпосереднє значення з поля інструкції. Операнди пересилаються протягом першої половини циклу. Результат передається у старші 32 розряди регістра протягм другої половини циклу, при цьому молодші вісім бітів обнуляються. Таким чином, пристій зсуву може здійснювати читання та запис того самого місця пам'яті за один машинний цикл. X та Z – це завжди 32-розрядні числа з фіксованою точкою. Вхід Y – 32- розрядне значення з фіксованою точкою або 8-бітове поле.

Наприклад, при виконанні команди FDEP, яка дозволяє маніпулювати групами бітів в межах 32-розрядного слова фіксованої довжини, вхід Y визначає два 6-розрядних значення (bit6 та len6), розташованих в Ry. При цьому bit6 – початкова позиція для виділення або депонування, а len6 – довжина поля бітів, яка визначає кількість бітів, які виділяються або депонуються. Так, при виконанні команди R0=FDEP R1 BY R2 (R1=0x000000FF00 R2=0x000002100) в регістр R0 буде завантажено фрагмент вмісту регістру R1, що складається з 8 бітів, починаючи від початку цього регістру (len6=8), та що відстоїть від початку регістра результату R0 на 16 розрядів (bit6=16). Оскільки в даному випадку обробка 32-розрядних слів здійснюється у 40-розрядних регістрах, то відлік бітів в регістрах починається з 8-го розряду (біти з 0 по 7 ігноруються та обнуляються). У результаті виконання такої операції вміст регістру R0 буде 0x00FF000000.

Для забезпечення можливостей передачі інформації між 48-розрядною шиною даних пам'яті програм та 32-розрядною шиною пам'яті даних використовується PX регістр, який складається з двох частин: 16-розрядного PX1 та 32-розрядного PX2. Вони можуть використовуватися незалежно або розглядатися як складові частини регістру PX. Регістр PX дозволяє здійснити обмін даними по шині даних пам'яті програм або по шині даних пам'яті даних з пам'яттю або регістрами ФР. Наприклад, щоб записати 48-розрядне слово до комірки пам'яті з ім'ям Port1 по шині даних пам'яті програм можна використати такі команди:

```
R0=0x9A00;/*Завантаження до R0 16-ти молодших бітів*/
R1-0x12345678;/*Завантаження до R1 32-х старших бітів*/
PX1=R0;
PX2=R1;
PM(Port1)=PX;
```

Для управління таймером застосовуються два регістри: TPERIOD та TCOUNT. Обидва регістри доступні як для читання, так і для запису. До першого з них заноситься кількість періодів підрахунку, яка після запуску таймера переноситься до регістру підрахунку TCOUNT. На кожному періоді підрахунку таймер зменшує вміст TCOUNT на одиницю. Коли вміст регістру стає таким, що дорівнює нулю, таймер генерує переривання та заносить на

вихід TIMEXP логічну одиницю тривалістю 4 цикли тактових імпульсів (якщо робота таймера не заборонена встановленням відповідного прапора регістру режимів MODE2). Далі відбувається автозавантаження числа періодів підрахунку з TPERIOD в TCOUNT.

Системні регістри призначені для задавання різноманітних режимів роботи процесора, управління операціями переривання, індикації результатів логічних та арифметичних операцій. Для управління режимом та станом процесора використовуються два регістри режиму MODE1 та MODE2. Шляхом встановлення та скидання окремих бітів цих регістрів можна змінювати режими адресації, перемикати ФР на альтернативні регістри, управляти перериваннями, дозволяти роботу таймера, визначати тип сигнального процесора та ін. [60].

Назва та призначення бітів MODE1 наведено в табл.. 3.12. Регістр MODE2 використовується для управління таймером, для заборони виходу на зовнішню шину, він блокує кеш, містить інформацію про тип процесора, що використовується, та ін.

Таблиця 3.12 – Призначення бітів регістру MODE1

Номер біта	Назва	Призначення
0	BR8	Інвертування розрядів регістру I8 (ГА-2)
1	BR0	Інвертування розрядів регістру I0 (ГА-1)
2	SRCU	Вибір альтернативного регістру обчислювача
3	SRD1H	Вибір альтернативних регістрів ГА-1 (7 - 4)
4	SRD1L	Вибір альтернативних регістрів ГА-1 (3 - 0)
5	SRD2H	Вибір альтернативних регістрів ГА-2 (15 - 12)
6	SRD2H	Вибір альтернативних регістрів ГА-2 (11 - 8)
7	SRRFH	Вибір альтернативних файлових регістрів (R15 – R8)
8-9	-	Зарезервовані
10	SRREL	Вибір альтернативних файлових регістрів (R7 – R0)
11	NESTM	Дозвіл вкладеності переривань
12	IRPTEN	Дозвіл глобальних переривань
13	ALUSAT	Дозвіл насиченості АЛП
14	SSE	Дозвіл знакового розширення короткого слова
15	TRUNC	1 – усікання з плаваючою точкою, 0 – округлення
16	RND32	1 – округлення з плаваючою точкою до 32 біт, 0 – до 40 біт
17-18	CSEL	00 – вибір основної шини
19-31	-	Зарезервовані

Регістр арифметичного стану ASTAT відображає результат виконання операції не тільки в АЛП, але й у помножувачі та пристрої зсуву.

Регістр стану помилки STKY показує додаткові ознаки стану виконання операцій або уточнює причину встановлення одного чи декількох прапорів регістру ASTAT. Прапори в регістрі STKY перевіряються після закінчення серії операцій. Якщо будь-який прапор встановлено, то деякі з

результатів є помилковими. Призначення бітів регістрів ASTAT та STKY наведено в табл.. 3.13 та 3.14 відповідно.

Таблиця 3.13 – Призначення бітів регістру ASTAT

Номер біта	Назва	Індикація стану (регістр ASTAT)
0	AZ	Вміст АЛП = 0, або втрата значності при операціях з плав. точкою
1	AV	Переповнення АЛП
2	AN	Вміст АЛП менше 0
3	AC	Без переносу АЛП з фіксованою точкою
4	AS	Знак входу X АЛП
5	AI	Неможлива операція в режимі з плаваючою точкою
6	MN	Знак множення негативний (від'ємний)
7	MV	Переповнення помножувача
8	MU	Втрата значності при операціях с плаваючою точкою
9	MI	Неможлива операція помножувача при операціях з плав. точкою
10	AF	Індикація операції АЛП з плаваючою точкою
11	SV	Переповнення пристрою зсуву
12	SZ	Вміст пристрою зсуву дорівнює 0
13	SS	Знак операнда на вході пристрою зсуву
14-17	-	Зарезервовані
18	BTF	Прапор тестування біта
19	FLG0	Значення прапора 0
20	FLG1	Значення прапора 1
21	FLG2	Значення прапора 2
22	FLG3	Значення прапора 3
23	-	Зарезервовані
24-31	CACC	Порівняння бітів акумулятора

Таблиця 3.14 – Призначення бітів регістру STKY

Номер біту	Назва	Індикація стану (регістр STKY)
1	2	3
0	AUS	Втрата значності АЛП з плаваючою точкою
1	AVS	Переповнення АЛП з плаваючою точкою
2	AOS	Переповнення АЛП з фіксованою точкою
3-4	-	Зарезервовані
5	AIS	Невірна операція в АЛП з плаваючою точкою
6	MOS	Переповнення помножувача з фіксованою точкою
7	MVS	Переповнення помножувача з плаваючою точкою
8	MUS	Втрата значності помножувача з плаваючою точкою
9	MIS	Невірна операція в помножувачі з плаваючою точкою
10-16	-	Зарезервовані
17	CB7S	Переповнення циркулярного буфера 7 ГА-1
18	CB15S	Переповнення циркулярного буфера 15 ГА-1
19-20	-	Зарезервовані
21	PCFL	Переповнення покажчика стека
22	PCEM	Покажчик стека пустий

Продовження табл. 3.14.

1	2	3
23	SSOV	Переповнення стека стану (MODE1 та ASTAT)
24	SSEM	Регістр стека пустий
25	LSOV	Переповнення стека циклів
26	LSEM	Стек циклу пустий
27-31	-	Зарезервовані

3.3.7. Розподіл адресного простору пам'яті в ADSP-2106x

Весь простір пам'яті розподіляється на три секції: простір внутрішньої пам'яті, простір мультипроцесорної пам'яті та простір зовнішньої пам'яті. Простір внутрішньої пам'яті ADSP-2106x складається з внутрішньокристалічної пам'яті та власних ресурсів. Мультипроцесорний простір пам'яті відповідає внутрішньокристалічній пам'яті та ресурсам іншого ADSP-2106x у мультипроцесорній системі. Простір зовнішньої пам'яті відповідає зовнішній пам'яті та пристроям вводу/виводу, що відображаються на пам'ять. Для звертання до пам'яті в процесорі генеруються адреси, формат яких наведено на рис. 3. 20 [60].

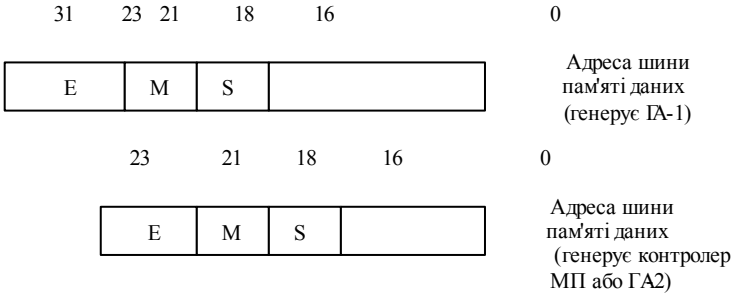

Рис. 3.20. Формат адрес процесора ADSP-2106x

Частина розрядів адресного слова виділяється для ідентифікації секції пам'яті, що використовується: E – зовнішня, M – мультипроцесорна та S – внутрішня пам'ять. Якщо поле E містить всі нулі, то поля M та S стають активними та використовуються для адресації власної внутрішньої пам'яті (M=000), або для пересилання у внутрішню пам'ять іншого процесора, що входить до складу мультипроцесорної системи. В останньому випадку поле M ідентифікує номер іншого процесора. Секції пам'яті обмежені такими адресами:

- внутрішня пам'ять від 0x00000000 до 0x0007FFF;
- мультипроцесорна пам'ять від 0x00080000 до 0x003FFFF;

• зовнішня пам'ять від 0x00400000 до 0xFFFFFFF.

Внутрішня пам'ять ADSP-2106x має три області адрес:

• регістри процесора вводу/виводу від 0x00000000 до 0x00000FF;

• адреси нормальних слів від 0x00020000 до 0x000FFFF;

(таблиця векторів переривань від 0x00020000 до 0x0002007F);

• адреси коротких слів від 0x00040000 до 0x0007FFFF.

Регістри процесора вводу/виводу являють собою 256 регістрів, що відображаються на пам'ять і контролюють конфігурації системи ADSP-2106x, а також різноманітні операції вводу/виводу. Адресний простір між регістрами вводу/виводу та адресами нормальних (за довжиною) слів (комірки від 0x00000100 до 0x0001FFF) зарезервовано, та інформація до них записуватися не повинна.

Блок пам'яті 0 починається з початку простору нормальних слів (з адреси 0x002000), а блок 1 – з середини простору нормальних слів (з адреси 0x00030000). Таким чином, для різних областей внутрішньої пам'яті використовуються такі діапазони адрес:

• 0x00000000 – 0x000000FF – регістри вводу/виводу (IOP);

• 0x00010000 – 0x0001FFFF – резервні адреси;

• 0x00020000 – 0x0002FFFF – блок 0 (адресація 32- та 48-розряднх слів);

• 0x00030000 – 0x0003FFFF – блок 1 (адресація 32- та 48-розряднх слів);

• 0x00040000 – 0x0005FFFF – блок 0 (адресація 16-розрядних слів);

• 0x00060000 – 0x0007FFFF – блок 1 (адресація 16-розрядних слів).

Простір адрес нормальних (32 та 48 розрядів) та коротких (16 розрядів) слів являє собою ту саму фізичну пам'ять. Наприклад, вибірка слова з адресою 0x00020000 – це ті ж самі комірки, що і доступ до коротких слів з адресами 0x00040000 та 0x00040001 (для 32-розрядних даних у просторі нормальних слів). Адресація коротких слів збільшує кількість 16-розрядних даних, які можуть зберігатися у внутрішній пам'яті. Вона широко застосовується, зокрема, у системах, що здійснюють операції з матрицями.

Простір мультипроцесорної пам'яті у мультипроцесорній системі відображається на внутрішню пам'ять іншого ADSP-2106x. Це дозволяє кожному з ADSP-2106x звертатися до внутрішньої пам'яті та до регістрів вводу/виводу, що відображаються на пам'ять іншого процесора. Значення поля M адресної частини визначає ідентифікаційний номер (ID2-0) зовнішнього ADSP-2106x процесора, до якого здійснюється доступ, і тому тільки цей процесор буде реагувати на цикл читання/запис. Якщо поле M=111, то відбувається циркулярний запис у всі процесори мультипроцесорної системи.

Звертання до зовнішньої пам'яті може відбуватися через зовнішній порт по шинах пам'яті програм, пам'яті даних та вводу/виводу. Цими шинами управляє ГА-1, контролер мікропрограм (ГА-2) або процесор вводу/виводу. ГА-1 та процесор вводу/виводу видають 32-розрядні адреси на шину адреси пам'яті програм та на шину адреси вводу/виводу. При цьому можливо адресувати 4Г слів пам'яті. Контролер мікропрограм та ГА-2 генерують 24-розрядні адреси по шині адреси пам'яті програм, обмежуючи адресацію молодшими 12М словами пам'яті (0x00400000 – 0x00FFFFFF).

Кожен блок внутрішньої пам'яті може бути пристосований для збереження 32-розрядних даних з одинарною точністю або для збереження 40-розрядних даних з підвищеною точністю. Ці дії ініціюються встановленням чи скиданням бітів IMDW0 та IMDW1 в регістрі системної конфігурації SYSCON.

На додаток до розгляду внутрішньокристалічної пам'яті слід зазначити, що ADSP-2106x забезпечує адресацію 4Г слів зовнішньої пам'яті через зовнішній порт. Ця пам'ять може зберігати як інструкції, так і дані. Зовнішній адресний простір включає до себе простір мультипроцесорної пам'яті, внутрішню пам'ять всіх інших ADSP-2106x, що об’єднані у мультипроцесорну систему, а також простір зовнішньої пам'яті та область для стандартної позакристалічної пам'яті.

У сигнальному процесорі використовуються безпосередня, відносна та непряма регістрова адресації. У першому випадку адреса комірки пам'яті вказується безпосередньо в полі операнда команди, наприклад

dm(0x000047E0)=astat

при виконанні цієї команди значення astat заноситься до комірки пам'яті даних з адресою 000047E0h.

При відносній адресації адреса обчислюється як сума програмного лічильника PC та зміщення, що задається в полі команди. Наприклад,

call(pc,20)

При непрямій регістровій адресації використовуються регістри ГА-1 та ГА-2. У команді вказується регістр, в якому міститься адреса необхідної комірки пам'яті. При цьому можливі два режими роботи:

• попередня модифікація (pre-modify). Ефективна адреса визначається як сума вмісту індексного регістру I та регістру M або безпосереднього операнда. При цьому вміст регістру I не змінюється;

• формування адреси з наступною модифікацією індексного регістру (post-modify). ГА видає на шину адреси вміст індексного регістру I, а потім модифікує його шляхом додавання вмісту регістру M або безпосереднього значення модифікатора.

В асемблері ADSP-2106x ці операції різняться позиціями індексу та модифікатора (M-регістру або безпосереднього значення) в інструкції.

Розташування першим в полі команди регістра I означає режим post-modify. Наприклад,

R6=PM(I15,M12)

Ця інструкція здійснює звертання до комірки програмної пам'яті з адресою, що знаходиться в I15, а потім I15+M12 записується до I15.

Якщо модифікатор записаний першим, то це свідчить про використання попередньої модифікації без операції оновлення вмісту регістру I. При виконанні команди

R6=PM(M12, I15)

обирається комірка програмної пам'яті з адресою I15+ M12, але при цьому вміст I15 не змінюється.

Зміна будь-якого регістру I здійснюється в межах одного ГА. Так, команда

DM=(M0,I2)=TPERIOD

записана правильно і дозволяє обрати комірку пам'яті даних < M0+I2>, але команда

DM=(M0,I4)=TPERIOD

помилкова, тому що M та I належать різним генераторам адреси.

Максимальне безпосереднє значення, що модифікує регістр I, залежить від типу інструкції та від того, де розташований регістр I: в ГА-1 чи ГА-2. При використанні ГА-1 максимальне безпосереднє значення може дорівнювати 2^{32}, а при використанні ГА-1 – 2^{24}.

3.3.8. Реалізація мультипроцесорних систем на базі ADSP-2106x. Архітектура й технічні засоби

Різноманітні операції ЦОС потребують дуже високої швидкості обчислень, які не можуть реалізуватися на одному процесорі навіть з максимально можливою швидкодією. У той же час багату кількість алгоритмів можливо розподілити на окремі задачі, які будуть виконуватися паралельно різними процесорами.

При розробці процесорів сімейства ADSP-2106x до його схеми були введені функціональні особливості, які дозволяють використовувати ЦПОС у багатопроцесорних системах. Ці особливості включають наявність вбудованого арбітражу доступу до провідної шини та можливість звертання до ОЗП та до регістрів процесора вводу/виводу інших ADSP-2106x.

У багатопроцесорних системах з декількома ADSP-2106x, що функціонують в режимі розподілу зовнішньої шини, будь-який процесор може стати провідним та управляти системною шиною яка, складається з 40-бітової ШД, 32-розрядної ША та групи управляючих ліній. При цьому ОЗП та регістри процесора вводу/виводу (ПВВ) всіх ADSP-2106x, що входять

до системи, створюють простір пам'яті багатопроцесорної системи. Адреса внутрішнього ОЗП та регістри ПВВ кожного процесора відображаються на цей простір. Як тільки один з ЦПОС стає провідним, він може безпосередньо працювати з ОЗП та регістрами ПВВ будь-якого підлеглого ADSP-2106x, включаючи порт FIFO-буферів даних. Провідний процесор може записувати дані до регістрів ПВВ для встановлення режиму ПДП або для генерації векторного переривання.

У багатопроцесорних системах зазвичай використовують дві схеми зв'язків між процесорами. У першій застосовують виділений канал для обміну даними між ЦПОС, у другій – єдину глобальну пам'ять, доступ до якої здійснюється через глобальну шину. У випадку виділення каналу обміну процесор ADSP-2106x може бути з'єднаний з іншими процесорами через власні лінійні порти, або він може підключатися до паралельної шини, що розподіляється (так зване кластерне з'єднання).

Ефективність багатопроцесорної системи суттєво залежить від схеми обміну інформацією між процесорами, яка визначає втрати на зв'язки між процесорами та швидкість передачі даних. Наявність у ADSP-2106x вбудованих апаратних засобів дозволяє будувати багатопроцесорні системи на основі однієї з трьох топологій: потокової, кластерної або матричної [60]. Структурна схема потокової багатопроцесорної системи, де процесори з'єднані через лінійні порти, наведена на рис. 3. 21.

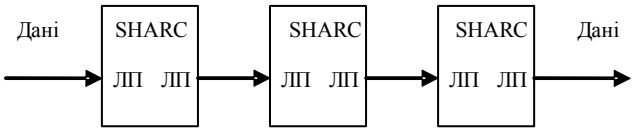

Рис. 3.21. Потокова мікропроцесорна система

Багатопроцесорна система, побудована за потоковою топологією (Data Flow Multiprocessing), складається з декількох процесорів SHARC, що об'єднані за допомогою лінійних портів. Ця топологія найбільш придатна для обробки великих міжпроцесорних потоків. У випадку використання такої схеми програміст розподіляє алгоритм обробки на декілька процесів та пропускає дані послідовно по конвеєру. ADSP-2106x SHARC ідеально підходить для систем такого типу, тому що не має необхідності у зовнішній пам'яті та буферах FIFO для зв'язків між процесорами. Ємність вбудованої пам'яті процесора достатня для розміщення кодів команд та даних більшості прикладних задач. Крім того, схема обробки проста у реалізації та має невелику вартість. Недоліком цієї топології є обмеження гнучкості системи.

Кластерна топологія (Cluster Multiprocessing) найбільш придатна для систем, які потребують максимальної гнучкості, зокрема, при підтримці

декількох задач, частина з котрих може конкурувати. Кластерні багатопроцесорні системи складаються з декількох процесорів SHARC, що з'єднуються паралельною шиною, яка дозволяє здійснювати міжпроцесорний доступ як до вбудованого у кристал ОЗП, так і до глобальної пам'яті, що розподіляється. Структурна схема цієї топології наведена на рис. 3. 22.

У типовому кластері (блоці) на процесорах ADSP-2106x може вміщуватися до шести процесорів та одного головного (хост-процесора), що здійснює арбітраж шини. Вбудована у кристал логіка дозволяє розподіляти загальну шину без додаткової апаратної частини. Запит шини генерується автоматично кожного разу, коли процесор звертається до зовнішньої адреси. Як тільки ADSP-2106x стає провідним, він може отримати доступ не тільки до зовнішньої, але і до внутрішньої пам'яті та до регістрів інших процесорів. Провідний процесор напряму переміщує дані в інший процесор або використовує для цього канал ПДП. Кластерна конфігурація дозволяє SHARC-процесорам працювати на максимальній швидкості міжпроцесорного обміну.

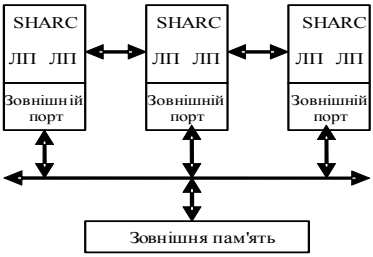

Рис. 3.22. Кластерна мікропроцесорна система

Матрична топологія – це декілька процесорів, що з'єднані, як показано на рис. 3. 23. Така топологія може бути доволі ефективною у ряді задач. Наприклад, у системах відображення даних у радіо- та гідролокації.

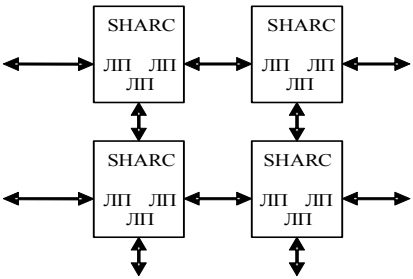

Рис. 3.23. Матрична топологія мікропроцесорної системи

На рис. 3. 23 зображена двовимірна матриця ЦПОС. Лінійні порти забезпечують зв’язок з сусідніми процесорами та маршрутизацію даних. Один провідний процесор забезпечує потік команд, який виконує масив процесорів.

Декілька сигнальних процесорів ADSP-2106x можуть сумісно використовувати зовнішню шину без додаткової схемної частини для арбітражу шин. Логіка арбітражу, вбудована до мікросхеми, дозволяє з’єднати до шести ADSP-2106x та один головний (host) комп’ютер.

Кожен з процесорів багатопроцесорної системи шляхом аналізу відповідних бітів регістру SYSTAT, може визначити, який ЦПОС є провідним у системі. Ці біти показують номер провідного процесора.

Використовуючи команди перевірки виконання умови (If condition computer), можна програмно визначити, чи є даний процесор провідним (Bus master), чи підлеглим. Мнемоніка асемблеру з визначення провідного процесора – BM (або NBM – не є провідним).

У більшості багатопроцесорних систем бажано обмежити час, протягом якого один процесор може займати системну шину. Для цього програмним способом завантажують до спеціального регістра BMAX максимальну кількість циклів мінус два, після яких процесор втрачає статус провідного. Коли в результати декремента це число стане таким, що дорівнює нулю, поточний провідний процесор дозволяє передачу управління системною шиною іншому процесору. Цикл, у якому провідний процесор передає управління шиною іншому ЦПОС, називається циклом переходу шини. Будь-який ADSP-2106x у процесі роботи у складі багатопроцесорної системи може визначити, коли настає цикл переходу шини та який процесор стає провідним після його завершення.

У багатопроцесорних систем провідний процесор може здійснювати безпосередній доступ до внутрішньої пам'яті та регістрів вводу/виводу інших процесорів. Він здійснює читання/запис комірок пам'яті, що входять до простору пам'яті мультипроцесорної системи. Такий режим доступу отримав назву “пряме читання” або “прямий запис”. Коли відбувається прямий запис у підлеглий ADSP-2106x, то дані та адреси зберігаються у 4-рівневому FIFO-буфері процесора вводу/виводу. У випадку переповнення буфера підлеглий процесор знімає сигнал підтвердження на запис, поки буфер не звільниться. Такий доступ є невидимим для підлеглих процесорів тому, що він відбувається через зовнішній порт та внутрішньокристалічну шину процесора вводу/виводу. Це дозволяє підлеглому процесору продовжувати виконання програми без перерв. Процес прямого читання не конвеєризується, тому цей режим не є найбільш ефективним способом переміщення даних з підлеглого процесора.

Для одночасного переміщення даних у всі процесори використовується широкомовний запис. Провідний процесор може здійснювати широкомовний запис в однакові області пам'яті або регістри процесора вводу/виводу у всі підлеглі процесори і в себе самого. Такий вид запису використовується для одночасного завантаження даних та програм до декількох процесорів.

Найбільш ефективним способом переміщення даних між провідним та підлеглими процесорами є прямий доступ до пам'яті (ПДП). Провідний процесор може ініціювати операції ПДП шляхом запису управляючих слів до регістрів параметрів та управління ПДП, а також встановити зовнішній порт каналів ПДП для пересилання блоків даних як із так і до внутрішньої пам'яті підлеглого процесора. Після установлення каналу ПДП провідний ЦПОС може читати/писати із відповідного буфера підлеглого процесора або може запрограмувати для цього свій власний контролер ПДП. Провідний процесор може встановити зовнішній порт ПДП для переміщення даних до зовнішньої пам'яті, вирисовуючи лінії запиту та підтвердження ПДП.

Для розподілу обчислювальних ресурсів та синхронізації у багатопроцесорній системі використовуються семафори. Обидві пам'яті, зовнішня і вбудована в ADSP-2106x, доступні з боку будь-якого процесора, тому семафори можуть бути розташовані практично де завгодно. На практиці для семафорів частіше всього використовуються регістри загального призначення процесора вводу/виводу MSGR0 – MSGR7. Використовуючи можливість блокування шини, ADSP-2106x може читати та модифікувати семафор на протязі однієї команди.

Для передачі управління у багатопроцесорній системі ADSP-2106x використовує векторні переривання. Провідний процесор видає вектор переривання підлеглому процесору шляхом запису адреси процедури обслуговування переривання до регістру VIRT підлеглого процесора. Це обумовлює переривання з високим пріоритетом, яке переключає підлеглий процесор на виконання процедури, що потрібна. Молодші 24 біта вектора містять адресу підпрограми, а старші 8 біт можуть бути додатково задіяні як дані для читання процедурою обслуговування переривання.

Контрольні питання

1. Основні принципи SHARC-архітектури.
2. Склад та призначення основних модулів мікропроцесорного ядра ADSP-2106x.
3. Призначення основних блоків управління: генераторів адрес (ГА-1 та ГА-2), контролера мікропрограм, кеша команд.
4. Організація ОЗП в ADSP-2106x

5. Організація зовнішніх зв'язків в ADSP-2106x. Призначення процесора вводу/виводу, портів зв'язку та зовнішньої системної шини.

6. Класифікація команд ЦПОС ADSP-2106x. Склад та призначення регістрів кожної групи команд.

7. Класифікація програмно доступних регістрів ЦПОС ADSP-2106x. Склад та призначення регістрів кожної групи регістрів.

8. Реалізація розподілу адресного простору пам'яті в ADSP-2106x.

9. Призначення полів E, M та S у форматі адресації ADSP-2106x.

10. Основні способи об’єднання ADSP-2106x у багатопроцесорні системи.

3.4. Система команд та особливості програмування

Системи команд різних сімейств ЦПОС мають свої особливості, але основні принципи та мнемоніка основних команд залишаються спільною. Даний розділ розглядає питання адресації та програмування ПОС на прикладі ЦПОС TMS320C2x. При виконанні будь-якої команди процесор звертається до пам'яті програм та пам'яті даних. Пам'ять програм містить команди, що виконуються процесором, а пам'ять даних – операнди, над якими здійснюються необхідні дії, що задаються полем «код операції» (КОП) команди. Спосіб визначення адреси операнда чи адреси передачі управління називають режимом адресації. TMS320C2x підтримує режими трьох видів адресації: прямої, непрямої, безпосередньої [60].

3.4.1. Режими адресації та формати команд

У режимі прямої адресації сім молодших розрядів слова команди об’єднуються з дев’ятьма розрядами регістру покажчика сторінок (DP) для отримання повної 16-розрядної адреси пам'яті даних (dma). Нульове значення сьомого біта слова команди визначає режим прямої адресації (рис. 3. 24).

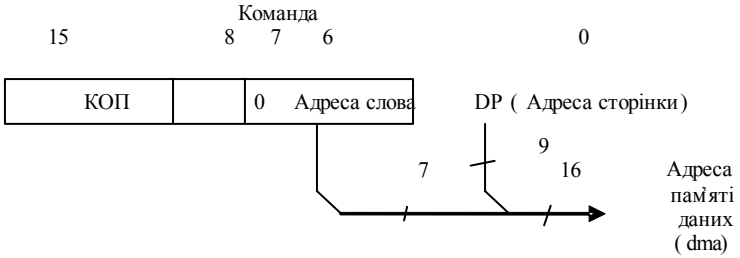

Рис. 3.24. Пряма адресація

Покажчик сторінок дозволяє обрати одну з 512 сторінок пам'яті даних. Кожна сторінка містить 128 слів. Адреса слова на сторінці визначається молодшими розрядами команди. Завантаження покажчика сторінок DP виконується командами LDP (завантажити покажчик сторінок DP) та LDPK (завантажити покажчик сторінок безпосередньо). Покажчик сторінок не ініціалізується при скиданні процесора.

У випадку непрямої адресації адреса пам'яті даних визначається вмістом одного з допоміжних регістрів AR0 – AR7. Вибір необхідного регістру здійснюється за допомогою 3-розрядного покажчика допоміжного регістру ARP, до якого вміщується число, що визначається трьома молодшими бітами (NARP) команди (рис. 3. 25). При описі команд для позначення поточного допоміжного регістру, на який посилається покажчик ARP, використовується запис AR(ARP).

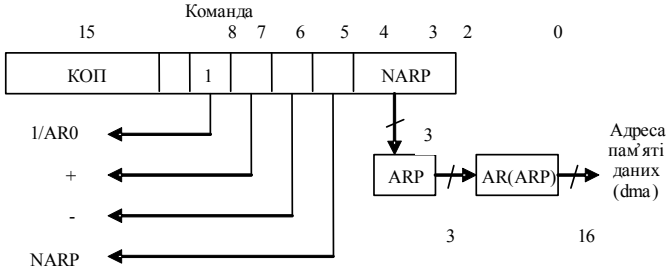

Рис. 3.25. Непряма адресація

Над вмістом регістрів AR0 – AR7 можна виконувати арифметичні операції за допомогою арифметичного пристрою допоміжних регістрів (ARAU). Зазначений пристрій виконує операції над вмістом регістрів AR0 – AR7 паралельно з операціями, що відповідають виконанню поточної команди у ЦАЛП. Наявність ARAU дозволяє організувати сім різних варіантів непрямої адресації. При запису команд з непрямою адресацією на мові асемблеру використовують спеціальні позначення, наведені в табл. 3. 15.

Таблиця 3. 15 – Режими непрямої адресації

Позначення	Назва режиму адресації
*[,NARP]	Адресація без зміни (AR)
*-[,NARP]	Адресація із зменшенням (AR) на 1
*+[,NARP]	Адресація із збільшенням (AR) на 1
*0-[,NARP]	Адресація із зменшенням (AR) на (AR0)
*0+[,NARP]	Адресація із збільшенням (AR) на (AR0)
*BR0-[,NARP]	Адресація із зменшенням (AR) на (AR0) та зворотним розповсюдженням переносу
*BR0+[,NARP]	Адресація із збільшенням (AR) на (AR0) та зворотним розповсюдженням переносу

У цій таблиці операнад NARP відповідає новому значенню покажчика ARP, що буде встановлене після виконання команди. Операнд, який міститься у квадратних дужках, не є обов'язковим. Запис (AR) означає вміст допоміжного регістру. Зменшення або збільшення значення допоміжного регістру відбувається після його використання поточною командою.

Режими адресації, засновані на зворотному розповсюдженні переносу, називають біт-інверсною адресацією та використовують при упорядкуванні даних в програмах ШПФ. Як показано на рис. 3. 25, одиниця в сьомому розряді команди означає непряму адресацію. Розряди 4, 5, 6 визначають один із різновидів непрямої адресації. Розряд 3 визначає використання поля NARP. Якщо значення цього поля дорівнює нулю, то поле NARP ігнорується та значення ARP залишається незмінним.

При безпосередній адресації операнд міститься прямо в команді. Розрізняють короткі (одне слово) (рис. 3. 26, *а*) та довгі (два слова) (рис. 3. 26, *б*) команди з безпосереднім операндом. У довгих командах у першому слові міститься код операції, а в другому – безпосередній операнд.

Рис. 3.26. Безпосередня адресація

Для запису команди будемо використовувати розширені формули Бекуса–Наура. Формат асемблерних команд процесора залежить від режиму адресації:

- <команда з прямою адресацією>::=
  [<мітка>] <мнемоніка> <ас> [,<зсув>]

- <команда з непрямою адресацією>::=

[<мітка>] <мнемоніка> <ка> [,<зсув> [,<NARP>]]

- <команда з безпосередньою адресацією>::=

[<мітка>] <мнемоніка> [<константа>]

Мітка – визначене користувачем ім'я. Значенням мітки є поточне значення лічильника команд. Мітки використовуються у програмі для передачі управління та не є обов'язковими.

Мнемоніка являє собою заздалегідь визначене ім'я, яке ідентифікує код машинної операції (КОП). Як мнемоніки використовують скорочені англійські слова або абревіатури англійських назв команд, що передають зміст основної функції команди. Наприклад, ADDH (add to high accumulator) – складання із старшими розрядами акумулятора, LAR (load auxiliary register) – завантажити допоміжний регістр.

Після мнемоніки записується операнд команди. Значеннями операндів <ас>, <зсув>, <NARP>, <константа> є цілі числа. При цьому <ас> задає адресу слова на поточній сторінці пам'яті даних. Значення <ас> лежить у діапазоні 0 – 127. Необов'язковий операнд <NARP> визначає нове значення покажчика допоміжного регістру ARP та змінюється у діапазоні 0 – 7.

Операнд <константа> – ціле 8-розрядне або 16-розрядне число. Операнд <зсув> указує кількість двійкових розрядів, на яку необхідно зсунути дане число, що бере участь у операції. Значення <зсув> залежить від команди, що виконується. Зазвичай значення поля <зсув> лежить у межах від 0 до 15.

Для команд з непрямою адресацією операнд <ка> визначається таким чином:

<ка>::=*|*+|*0+|*BR0+|*BR0-

При запису асемблерних команд операнди відокремлюються один від одного комою. Нижче наведено приклади запису команд відповідно до сформульованих правил:

```
ADD DAT1,3
ADD *,3
ADDK 5h
```

У цих прикладах ADD та ADDK є мнемоніками команд. Операнд DAT1 відповідає полю <ас> формату команди з прямою адресацією. Число 3 визначає зсув операнда, що адресується на 3 розряди вліво. Операнд 5h – 16-річна константа.

3.4.2. Класифікація команд

Систему команд ЦПОС можна розподілити на такі групи команд:

• команди акумулятора, призначені для виконання арифметичних та логічних операцій, виконання завантаження та збереження вмісту акумулятора;

• команди допоміжних регістрів, що виконують арифметичні операції над вмістом допоміжних регістрів, та операції порівняння допоміжних регістрів;

• команди T- та P-регістрів та команди множення, що призначені для обміну даними між пам'яттю та регістрами помножувача T та P. Деякі команди цієї групи розповсюджують свої дії не тільки на T- та P-регістри, але й на акумулятор, що дозволяє разом з передачею даних виконувати операції складання та віднімання акумулятора з регістрами T та P;

• команди вводу/виводу та пересилань даних, до яких входять команди пересилань блоків даних як всередині пам'яті даних, так і між пам'яттю програм та пам'яттю даних. До цієї групи команд також входять команди роботи з послідовним та паралельним портами вводу/виводу;

• команди управління, що включають команди управління стеком, кількістю повторень циклів, перериваннями. До цієї групи також входять команди завантаження та збереження регістрів стану та команди управління окремими бітами: C, OVM, HM, TC [60].

Команди акумулятора. Перелік арифметичних команд акумулятора наведено в табл. 3. 16.

Таблиця 3. 16 – Арифметичні команди акумулятора

Команда	Опис
ABS	Абсолютна величина акумулятора
ADD	Складання з акумулятором із зсувом
ADDC	Складання з акумулятором з переносом
ADDH	Складання із старшим словом акумулятора
ADDK	Складання коротке безпосереднє з акумулятором
ADDS	Складання з акумулятором без розширення знака
ADDT	Складання з акумулятором із зсувом, що визначає T-регістр
ADLK	Складання з акумулятором безпосередньої довгої константи із зсувом
SBLK	Віднімання з акумулятора безпосередньої довгої константи із зсувом
SUB	Віднімання з акумулятора із зсувом
SUBB	Віднімання з акумулятора із зсувом
SUBB	Віднімання з акумулятора з переносом
SUBC	Умовне віднімання
SUBH	Віднімання із старшого слова акумулятора
SUBK	Коротке безпосереднє віднімання з акумулятора
SUBS	Віднімання з молодшого слова акумулятора без розширення знака
SUBT	Віднімання з акумулятора із зсувом, що визначає T-регістр
NORM	Нормалізація вмісту акумулятора

Команди цієї групи виконують арифметичні дії над вмістом акумулятора та вмістом заданої комірки пам'яті даних, або над 8- чи 16-розрядною константою. При цьому може враховуватися біт переносу та виконуватися зсув вмісту заданої комірки пам'яті. Зсув вказується безпосередньо у команді, або визначається вмістом Т-регістру.

Для кожної команди вказується формат та операція, що виконується, яка умовно описується у вигляді деякого виразу, а також у текстовій формі. Нижче наводяться формати деяких команд та приклади їх використання.

Тут і далі, якщо адреса чи ім’я вказані у круглих дужках (), це означає вміст області пам'яті, на яку посилається ім’я чи адреса. Наприклад, запис (AR(ARP)) означає вміст пам'яті, на яку посилається поточний допоміжний регістр, а запис (ACC) – вміст акумулятора. У прикладах лівий стовпчик відображає стан регістрів та пам'яті процесора до виконання команди, а правий – після виконання.

Команда ADD (add to accumulator with shift) – виконує складання з акумулятором із зсувом.

Формат команди:

ADD <ас> [,<зсув>]

ADD <ка> [,<зсув> [,<NARP>]]

Операція реалізує дії (ACC)+(dma)*$2^{\text{зсув}}$ → ACC. Вміст пам'яті даних, що визначається адресою в команді, зсувається вліво на кількість розрядів, що відповідає полю <зсув>, а потім додається до вмісту акумулятора. Результат поміщається до акумулятора.

Приклад:

(AR(ARP)=1025		(1025)=4h
(1025)=4h	ADD *,3	(ACC)=28h
(ACC)=8h		(C)=0

Вміст комірки пам'яті за адресою 1025 (4h) зсувається на 3 розряди (дорівнює 20h) та складається із вмістом акумулятора (20h+8h=28h).

Команда ADDC – (add to accumulator with carry) – виконує складання з акумулятором із урахуванням переносу.

Формат команди:

ADDC <ас>

ADDC <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma)+(C) → ACC. Команда використовується при виконанні арифметичних операцій з підвищеною розрядністю.

Приклад:

(AR(ARP)=1025	ADDC *,3	(1025)=4h
(1025)=4h		(ACC)=29h
(C)=1		(C)=0

Команда ADDH – (add to high accumulator) – виконує складання із старшим словом акумулятора.

Формат команди:

ADDH <ас>

ADDH <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma)*2^{16} → ACC. Вміст пам'яті складається із старшим словом акумулятора (розряди 31..16). Значення молодшого слова не змінюється. Біт С може бути тільки встановлений даною командою, але не скинутий.

Приклад:

(AR(ARP)=1025 ADDH *	(1025)=4h
(1025)=4h	(ACC)=40008h
(ACC)=8h	(C)=1
(C)=1	

Команда ADDK – (add to accumulator short immediate) – виконує коротке безпосереднє складання з акумулятором.

Формат команди:

ADDK <константа>

Операція реалізує дії (ACC)+8-розрядна константа → ACC.

Команда ADDS – (add to accumulator with sign-extension suppressed) – виконує складання з акумулятором без розширення знаку.

Формат команди:

ADDS <ас>

ADDS <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma) → ACC, де (dma) – 16-розрядне число без знака. Акумулятор розглядається як число із знаком, вміст за адресою dma – як число без знака.

Команда ADDT – (add to accumulator with shift specified by T-register) – виконує складання з акумулятором із зсувом, що визначається Т-регістром.

Формат команди:

ADDT <ас>

ADDT <ка> [,<NARP>]

Операція реалізує дії (ACC)+(dma)*$2^{Treg(3..0)}$ →ACC. Вміст пам'яті за адресою dma зсувається вліво на кількість розрядів, що визначається чотирма молодшими розрядами Т-регістру, а потім додається до вмісту акумулятора. Команда аналогічна команді ADD.

Команда ADLK – (add to accumulator long immediate with shift) – виконує довге безпосереднє складання з акумулятором із зсувом.

Формат команди:

ADLK <константа> [,<зсув>]

Тут <константа> – це 16-розрядне число. При виконанні команди константа зсувається вліво а потім додається до вмісту акумулятора. Довжина команди 2 слова.

Приклад:

(ACC)=13AFh	ADLK 7,8	(ACC)=22FAh
		C=0

До групи арифметичних команд входять операції віднімання (табл. 3. 16), що аналогічні операціям додавання. Їх відмінність від розглянутих вище команд полягає в тому, що замість операції додавання виконується операція віднімання.

Приклади використання цих команд:

(AR(ARP)=1041	SUB *	(1041)=21h
(1041)=21h		(ACC)=24h
(ACC)=45h		(C)=1
(ACC)=48h	SUBK 11h	(ACC)=37h
		(C)=1

Одна з команд віднімання не має аналогів серед команд додавання та називається умовним відніманням:

SUBC (conditional subtract)

SUBC <ac>

SUBC <ка> [,<NARP>]

Команда SUBC виконує умовне віднімання, що використовується при виконанні операцій ділення. 16-розрядне ділиме поміщається до молодшого слова акумулятора, а старше слово обнуляється. Дільником є вміст визначеної комірки пам'яті даних. Команда SUBC виконується 16 разів при 16-розрядному діленні. Після виконання команди SUBC результат ділення (ціла частина) буде знаходитися у молодшому слові акумулятора, а залишок – у старшому. Команда дозволяє отримати коректні результати, якщо числа, що діляться, є додатними. Якщо ділиме вміщує менш ніж 16 розрядів, то його можна попередньо зсунути вліво. При цьому кількість повторів команди SUBC скорочується на кількість зсувів.

Приклад:

(AR(ARP)=1025	PRTK 15	(1025)=6h
(1025)=6h	SUBC *	(ACC)=2000Eh
(ACC)=56h		(C)=1

У цьому прикладі команда PRTK забезпечує повторне виконання команди PRTK 16 разів.

Команда ABS обчислює абсолютну величину вмісту акумулятора. Виконується операція |(ACC)| → ACC. Якщо вміст акумулятора більше або дорівнює 0, то його значення залишається незмінним. Якщо вміст

акумулятора менше 0, то до акумулятора поміщається його двійкове доповнення.

Приклади:

(ACC)=2341h	ABS	(ACC)=2341h C=0
(ACC)=AFFF FFFFh	ABS	(ACC)=5000 0001h C=0

Команда NORM (normalize contents of accumulator) нормалізує вміст акумулятора.

Формат команди:

NORM <ка>

Нормалізація числа з фіксованою точкою потребує виділення в ньому мантиси та порядку. Для визначення порядку необхідно визначити кількість бітів розширення знака. Ці дії всередині команди NORM реалізуються за допомогою операції (ACC(31)XOR(ACC(30)). Якщо розряди акумулятора 30 та 31 однакові, то обидва вони є знаковими, У цьому випадку вміст акумулятора зсувається вліво для того, щоб вилучити зайвий знаковий біт. Потім модифікується значення допоміжного регістру відповідно до обраного режиму непрямої адресації для того, щоб отримати в ньому значення порядку. Зазвичай воно інкрементується.

Для повної нормалізації 32-розрядного значення може знадобиться багатократне виконання команди NORM.

Перелік логічних операцій акумулятора наведено в табл.. 3. 17. Ця група команд реалізує порозрядні операції булевої алгебри над вмістом акумулятора та вмістом комірки пам'яті даних, або константою, що задана безпосередньо в команді.

Таблиця 3. 17 – Логічні команди акумулятора

Команда	Опис
AND	Логічне "І" з акумулятором
ANDK	Логічне безпосереднє "І" з акумулятором із зсувом
NEG	Логічне "НІ" акумулятора
OR	Логічне "АБО" з акумулятором
ORK	Логічне безпосереднє "АБО" з акумулятором із зсувом
XOR	Виключне "АБО" з акумулятором
XORK	Виключне "АБО" акумулятора і безпосередньої константи. із зсувом
CMPL	Доповнення (інверсія) акумулятора
ROL	Циклічний зсув акумулятора вліво
ROR	Циклічний зсув акумулятора вправо
SFL	Зсув акумулятора вліво
SFR	Зсув акумулятора вправо

Команда NEG (negative accumulator) заміщає значення акумулятора на його двійкове доповнення. Виконується операція (ACC) → ACC.

Приклад:

(ACC)=AFFF FF34h NEG (ACC)=5000 00CCh
C=0

Команда AND (and with accumulator) – виконує логічне «І» з акумулятором.

Формат команди:

AND <ас>

AND <ка> [,<NARP>]

Команда реалізує дії (ACC(15..0))AND(dma) → ACC.

Приклад:

DAT1=16 AND DAT1 (DAT1)=FFh
(DP)=4 (ACC)=0000 0078h
(DAT1)=FFh (C)=1
(ACC)=5432 1678h

Команда ANDK (and immediate with accumulator with shift) – виконує безпосереднє логічне «І» з акумулятором із зсувом.

Формат команди:

ANDK <константа> [,<зсув>]

Команда реалізує дії (ACC(30..0))AND константа*$2^{зсув}$) → ACC(30..0), 0→ ACC(31); 16-розрядна константа, що безпосередньо задана у команді, зсувається вліво на кількість розрядів, що визначена полем команди <зсув>. Над константою та вмістом акумулятора виконується логічна операція «І». Біти, розташовані поза межами, що визначаються даною константою, вважаються нульовими і тому відповідні розряди після виконання команди стають такими, що дорівнюють нулю. Старший розряд акумулятора дорівнює нулю завжди. Довжина команди два слова.

Приклад:

(ACC)=5423 1678h ANDK FFFFh,8 (ACC)=0032 1600h

Команди OR та ORK реалізують операції логічного «АБО» з акумулятором та безпосереднього логічного «АБО» з акумулятором із зсувом відповідно. Команди XOR та XORK реалізують операції виключного «АБО» з акумулятором та безпосереднього виключного «АБО» з акумулятором із зсувом відповідно. Формати команд аналогічні форматам команд AND та ANDK, де в тексті команди замість мнемоніки AND використовується мнемоніка OR чи XOR, а замість ANDK – мнемоніка ORK чи XORK відповідно.

Команда CMPL (complement accumulator) замінює вміст акумулятора його порозрядною логічною інверсією.

Команди ROL (rotate accumulator left) та ROR (rotate accumulator right) виконують циклічний зсув вмісту акумулятора на один біт вліво та вправо відповідно. При цьому виконуються такі операції:

• при зсуві вліво – операції ACC(31) → C; (ACC(30..0)) → ACC(31..1); (C) → ACC(0);

• при зсуві вправо – операції ACC(0) → C; (ACC(31..1)) → ACC(30..0); (C) → ACC(31).

Команди SFL (shift accumulator left) та SFR (shift accumulator right) виконують зсув вмісту акумулятора на один біт вліво та вправо відповідно. При цьому виконуються такі операції:

• при зсуві вліво – операції ACC(30) → C; (ACC(30..0)) → ACC(31..1); 0 → ACC(0);

• при зсуві вправо – операції ACC(0) → C; (ACC(31..1)) → ACC(30..0). Якщо біт режиму SXM=0, то 0 → ACC(31), інакше – (ACC(31)) → ACC(31).

У табл.. 3. 18 наведено перелік команд, що використовуються для завантаження та збереження вмісту акумулятора. Команди цієї групи заносять до акумулятора значення, що вилучається з пам'яті даних, або задається безпосередньо у команді, а також зберігають значення акумулятора у пам'яті даних.

Таблиця 3. 18 – Команди завантаження та збереження вмісту акумулятора

Команда	Опис
LAC	Завантажити акумулятор із зсувом
LACK	Завантажити акумулятор безпосередньо (коротка константа)
LALK	Завантажити акумулятор безпосередньо (довга константа)
LACT	Завантажити акумулятор із зсувом, що визначається Т-регістром
SACH	Зберегти старше слово акумулятора із зсувом
SACL	Зберегти молодше слово акумулятора із зсувом
ZAC	Обнулити акумулятор
ZALH	Обнулити молодше слово акумулятора і завантажити число в старше слово акумулятора
ZALR	Обнулити молодше слово акумулятора і завантажити число в старше слово акумулятора з обертанням
ZALS	Обнулити акумулятор, завантажити молодше слово акумулятора без розширення знака

Команда LAC (load accumulator with shift) дозволяє завантажити акумулятор із зсувом.

Формат команди:

LAC <ас> [,<зсув>]

LAC <ка> [,<зсув> [,<NARP>]]

Виконується операція (dma)*$2^{зсув}$ → ACC. Вміст пам'яті, що адресується, зсувається вліво та вміщується до акумулятора. При зсуві молодші розряди заповнюються нулями, а старші – знаковим бітом (якщо біт режиму SXM=1) або нулями якщо (SXM=0).

Приклад:

(AR(ARP))=1027	LAC *,4,5	(1027)=3h
(1027)=3h		(ACC)=30h
(ACC)=5432 1678h		(ARP)=5

Команда LACK (load accumulator immediate short) поміщає коротку (8-розрядну) константу до акумулятора. Старші 24 розряди акумулятора обнуляються. Виконується операція 8-розрядна константа → ACC.

Формат команди:

LACK <константа>

Команда LACT (load accumulator with shift specified by T register) забезпечує завантаження акумулятора із зсувом, що визначається Т-регістром.

Формат команди:

LACT <ас>

LACT <ка> [,<NARP>]

Виконується операція (dma)*$2^{Treg(3..0)}$ → ACC. Акумулятор завантажується значенням, зчитаним з пам'яті даних та зсунутим вліво на кількість розрядів, що визначаються чотирма молодшими розрядами Е-регістру. Якщо біт режиму SXM=1, то значення, що завантажується до акумулятора, доповнюється зліва до необхідної кількості розрядів знаковим бітом.

Приклад:

(AR(ARP))=1300 LACT * (1300)=276h
(1300)=276h (ACC)=2760h
(ACC)=8432 1AB5h (T)=2014h
(T)=2014h

Команда LALK (load accumulator long immediate with shift) виконує безпосереднє завантаження акумулятора довгою константою (16 розрядів) із зсувом.

Формат команди:

LALK <константа> [,<зсув>]

Зсунуте вліво значення 16-розрядної константи поміщається до акумулятора. Кількість розрядів, на яку зсувається константа, визначається полем <зсув>. Виконується операція (16-розрядна константа)*$2^{зсув}$ → ACC. Якщо біт SXM=1, то значення, що завантажується до акумулятора, доповнюється зліва до необхідної кількості розрядів знаковим бітом. Значення старшого біта акумулятора може бути встановлене у 1 тільки у режимі SXM=1. Тоді до акумулятора може бути завантажено значення з діапазону -32768..32767. Якщо SXM=0, то це значення належить діапазону 0..65535. Довжина команди два слова.

Приклади:

SXM=1 LALK F634h,4 (ACC)=FFFF 6340h
(ACC)=5432 6781h
SXM=0 LALK F634h,4 (ACC)=000F 6340h
(ACC)=5432 6781h

Команда ZAC (zero accumulator) обнуляє акумулятор. Виконується операція 0 → ACC. Дана команда відповідає команді LACK 0.

Команда ZALH (zero low accumulator and load high accumulator) дозволяє обнулити молодше слово акумулятора та завантажити старше слово значенням, яке вилучається з пам'яті даних, що адресується.

Приклад:

(AR(ARP))=4033 ZALH * (4033)=2A01h
(4033)=2A01h (ACC)=2A01 0000h
(ACC)=FFFF FFFFh

Команда ZALR (zero low accumulator, load high accumulator with rounding) дозволяє обнулити молодше слово акумулятора та завантажити старше слово з округленням.

Формат команди:

ZALR <ac>
ZALR <ка> [,<NARP>]

Виконуються операції: 8000h → ACC; (dma) → ACC(31..16).

Приклад:

(AR(ARP))=4033 ZALR * (4033)=2A01h
(4033)=2A01h (ACC)= 2A01 8000h
(ACC)=AFAF AFAFh

Команда ZALS (zero accumulator, load low accumulator with sign extension suppressed) дозволяє обнулити акумулятор та завантажити молодше слово без розширення знака.

Формат команди:

ZALS <ac>
ZALS <ка> [,<NARP>]

Виконуються операції 0 → ACC(31..16); (dma) → ACC(15..0). Значення з пам'яті даних завантажується до молодшого слова акумулятора. Старше слова акумулятора обнуляється.

Приклад:

(AR(ARP))=4033 ZALS * (4033)=2A01h
(4033)=2A01h (ACC)=0000 2A01h
(ACC)=FAFA FAFAh

Команди додаткових (допоміжних) регістрів. Ця група команд виконує арифметичні операції над вмістом допоміжних регістрів та операції порівняння допоміжних регістрів. Перелік команд цієї групи наведено у табл. 3.19. Команди можна розподілити на 2 підгрупи: арифметичні та завантаження і збереження вмісту регістрів.

Таблиця 3. 19 – Команди додаткових (допоміжних) регістрів

Команда	Опис
ADRK	Складання коротке безпосереднє із допоміжним регістром
CMPR	Порівняння допоміжного регістра з допоміжним регістром ARO
LAR	Завантаження допоміжного регістру
LARK	Завантаження допоміжного регістру безпосереднє, коротке
LARP	Завантаження покажчика допоміжного регістру
LDP	Завантаження покажчика сторінок пам'яті даних
LDPK	Завантаження покажчика сторінок пам'яті даних безпосереднє
LRLK	Завантаження допоміжного регістру безпосереднє, довге
MAR	Модифікація допоміжного регістру
SAR	Збереження допоміжного регістру
SBRK	Віднімання коротке безпосереднє із допоміжного регістра
BLKD	Переміщення блока з пам'яті даних в пам'ять даних

Команда ADRK (add to auxiliary register short immediate) здійснює коротке безпосереднє складання допоміжного регістру.

Формат команди:

ADRK <константа>

Виконується операція (AR(ARP)) + 8-розрядна константа → (AR(ARP)). Вміст поточного допоміжного регістру складається з 8-розрядною константою. Результат поміщається до поточного допоміжного регістру.

Приклад:

(AR(ARP))=2341h ADRK 40h (AR(ARP))=2381h

Команда SBRK (subtract from auxiliary register short immediate) здійснює коротке безпосереднє віднімання допоміжного регістру.

Формат команди:

SBRK <константа>

Виконується операція (AR(ARP)) - 8-розрядна константа → (AR(ARP)). З вмісту поточного допоміжного регістру віднімається 8-розрядна константа. Результат поміщається до поточного допоміжного регістру.

Приклад:

(AR(ARP))=0h SBRK 03Fh (AR(ARP))=FFC1h

Команда MAR (modify auxiliary register) виконує модифікацію вмісту допоміжного регістру.

Формат команди:

MAR <ас>

MAR <ка> [,<NARP>]

У випадку прямої адресації команда відповідає пустій операції. У випадку непрямої адресації здійснюється модифікація допоміжного регістру

та покажчика ARP. Ніяких операцій з пам'яттю не виконується. Якщо у команді задано значення NARP, то старий вміст ARP копіюється до поля ARB регістру стану ST1. Слід відзначити, що дії, які виконуються командою MAR, можна також виконати за допомогою будь-якої команди з безпосередньою адресацією.

Приклад:

(ARP)=1h	MAR *+,4	(AR1)=38h
(AR1)=37h		(ARP)=4

Команда CMPR (Compare auxiliary register with auxiliary register AR0) порівнює допоміжний регістр з допоміжним регістром AR0.

Формат команди:

CMPR <константа>

Виконується операція if ARn <константа> AR0 then 1 → TC else 0 → TC. Команда виконує порівняння вмісту поточного допоміжного регістру з вмістом допоміжного регістру AR0. Вид операції порівняння залежить від константи, яка може набувати значення з діапазону 0..3. При цьому має місце наступна відповідність між значенням константи та видом операції:

- 0 – (ARn=AR0);
- 1 – (ARn<AR0);
- 2 – (ARn>AR0);
- 3 – (ARn≠AR0).

Якщо результат порівняння буде мати значення «істина», то встановлюється біт TC регістру стану ST1.

Приклад:

(AR0)=00F7h	CMPR 1	(AR0)=00F7h
(AR(ARP))=00A1h		(AR(ARP))=00A1h
(TC)=0h		(TC)=1h

Команди T- та P-регістрів та команди множення. Перелік команд T- та P-регістрів наведено у табл.. 3. 20.

Таблиця 3. 20 – Команди T- та P-регістрів

Команда	Опис
1	2
APAC	Додати P-регістр до акумулятора
LPH	Завантаження старших розрядів P-регістру
LT	Завантаження T-регістру
LTA	Завантаження T-регістру і складання попереднього добутку
LTD	Завантаження T-регістру, складання попереднього добутку, пересилання даних
LTP	Завантаження T-регістру і збереження P-регістру в акумуляторі
LTS	Завантаження T-регістрау, віднімання попереднього добутку
PAC	Завантаження P-регістру в акумулятор
SPAC	Віднімання P-регістру з акумулятора

Продовження табл. 3.20

1	2
SPH	Збереження старшого слово Р-регістру
SPL	Збереження молодшого слово Р-регістру
SPM	Завдання режиму зсуву на виході Р-регістру

Команди цієї групи призначені для обміну даними між пам'яттю та регістрами помножувача Т та Р. Деякі з команд даної групи розповсюджують свою дію не тільки на регістри Т та Р, але й на акумулятор. Це дозволяє поряд з передачею даних виконувати операції складання та віднімання акумулятора з регістрами Т та Р. Перелік команд наведено у табл.. 3. 20.

Команда LPH (load high P register) – завантажує старше слово Р-регістра.

Формат команди:

LPH <ас>

LPH <ка> [,<NARP>]

Виконується операція (dma) → Preg(31..16). Вміст пам'яті даних за адресою dma завантажується до старшого слова Р-регістру.

Приклад:

(AR(ARP))=1025 LPH * (1025)=3A64h

(1025)=3A64h (P)=3A64 5214h

(P)=3041 5214h

Команда LT (load T register) завантажує Т-регістр.

Формат команди:

LT <ас>

LT <ка> [,<NARP>]

Виконується операція (dma) → Treg. Вміст пам'яті даних за адресою dma завантажується до Т-регістру.

Команда LTA (load T register and accumulate previous product) завантажує Т-регістр та додає попередній добуток.

Формат команди:

LTA <ас>

LTA <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (ACC)+(Preg)*$2^{\text{зсув}}$ → (ACC). Вміст пам'яті даних за адресою dma завантажується до Т-регістру. Вміст регістру Р зсувається відповідно до значення бітів РМ регістру стану ST1 та додається до вмісту акумулятора. Якщо РМ=00, то зсув не відбувається. Якщо РМ=01 або РМ=10, то вміст регістру Р зсувається вліво відповідно на 1 або на 4 розряди, при цьому молодші біти заповнюються нулями. У випадку коли РМ=11, виконується зсув вправо на шість розрядів, старші біти заповнюються значенням знакового біта.

Зсув відбувається у процесі передачі вмісту Р-регістру до акумулятора, тому операції зсуву не впливають на вміст Р-регістру.

Зсув вліво використовується для вирівнювання значень добутків в операціях з дробовими числами. Зсув вправо на шість розрядів дозволяє виконати 128 операцій типу «множення з накопиченням» без небезпеки переповнення.

Команда LTA впливає на біти С та OV. Останній встановлюється, якщо задано режим OVM:

(AR(ARP))=768 LTA *	(768)=30h
(768)=30h	(T)=30h
(T)=4h	(P)=AFh
(P)=AFh	(ACC)=B2h
(ACC)=3h	(C)=0
PM=00	

Команда LTD (load T register, accumulate previous product and move data) завантажує Т-регістр, додає попередній добуток, переслає дані.

Формат команди:

LTD <ac>

LTD <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (ACC)+(Preg)*$2^{зсув}$ → (ACC); (dma) → (dma)+1. Дії, що виконуються командою, аналогічні діям команди LTA. Додатково виконується пересилання даних з комірки з адресою (dma) до комірки з адресою (dma)+1. Команда може використовуватися при роботі з блоками пам'яті B0, B1, B2.

Команда LTP (load T register and store P register in accumulator) завантажує Т-регістр та зберігає вміст Р-регістру в акумуляторі.

Формат команди:

LTP <ac>

LTP <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (Preg)*$2^{зсув}$ → (ACC). Вміст пам'яті (dma) завантажується до Т-регістру, вміст регістру Р пересилається до акумулятора. При цьому виконується зсув Р-регістру на кількість розрядів, що визначається полем PM.

Команда LTS (load T register, subtract previous product) завантажує Т-регістр, віднімає попередній добуток.

Формат команди:

LTS <ac>

LTS <ка> [,<NARP>]

Виконуються операції (dma) → Treg; (ACC)-(Preg)*$2^{зсув}$ → (ACC). Команда аналогічна команді LTA. Відмінність полягає в тому, що замість складання виконується віднімання.

Команда PAC (load accumulator with P register) завантажує до акумулятора вміст регістру P. Виконується операція (Preg)*$2^{зсув}$ → (ACC). Вміст регістру P зсувається на 1, 4 або 6 розрядів відповідно до значення бітів PM та поміщається до акумулятора.

Команда SPM (set P register output shift mode) встановлює режим зсуву регістру P.

Формат команди:

SPM <константа>

Виконується операція 2-розрядна константа → PM. Константа, що задається в команді, копіюється у поле PM регістру стану ST1. Поле PM визначає зсув вмісту регістру при передачі його до акумулятора.

Команди множення забезпечують виконання множення, множення з накопиченням результатів в акумуляторі, множення з накопиченням та пересиланням даних. Зазвичай помножуються аргументи, що знаходяться у пам'яті даних, або у регістрі T та пам'яті даних. Перелік команд цієї групи наведено у табл.. 3.21.

Таблиця 3. 21 – Команди множення

Команда	Опис
MAC	Множення та додавання
MACD	Множення та додавання з рухом даних
MPY	Множення
MPYA	Множення і складання з акумулятором попереднього добутку
MPYK	Множення безпосереднє
MPYS	Множення і віднімання з акумулятора попереднього добутку
MPYU	Множення без урахування знака
SQRA	Піднесення до квадрата і складання результату попереднього множення з акумулятором
SQRS	Піднесення до квадрата і віднімання результату попереднього множення з акумулятора

Команда MAC (multiply and accumulate) виконує множення з накопиченням.

Формат команди:

MAC <pma>, <ac>

MAC < pma>, <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (dma) → Treg; (pma)*(dma) → Preg. За командою MAC обчислюється добуток вмісту пам'яті даних за адресою dma та пам'яті програм за адресою pma. Команда також забезпечує накопичення попереднього добутку в акумуляторі. При цьому зсув вмісту регістру P виконується відповідно до вмісту поля PM регістру стану ST1. Адреса пам'яті програм pma, що задається у команді, лежить у діапазоні 0..65535.

Команда MAC може використовуватися сумісно з командами циклу RPT та RPTK. У цьому випадку після кожного поточного виконання

MAC-операції модифікується вміст допоміжного регістру відповідно до значення поля <ка>, а також інкрементується вміст лічильника команд: (PFC)+1 → PFC. Це дозволяє адресувати ланцюжок операндів з пам'яті програм.

Зазвичай команда MAC займає два слова та для її виконання потрібні два командних цикли. Але у випадку використання команд RPT та RPTK вона виконується за один цикл.

Нижче наведено фрагмент програми сумісного використання команд MAC та RPTK:

```
SPM 3              ; встановити зсув вправо на 6 розрядів
CNFP               ; B0 – блок пам'яті програм
LARP 3             ; поточний регістр AR3
LRLK 3,300h        ; 300h – початкова адреса B1
RPTK 255           ; повторити 256 разів
MAC FF00h,*+       ; підсумовування добутків
```

У цьому прикладі блок B0 – пам'ять програм. Як пам'ять даних використано блок B1, який адресується за допомогою регістру AR3. Виконується підсумовування добутків, що вилучаються з блоків B0 та B1. Завантаження поля PM регістру ST1 константою 3 (команда SPM 3) дозволяє виключити переповнення акумулятора, яке може виникнути в результаті операції додавання.

Команда MACD (multiply and accumulate with data move) здійснює множення з накопиченням та передачею даних.

Формат команди:

MACD <pma>, <ac>

MACD < pma>, <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (pma)*(dma) → Preg; (dma) → dma+1. Команда MACD аналогічна команді MAC та додатково забезпечує пересилання даних: (dma) → dma+1. Пересилання даних буде виконуватися, якщо як пам'ять даних використовуються блоки B0, B1, B2. В іншому випадку команда виконує ті ж дії, що і MAC.

Команда MPY (multiply) здійснює множення.

Формат команди:

MPY <ac>

MPY <ка> [,<NARP>]

Виконується операція Treg*(dma) → Preg. Вміст регістру T помножується на вміст пам'яті даних за адресою dma. Результат спрямовується до регістру P.

Команда MPYA (multiply and accumulator previous product) здійснює множення та накопичення попереднього добутку.

Формат команди:

MPYA <ac>

MPYA <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (Treg)*(dma) → Preg. Попередній добуток зсувається та додається до вмісту акумулятора. Вміст регістру T помножується на вміст пам'яті даних за адресою dma. Результат спрямовується до регістру P.

Приклад:

(AR(ARP))=737 MPYA *	(737)=6h
(737)=6h	(T)=7h
(T)=7h	(P)=2Ah
(P)=34h	(ACC)=8Bh
(ACC)=57h	(C)=0
PM=0	

Команда MPYK (multiply immediate) здійснює безпосереднє множення.

Формат команди:

MPYK <константа>

Виконується операція (Treg)*13-розрядна константа → Preg. Вміст регістру T помножується на 13-розрядну константу, значення якої лежить в діапазоні -4096..4096. Перед множенням константа вирівнюється по правій границі, старші біти заповнюються бітом знака.

Приклад:

(T)=9h	MPYK -9	(T)=9h
(P)=3Ch		(P)=FFFF FFAFh

Команда MPYS (multiply and subtract previous product) здійснює множення та віднімання попереднього добутку.

Формат команди:

MPYS <ac>

MPYS <ка> [,<NARP>]

Виконуються операції (ACC)-(Preg)*$2^{зсув}$ → ACC; (Treg)*(dma) → Preg. Команда аналогічна команді MPYA, тільки замість операції додавання виконується операція віднімання.

Команда MPYU (multiply unsigned) здійснює множення без урахування знака.

Формат команди:

MPYU <ac>

MPYU <ка> [,<NARP>]

Виконується операція: Usgn(Treg)*Usgn(dma) → Preg. Множення вмісту регістру T та вмісту пам'яті даних за адресою dma відбувається без урахування знаків співмножників. Слід додати, що помножувач перемножує 17-бітові аргументи. У випадку команди MPYU старші біти кожного із співмножників, що подаються на вхід помножувача, вважаються нульовими.

З командою MPYU не слід використовувати режим PM=3, оскільки при цьому відбувається розширення знака. Команда MPYU може використовуватися для обчислення добутку 32-розрядних аргументів.

Команда SQRA (square and accumulate previous product) підносить до квадрата та накопичує попередній добуток.

Формат команди:

SQRA <ac>

SQRA <ка> [,<NARP>]

Виконуються операції (ACC)+(Preg)*$2^{зсув}$ → ACC; (dma)*(dma) → Preg. Вміст P-регістру зсувається відповідно до вмісту PM регістру ST1 та додається до акумулятора. Потім значення (dma) завантажується до T- регістру, підноситься до квадрата та заноситься до P-регістру.

Команда SQRS (square and subtract previous product) підносить до квадрата та віднімає попередній добуток.

Формат команди:

SQRS <ac>

SQRS <ка> [,<NARP>]

Виконуються операції (ACC)-(Preg)*$2^{зсув}$ → ACC; (dma)*(dma) → Preg. Команда аналогічна SQRA, тільки замість операції додавання виконується операція віднімання.

Команди вводу/виводу, переходів, управління. Перелік команд вводу/виводу та пересилань даних наведено у табл.. 3. 22.

Таблиця 3. 22 – Команди вводу/виводу та пересилання даних

Команда	Опис
IN	Ввід даних з порту
OUT	Вивід даних у порт
RFSM	Скинути біт режиму кадрової синхронізації послідовного порту
RTXM	Скинути біт режиму передачі послідовного порту
RXF	Скинути зовнішній прапор
SFSM	Встановити біт режиму кадрової синхронізації послідовного порту
STXM	Встановити біт режиму передачі послідовного порту
SXF	Встановити зовнішній прапор
FORT	Формат регістрів послідовного порту
BLKD	Переміщення блока з пам'яті даних в пам'ять даних
BLKP	Переміщення блока з пам'яті програм в пам'ять даних
DMOV	Переміщення даних в пам'яті даних
TBLR	Читати таблицю
TBLW	Записати таблицю

Команда IN (input from port) вводить даних з порту.

Формат команди:

IN <ac>, <PA>

IN <ка>, <PA> [,<NARP>]

Виконуються дії PA → (A3-A0); 0 → (A15-A4); (D15-D0) →dma. Команда копіює 16-розрядне значення, що встановлене на шині даних D15-D0 зовнішнім пристроєм, до пам'яті даних за адресою dma. Значення поля <PA> лежить у діапазоні 0..15 і являє собою адресу порту.

Приклад:

(AR(ARP))=721 IN *+,PA15 (721)=31h
(721)=0h
(A15-A0)=Fh
(D15-D0)=31h

Команда OUT (output data to port) виводить дані у порт.

Формат команди:

OUT <ас>, <PA>

OUT <ка>, <PA> [,<NARP>]

Виконуються дії PA → (A3-A0); 0 → (A15-A4); (dma) → (D15-D0). Команда встановлює на шині адреси адресу порту PA та виводить на шину даних D15-D0 вміст пам'яті даних за адресою dma.

Команда RFSM (reset serial port frame synchronization mode) зжійснює скидання режиму кадрової синхронізації послідовного порту.

Формат команди:

RFSM

Виконується операція 0 → FSM. Команда встановлює біт FSM регістру стану ST1 в нуль. У цьому режимі не потребуються зовнішні імпульси FSR для ініціалізації операції прийому кожного слова: достатньо тільки одного імпульсу для встановлення «безперервного режиму».

Команда RTXM (reset serial port transmit mode) здійснює скидання режиму передачі послідовного порту.

Формат команди:

RTXM

Виконується операція 0 → TXM. Команда встановлює біт TXM регістру стану ST1 в нуль. У цьому випадку передача запускається, коли приходить імпульс на вхід FXM.

Команда RXF (reset external flag) здійснює скидання зовнішнього прапора.

Формат команди:

RXF

Виконується операція 0 → XF. Команда встановлює вивід XF та біт XF регістру стану ST1 в нуль.

Команда SFSM (set serial port frame synchronization mode) встановлює режим кадрової синхронізації послідовного порту.

Формат команди:

SFSM

Виконується операція 1 → FSM. Команда встановлює біт FSM регістру стану ST1 в нуль. У цьому режимі для прийому даних потребується подача зовнішніх імпульсів FSR. Якщо TXM=0, то потребується подача імпульсів на вивід FSX. Якщо TXM=1, то імпульси FSX генеруються кожного разу, коли завантажується регістр DXR. Передача ініціюється заднім фронтом цього імпульсу.

Команда SXF (set external flag) встановлює зовнішній прапор.

Формат команди:

SXF

Виконується операція 1 → XF. Команда встановлює вивід XF та біт XF регістру стану ST1 в одиницю.

Команда FORT (format serial port registers) здійснює формат регістрів послідовного порту.

Формат команди:

FORT <константа>

Виконується операція 1-бітна константа → FO. Біт стану FO завантажується 1-бітною константою. Біт FO використовується для управління форматом регістрів зсуву приймача та передавача послідовного порту. Якщо FO=0, то регістри приймача та передавача є 16-розрядними. Якщо FO=1, то регістри є 8-розрядними. Біт FO скидається в нуль при подачі сигналу скидання.

У табл. 3. 23 наведено перелік команд переходів та виклику підпрограм. Ці команди забезпечують виконання програм з розгалуженою та циклічною структурами, там де на відміну від лінійних програм, необхідно виконувати не наступну команду, а команду, що знаходиться в іншій області пам'яті.

Таблиця 3. 23 – Команди переходів та виклику підпрограм

Команда	Опис
1	2
B	Безумовний перехід
BACC	Перехід за адресою, що задана акумулятором
BBNZ	Перехід, якщо (TC)=1
BBZ	Перехід, якщо (TC)=0
BC	Перехід, якщо біт C=1
BNC	Перехід, якщо біт C=0
BGEZ	Перехід, якщо (ACC)>=0
BGZ	Перехід, якщо (ACC)>0
BLEZ	Перехід, якщо (ACC)<=0
BLZ	Перехід, якщо (ACC)<0
BNZ	Перехід, якщо (ACC)≠0
BZ	Перехід, якщо (ACC)=0
BNV	Перехід, якщо біт OV=0

Продовження табл.. 3. 23

1	2
BV	Перехід, якщо біт OV=1
BIOZ	Перехід, якщо $\overline{(BIO)}$ =0
CALA	Непрямий виклик підпрограми
CALL	Виклик підпрограми
RET	Повернення з підпрограми

Для цього у лічильник команд завантажується адреса нової комірки пам'яті програм, яка називається адресою переходу. Команди, що реалізують зазначену операцію, називаються командами передачі управління. Більшість команд даної групи займають у пам'яті два слова.

Команда B (branch unconditionally) здійснює безумовний перехід.

Формат команди:

B < pma>, [<ка> [,<NARP>]]

Виконується операція: pma → PC. Поточний допоміжний регістр ARn та покажчик ARB модифікуються так, як вказано у команді. Управління передається команді, що знаходиться за адресою pma. Значення цієї адреси лежить у діапазоні 0..65535. Команда займає у пам'яті два слова.

Команда BACC (branch on address specified be accumulator) здійснює перехід за адресою, що задається акумулятором.

Формат команди:

BACC

Виконується операція (ACC(15-0)) → PC. Управління програмою передається команді, адреса якої визначається молодшим словом акумулятора. Команда займає у пам'яті одне слово.

Команда BANZ (branch on auxiliary register not zero) здійснює перехід, якщо значення допоміжного регістру не дорівнює нулю.

Формат команди:

BANZ < pma>, [<ка> [,<NARP>]]

Виконується операція if (AR(ARP)) ≠ 0 then pma → PC else (PC) + 2 → PC. Якщо вміст поточного допоміжного регістру не дорівнює нулю, то управління передається команді, яка знаходиться за адресою pma. В іншому випадку виконується наступна команда. Якщо не задано поле <ка>, вміст поточного допоміжного регістру зменшується на одиницю, тобто виконується інструкція *-. Команду BANZ можна використовувати для програмування циклічних алгоритмів. У цьому випадку допоміжний регістр виступає як лічильник циклів.

Приклад:

(AR(ARP))=1h BANZ 40h, *- (AR(ARP))=0
(PC)=58h (PC)=40h

Команди умовного переходу, що наведені у табл. 3. 23 (BBZ, BC,…, BIOZ), мають загальний формат < pma>, [<ка> [,<NARP>]]. Вони виконують передачу управління команді, яка знаходиться за адресою pma, якщо виконується відповідна умова. В іншому випадку управління передається наступній команді. Потім модифікується вміст допоміжного регістру та покажчика ARP, якщо задані поля <ка> та <NARP>. Мнемоніка команд та відповідні умови зазначені у табл.. 3. 23.

Команда CALA (call subroutine indirect) здійснює непрямий виклик підпрограми.

Формат команди:

CALA

Виконуються операції (PC) + 2 → TOS; pma → PC. Лічильник команд збільшується на 2 та завантажується у вершину стека. Управління передається команді, що знаходиться за адресою pma.

Команда RET (return from subroutine) здійснює повернення з підпрограми.

Формат команди:

RET

Виконується операція (TOS) → PC. Вміст вершини стека копіюється у лічильник команд. Із стека виштовхується одне значення.

Перелік команд управління наведено в табл.. 3. 24.

Таблиця 3. 24 – Команди управління

Команда	Опис
1	2
B1T	Біт контролю
B1TT	Біт контролю, визначений Т-регістром
CNFD	Визначити блок B0 як пам'ять даних: 0 → CNF
CNFP	Визначити блок B0 як пам'ять програм: 1 → CNF
DINT	Блокування переривань
EINT	Дозвіл переривань
IDLE	Очікування переривань
LST	Завантаження регістру стану ST0
LST1	Завантаження регістру стану ST1
NOP	Немає операції
POP	Виштовхнути вершину стека в молодше слово акумулятора
POPD	Виштовхнути вершину стека в елемент пам'яті даних
PUSHD	Виштовхнути елемент пам'яті даних у вершину стека
PUSH	Виштовхнути молодше слово акумулятора у вершину стека
RC	Скинути біт переносу: 0 → C
RHM	Скинути режим захоплення: 0 → HM
ROVM	Скинути режим переповнення: 0 → OVM
RPT	Повторити команду стільки разів, скільки вказано в елементі пам'яті даних
RPTK	Повторити команду стільки разів, скільки вказано в безпосередній константі
RSXM	Скинути режим розширення знака: 0 → SXM

Продовження табл.. 3. 24

1	2
RTC	Скинути прапор контроль/управління: 0 → TC
SC	Встановити біт переносу: 1 → C
SHM	Встановити режим захоплення: 1 → HM
SOVM	Встановити режим переповнення: 1 → OVM
SST	Збереження регістру стану ST0
SST1	Збереження регістру стану ST1
SSXM	Встановити режим розширення знака: 1 → SXM
STC	Встановити прапор контроль/управління: 1 → TC
TRAP	Програмне переривання

До цієї групи команд входять такі команди: управління стеком; управління кількістю повторів циклів; управління перериваннями; завантаження та збереження регістрів стану; управління окремими бітами (C, OVM, HM, TC).

Для управління стеком використовуються команди: POP, POPD, PUSH, PUSHD, TRAP

Команда POP (pop top of stack to low accumulator) здійснює виштовхування вершини стека у молодше слово акумулятора.

Формат команди:

POP

Виконується операція (TOS) → ACC(15-0). Вміст вершини стека (TOS) пересилається (виштовхується) у молодше слово акумулятора. На місце, що звільнюється, пересилаються елементи з нижніх рівнів стека. Після семи пересилань всі елементи стека будуть мати те саме значення, що відповідає останньому елементу стека. Процесор на має засобів, що виявляють антипереповнення стека.

Команда POPD (pop stack to data memory) здійснює виштовхування стеку у пам'ять даних.

Формат команди:

POPD <ac>

POPD <ка> [,<NARP>]

Виконується операція (TOS) → dma. Вміст вершини стека (TOS) пересилається (виштовхується) за адресою dma. Із стека виштовхується один елемент.

Команда PUSH (push low accumulator into stack) здійснює виштовхування молодшого слова акумулятора у стек.

Формат команди:

PUSH

Виконується операція (ACC(15-0)) → TOS. Елементи стека послідовно пересилаються на рівень нижче. Вміст молодшого слова акумулятора пересилається до вершини стека, що звільняється.

Команда PUSHD (pop data memory value to stack) здійснює виштовхування значення пам'яті даних у стек.

Формат команди:

PUSHD <ас>

PUSHD <ка> [,<NARP>]

Виконується операція (dma) → TOS. Вміст пам'яті даних (dma) переміщується у вершину стека.

Команда TRAP (software interrupt) здійснює програмне переривання.

Виконуються операції (PC)+1 → TOS; 30 → PC. Команда передає управління команді, що знаходиться у комірці 30. У стек вштовхується адреса команди, що йде за командою TRAP. Це дозволяє повертатися у майбутньому за командою RET у точку переривання.

До команд управління циклами належать команди RPT, RPTK.

Команда RPT (repeat instruction as specified by data memory value) здійснює повторення відповідно до значення пам'яті даних.

Формат команди:

RPT <dma>

RPT <ка> [,<NARP>]

Виконується операція (dma(7-0)) → RPTC. Команда дозволяє занести у лічильник числа повторень RPTC значення молодших восьми бітів з пам'яті даних з адресою dma. При цьому команда, що йде за командою RPT, буде повторюватися на один раз більше, ніж кількість повторень, що задана безпосередньо числом, завантаженим у RPTC. Під час циклічного виконання команди, що йде за командою RPT, переривання будуть замасковані. Лічильник числа повторень RPTC скидається сигналом RC.

Приклад:

(AR(ARP))=1030 RPT * (RPTC)=Ah

(1030)=Ah

(RPTC)=0h

Команда RPTK (repeat instruction as specified by data immediate value) здійснює повторення відповідно до безпосередньо заданих значень.

Формат команди:

RPT K <константа>

Виконується операція 8-бітна константа → RPTC. Команда вміщує безпосередньо задану константу у лічильник числа повторень. У всьому іншому команда подібна до команди RPT.

До команд управління перериваннями входять команди: DINT, EINT, IDLE.

Команда DINT (disable interrupt) забороняє переривання.

Формат команди:

DINT

Виконується операція 1 → INTM. Команда встановлює біт режиму переривань INTM в одиницю. Після цього переривання, що маскуються, будуть заборонені. Переривання, що не маскуються, цією командою не забороняються.

Переривання також забороняються при надходженні сигналу скидання.

Команда EINT (enable interrupt) дозволяє переривання.

Формат команди:

EINT

Виконується операція 0 → INTM. Команда встановлює біт режиму переривань INTM в нуль. Після цього переривання, що маскуються, будуть дозволені.

Команда IDLE (idle until interrupt) дозволяє очікування переривання.

Формат команди:

IDLE

Виконується операція 0 → INTM. Виконання команди IDLE приводить до очікування переривання або сигналу скидання. Процесор переводиться у режим малого споживання енергії.

До групи команд завантаження та збереження регістрів станів входять команди LST, LST1, SST, SST1.

Команда LST (load status register ST0) завантажує регістр станів ST0.

Формат команди:

LST <ас>

LST <ка> [,<NARP>]

Виконується операція (dma) → ST0. Регістр станів ST0 завантажується вмістом пам'яті dma. Команда не діє на біт INTM та не здійснює завантаження ARB. Команда LST використовується для відновлення слова стану ST0 після переривань та виклику підпрограм. Виконання команди здійснює встановлення значень ARP, OV, OVM, DP. Якщо в команді задано поле NARP, воно ігнорується.

Команда LST1 (load status register ST1) завантажує регістр станів ST1.

Формат команди:

LST1 <ас>

LST1 <ка> [,<NARP>]

Виконується операція (dma) → ST1. Регістр станів ST1 завантажується вмістом пам'яті dma. Біти пам'яті даних, які завантажуються в поле ARB, також завантажуються у покажчик ARP. Якщо в команді задано поле NARP, воно ігнорується.

Команда SST (store status register ST0) зберігає регістр станів ST0.

Формат команди:

SST <ас>

SST <ка> [,<NARP>]

Виконується операція (ST0) → dma. Команда зберігає вміст регістру ST0 у пам'яті даних за адресою dma. У випадку прямої адресації регістр ST0 зберігається у нульовій сторінці незалежно від значення DP.

Команда SST1 (store status register ST1) зберігає регістр станів ST1.

Формат команди:

SST1 <ас>

SST1 <ка> [,<NARP>]

Виконується операція: (ST1) → dma. Команда аналогічна команді SST.

Команди управління окремими бітами наведено у табл. 3. 24. Окремо розглянемо 2 команди: BIT і BITT.

Команда BIT (test bit) встановлює біт контролю.

Формат команди:

BIT <ас> <двійковий код>

BIT <ка> <двійковий код> [,<NARP>]

Виконується така операція: заданий двійковим кодом біт (dma) → TC. Команда копіює заданий біт пам'яті даних у біт TC регістру станів ST1. Номер біта, який потрібно копіювати, задається полем <двійковий код>. Можливі значення цього поля 0..15.

Команда BITT (test bit specified by T register) встановлює біт контролю, що визначається Т-регістром.

Формат команди:

BITT <dma>

BITT <ка> [,<NARP>]

Виконується така операція: заданий за допомогою Treg(3-0) біт (dma) → TC. Команда аналогічна команді BIT, але номер біта, що копіюється у TC, задається чотирма молодшими бітами Т-регістру.

Для решти команд цієї групи, що управляють бітами CNF, C, HM, OVM, TC, SXM, відповідні операції мають простий формат, якій містить тільки мнемоніку команди. Операції, які здійснюються наведено у табл. 3. 24. Команди є парними – скинути/встановити.

3.4.3. Особливості програмування. Директиви асемблеру. Емуляція

Програмування на мові асемблеру передбачає запис всіх елементів програми у символічній формі. Перетворення символічних імен у машинні коди покладається на спеціальну програму, яка називається асемблером.

Програми на мові асемблеру записуються у вигляді послідовності команд, які також називаються операторами. У багатьох випадках один оператор асемблеру породжує одну машинну команду. Тому програмування

на асемблері є зручною формою запису машинних кодів і дозволяє досягти максимальної ефективності програм.

Програма-асемблер створює об'єктний файл. Розрізняють два різних формати об'єктниих файлів: COFF (common object file format) формат та TI-формат. У подальшому розглядається асемблер, що підтримує COFF-формат.

Команда асемблеру містить такі поля:

[<мітка>] <мнемоніка> <операнди> [<коментар>]

Необов'язкове поле мітки – це символічне найменування 16-розрядної адреси тієї комірки пам'яті, у якій буде розміщена помічена команда. Символічне найменування (ім'я, ідентифікатор) повинне починатися з літери і містити не більш 6 символів. Мітки використовуються як адреси переходів у командах та директивах передачі управління. Приклад використання мітки:

```
Мітка   Мнемоніка      Операнди        Коментар
LOOP    LACK           0FFFh
        …
        BGZ            LOOP
        …
        BLEZ           LOOP
```

Команди BGZ та BLEZ передають управління за однією адресою пам'яті програм, що містить команду LACK.

Поле мнемоніки містить символічне ім'я команди. Мнемоніки команд зазвичай є абревіатурами речень, що характеризують основні функції, які виконує команда. Довжина мнемоніки не перевищує 5 символів. Поле мнемоніки відокремлюється від поля мітки хоча б одним пропуском. Мнемоніки – це ключові слова, тому якщо вони записані з помилками, асемблер формує відповідне повідомлення.

Вміст поля операндів залежить від команди. Як операнди можуть виступати абсолютні та символічні адреси пам'яті, програмно доступні внутрішні регістри процесора (PRD, TIM, AR0 та ін.), адреси портів вводу/виводу (PA1-PA15), числові та символічні константи, режими непрямої адресації (*+, +0-, *BR0 та ін.). Поля операндів деяких команд можуть бути пустими (SST, CALA, RTC та ін.).

Якщо в полі операнда міститься шістнадцяткове число, воно повинне починатися з цифри та закінчуватися літерою h (hex).

Число, що починається з літери, доповнюється зліва незначущим нулем, наприклад, 0AFCh, 0FCEDh. У TI-форматі запис шістнадцяткового числа починається з символу «>», наприклад, >FF00, >A7. У іншому випадку число вважається десятковим. У полі операндів дозволяється вказувати символічні імена, які визначені у самому асемблері та які пов'язані з архітектурою процесора. У мову асемблеру вбудовані імена програмно

доступних регістрів: AR0-AR7, TIM, PRD, IMR, GREG, DRR, DXR; мітки: DATn, PRGn, що асоціюються відповідно з *n*-ю коміркою пам'яті даних та пам'яті програм.

Замість імен внутрішніх регістрів дозволяється зазначати їх адреси у десятковій або шістнадцятковій системі. Наприклад, наведені нижче команди ідентичні:

```
SALK 3h        ;(ACC) → 3h
SALK PRD       ;(ACC) → PRD
```

У командах передачі управління у полі операнда можна зазначати мітки, які введені у поля міток інших команд. Мітки, розташовані у полі операнду, є символічними адресами переходу. У процесі обробки та завантаження програми міткам ставляться у відповідність адреси. Мітки, що введені як операнди, повинні обов'язково один раз з'явитися у полі мітки деякої команди.

Приклад:

```
        LAC    0200h          ;
        SUBK   01h            ;
        BGZ    MITKA1         ; Перехід на мітку
        …
MITKA1  LAC    0300h          ;
        CALL   SUBR           ; Виклик підпрограми
        …
SUBR    SST
        SST1
```

У рядках асемблерної програми можуть використовуватися директиви, що передають вказівки програмі асемблеру про виконання визначених дій. Директиви визначають порядок дій, розміщують у пам'яті команди та дані, присвоюють чисельні значення символічним найменуванням, резервують пам'ять та ін. Нижче розглядаються спрощені формати деяких директив COFF-асемблеру.

Директива .title – директива заголовка програми. Формат директиви:

.title <рядок>

Поле <рядок> – послідовність символів (не більш 65), що взяті у подвійні лапки. Наприклад:

.title "Програма фільтрації"

Директива .set (встановити) виконує присвоєння. Формат директиви:

< ім'я > .set <вираз>

При виконанні директиви імені, що стоїть у полі мітки, присвоюється значення виразу. Зазвичай вираз задається шістнадцятковим числом. Коли ім'я зустрічається у полі операнда команди, замість нього підставляється присвоєне значення:

```
SADR   .set      00400h
EADR   .set      02000h
…
LRLK   AR0,EADR
LRLK   AR1,SADR
```

У команді LRLK замість імен SADR та EADR будуть використані значення 400h та 2000h відповідно.

Директива .bss резервує пам'ять під змінну та має такий формат:

.bss < ім’я > <константа>

Числова константа визначає кількість комірок пам'яті, що резервуються для зберігання змінної. Наприклад, запис .bss var,1 резервує одну комірку пам'яті під змінну var.

Директива .sect (секція, сегмент) визначає іменований сегмент та має формат:

.sect <рядок>

Програма може бути розбита на сегменти, які являють собою блоки коду або даних, що переміщуються. Сегменти розміщуються у пам'яті процесора. Конкретне розміщення сегментів у адресному просторі пам'яті визначається на стадії редагування зв’язків. Поле <рядок> відповідає назві сегмента. Наприклад,

```
.sect “reset”
B MAIN
```

Тут сегмент “reset” містить одну команду передачі управління на мітку MAIN, що відповідає початку програми. Абсолютна адреса, за якою буде розміщена команда, визначається при виклику редактора зв’язків.

Окрім іменованих сегментів, об’єктні файли формату COFF завжди містять три стандартних сегменти: .text, .data, .bss.

Директива .text визначає сегмент, у якому розташовується код програми. Код програми буде вміщуватися у даний сегмент, доки не буде відкритий інший сегмент за допомогою директив .sect, .data.

Директива .data відкриває сегмент, у якому зазвичай розміщуються таблиці вхідних даних та інші ініціалізовані змінні.

Існує два основних типи сегментів: ініціалізовані та неініціалізовані. Ініціалізовані сегменти містять дані або код. Сегменти .sect, .text, .data належать до ініціалізованих. Неініціалізовані сегменти резервують пам'ять у адресному просторі процесора для неініціалізованих даних. Наприклад, сегмент .bss.

Нижче наведено текст програми, що містить директиви .set, .bss, .sect, .data. Для кращого розуміння призначення директив програма містить додаткове поле адреси та поле машинного коду.

```
Адреса  Код
        0400    SADR    .set    0040h
        2000    EADR    .set    02000h
                  *
0000                    .bss    VAR,1
                  *
0000                    .sect   “RESET”
0000    FF80            B       MAIN
0001    000’
          *
0000                    .text
0000    C800    MAIN    LDRK    0               ; DP ← 0
0001    CA00            LACK    0               ; ACC ← 0
                …
0005    D000            LRLK    AR0, EADR       ; AR0 ← EADR
0006    2000
0007    D100            LRLK    AR1, SADR       ; AR1 ←  SADR
0008    0400
                …
0012    6000            SACL    VAR             ; (ACC) ← VAR
                …
                        .end
```

Друге слово машинного коду команди B містить відносну адресу переходу 000’ на мітку MAIN, яка може бути визначена тільки тоді, коли буде відомо, де розташовується сегмент .text. Абсолютна адреса розташування секцій задається користувачем на етапі редагування зв’язків.

Директива .bss також формує відносні адреси. Абсолютні адреси визначаються користувачем при редагуванні зв’язків.

Директива .end інформує програму-асемблер про досягнення фізичного кінця вхідної програми.

Однією з основних проблем програмування на мові асемблеру є те, що весь процес обчислень має бути сформульований у термінології команд процесора. Але команди процесора виконують дрібні операції у порівнянні з операціями, що потребуються для розв’язання задачі на змістовному рівні, тому програмування на асемблері досить трудомістке.

Для зменшення труднощів, пов’язаних з цим, програміст може визначати більш змістовні команди, які потім використовуються при складанні програм сумісно із звичайними командами. Такі команди називають макрокомандами. Макрокоманда вводиться за допомогою макровизначення, яке по суті близьке до визначення процедур у мовах програмування високого рівня. Формат макровизначення:

<ім’я макрокоманди> $MACRO [параметр{,параметр}]
тіло макровизначення
$ENDM

Директива $MACRO відкриває визначення макрокоманди, присвоює їй ім’я, що стоїть зліва від слова $MACRO та задає параметри макрокоманди, які не є обов’язковими. Макрокоманда викликається за допомогою свого імені та операндів. Коли макрокоманда викликана, асемблер ставить у відповідність перший операнд у виклику макрокоманди з першим операндом у визначенні макрокоманди і т.п. Директива $ENDM закінчує визначення макрокоманди. При виклику макрокоманди асемблер підставляє команди, що входять у її визначення, до коду програми.

Контрольні питання

1. Режими адресації та формати команд.
2. Класифікація команд ЦПОС.
3. Перелік та призначення основних арифметичних та логічних команд роботи з акумулятором. Особливості їх використання.
4. Перелік та призначення команд завантаження та збереження вмісту акумулятора. Особливості їх використання.
5. Перелік та призначення команд додаткових (допоміжних) регістрів. Особливості їх використання.
6. Перелік та призначення команд T- та P-регістрів. Особливості їх використання.
7. Перелік та призначення команд множення. Особливості їх використання.
8. Перелік та призначення команд вводу/виводу та пересилання даних. Особливості їх використання.
9. Перелік та призначення команд переходів та виклику підпрограм. Особливості їх використання.
10. Перелік та призначення команд управління. Особливості їх використання.
11. Формат команд асемблеру. Призначення полів команди.
12. Перелік та призначення основних директив асемблеру.
13. Призначення макрокоманд. Формат макровизначення.