І.С. Зиков, С.Г. Межерицький,
А.О. Подорожняк, І.П. Хавіна

ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ У ЗАХИЩЕНОМУ РЕЖИМІ

Навчально-методичний посібник

Харків, 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
"ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"

І. С. Зиков, С. Г. Межерицький, А. О. Подорожняк, І. П. Хавіна

ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ У ЗАХИЩЕНОМУ РЕЖИМІ

Навчально-методичний посібник
для студентів комп'ютерних спеціальностей

Затверджено редакційно-видавничою радою НТУ "ХПІ",
протокол № 1 від 30.01.2018

Харків
ТОВ «ДІСА ПЛЮС»
2018

УДК 004.42:004.45
3-60

Рецензенти:

О. О. Можаєв – д-р техн. наук, професор, Національний технічний університет "Харківський політехнічний інститут";

Ю. І. Скорін – канд. техн. наук, доцент, Харківський національний економічний університет імені Семена Кузнеця

Автори:

Зиков Ігор Семенович – канд. техн. наук, професор;

Межерицький Сергій Геннадійович – старший викладач;

Подорожняк Андрій Олексійович – канд. техн. наук, ст. наук. співроб., доцент;

Хавіна Інна Петрівна – канд. техн. наук, доцент

Затверджено редакційно-видавничою радою НТУ "ХПІ", протокол № 1 від 30.01.2018

Зиков І. С.

3-60 Програмування мікропроцесорів у захищеному режимі : навч.-метод. посібник / І. С. Зиков, С. Г. Межерицький, А. О. Подорожняк, І. П. Хавіна. – Харків : ТОВ «ДІСА ПЛЮС», 2018. – 264 с. : іл.

ISBN 978-617-7384-91-4

У даному посібнику розглядаються питання, пов'язані з програмуванням мікропроцесорів сімейства x86 та x64 в захищеному режимі: встановлювання захищеного режиму, оброблення переривань, організація мультізадачної роботи мікропроцесора та захисту пам'яті й пристроїв введення-виведення або зовнішніх пристроїв. Всі ці дії належать до функцій ядра сучасних операційних систем. Програмування проводиться на рівні регістрів мікропроцесора та системних сегментів пам'яті.

Для студентів комп'ютерних спеціальностей вищих навчальних закладів.

Іл. 13. Табл. 24. Бібліогр. 9 назв.

УДК 004.42:004.45

ISBN 978-617-7384-91-4

ЗМІСТ

ВСТУП

До найбільш видатних досягнень людства століття таких, як фундаментальні відкриття фізиків з останнього роз'яснення структури матерії й впровадженні на базі цього ядерної енергетики, відкриття біологами молекули ДНК і створення генних технологій, відкриття астрофізиків, які, пояснюють, як народжувався й розвивався наш Всесвіт та практичного освоєння космосу, без всяких сумнівів слід віднести й появу в середині XX століття електронних обчислювальних машин і швидкого розвитку комп'ютерних та інформаційних технологій.

Без комп'ютерів, які є власністю мільйонів людей, що становлять основу автоматизованих інформаційних систем підприємств, фірм, банків і т.п., на базі яких будуються всі комп'ютерні мережі, в тому числі й Internet, неможливо уявити сучасний світ.

Зараз, як й останні 25 років, комп'ютерні та інформаційні технології є тим напрямом науки й техніки, який розвивається найбільш швидко.

Основним елементом будь-якого комп'ютера є центральний процесорний пристрій (англ. CPU Central Processing Unit), який виконуючі команди (англ. instructions), здійснює оброблення даних та управління всіма пристроями комп'ютера.

Цей пристрій також називають мікропроцесором (МП). Коли фірма Intel 15 листопада 1971 року випустила перший в історії МП – Intel 4004, він мав дуже обмежені характеристики: 45 команд, шістнадцять 4-розрядних регістрів; пам'ять розміром 1 КБ даних та 4 КБ команд, вбудований 4-рівневий стек, шина адреса/дані розміром 12/4 розрядів з мультиплексуванням, продуктивність 0,06 мільйонів операцій в секунду. Саме за це дана мікросхема була названа мікропроцесором.

В наш час, коли мікропроцесори мають сотні мільйонів та мільярди тран-

зисторів, коли вони по всім технічних показниках перевершують процесори минулих років, їх називають просто процесорами. В подальшому всі ці терміни будуть застосовуватися як синоніми.

Більшість мікропроцесорів, які зараз випускаються промисловістю світу, належать до сімейств x86 та x64 (x86-64/AMD64/Intel 64/EM64T), які отримали таку назву через те, що всі вони мають базову систему команд ту ж саму, що й МП Intel 8086. Хоча система команд цих мікропроцесів постійно обновлюється, вони в цілому остаються програмно сумісними.

Основним режимом роботи мікропроцесорів x86 та x64 є захищений режим (ЗР), тому що, тільки працюючи в цьому режимі, МП може продемонструвати всі свої можливості.

У навчально-методичному посібнику розглядаються питання, пов’язані з програмуванням мікропроцесорів сімейства x86 та x64 у захищеному режимі: встановлювання ЗР, оброблення переривань, організація мультізадачної роботи мікропроцесора та захисту пам’яті й пристроїв введення-виведення (ПВВ) або, як їх ще називають, зовнішніх пристроїв (ЗП). Усі ці дії належать до функцій ядра сучасних операційних систем (ОС). Програмування проводиться на рівні регістрів мікропроцесора та системних сегментів пам’яті.

У першому розділі розглядаються питання організації захищеного режиму:

– встановлення ЗР;

– визначення сегментів пам’яті та їх параметрів;

– організація прямого доступу до відео-пам’яті для виведення повідомлень на екран монітора;

– повернення до реального режиму.

У другому розділі розглядаються питання оброблення переривань у захищеному режимі:

– розробка оброблювачів зовнішніх апаратних переривань (драйверів) для організації доступу до зовнішніх пристроїв ;

– розробка оброблювачів виключень, які виконують необхідні дії, наприклад, виводять повідомлення на екран та припиняють виконання програми, при виникненні нестандартних ситуацій, які МП не може самостійно обробити;

– розробка оброблювачів програмних переривань, які дозволяють створити більш гнучкі програмні засоби.

У третьому розділі розглядаються питання організації мультипроцесорної

роботи мікропроцесора, яка дозволяє більш ефективно використовувати процесор та інші системні ресурси й тому є основною в роботі сучасних ОС:

– створення задач;

– розроблення структур даних для цих задач;

– організація переключення задач.

Апаратна підтримка захисту пам’яті та ЗП була вперше реалізована в МП Intel 80286 (в повному об’єму з МП Intel 80386). Це дозволило створювати складні програми, що працюють надійно, прикладами яких є ОС Windows або Unix та UNIX-подібні системи.

У четвертому розділі розглядаються питання захисту пам’яті та зовнішніх пристроїв:

– створюється програма вищого рівня привілею (рівня ОС);

– створюється задача рівня користувача;

– перевіряються можливості користувача при доступі до сегментів даних, сегментів коду та пристроїв введення-виведення.

1. ОРГАНІЗАЦІЯ РОБОТИ МІКРОПРОЦЕСОРА У ЗАХИЩЕНОМУ РЕЖИМІ

1.1. Особливості захищеного режиму

Усі мікропроцесори фірми Intel, починаючи з 80286 і закінчуючи Pentium 4 та Core i7, а також програмносумісні з ними МП інших фірм (вони входять до сімейства x86) мають два основних режими роботи: захищений режим (Protected Mode) і режим реальних адрес (Real-Address Mode), або просто реальний режим (Real Mode).

Найбільш повно можливості мікропроцесорів реалізуються при роботі у захищеному режимі. А саме:

– збільшується адресний простір фізичної пам'яті з 1 МБ (точніше 1 МБ - 16 + 64 КБ) до 4 ГБ (2^{32}), а при сторінковій організації – до 64 ГБ (2^{36});

– підтримується доступ до віртуальної пам'яті (розташована на жорсткому магнітному диску) об'ємом до 64 ТБ (2^{46});

– на апаратному рівні реалізована система захисту пам'яті, що регламентує доступ до сегментів пам'яті залежно від ступеня їхньої захищеності та рівня привілеїв програм і забороняє несанкціоноване втручання в роботу операційної системи та програм користувачів;

– крім сегментації пам'яті, може бути здійснена також її сторінкова організація;

– апаратно підтримується мултізадачний режим роботи МП;

– виконується режим віртуального мікропроцесора 8086.

Після включення живлення в комп'ютері та при скиданні мікропроцесора останній починає працювати в реальному режимі.

Для організації роботи МП у захищеному режимі необхідно виконати такі дії:

1) визначити розрядність встановленого в персональному комп'ютері мікропроцесора;

2) сформувати в пам'яті глобальну дескрипторну таблицю, яка містить дескриптори всіх сегментів пам'яті, що використовуються в програмі;

3) задати базову адресу і розмір глобальної дескрипторної таблиці;

4) сформувати дані, які потрібні при поверненні в реальний режим шляхом скидання мікропроцесора;

5) заборонити масковані та немасковані апаратні переривання;

6) зберегти в пам’яті вміст регістрів МП;

7) перевести мікропроцесор до захищеного режиму;

8) виконати у захищеному режимі потрібні дії;

9) повернутися до реального режиму;

10) відновити вміст регістрів МП;

11) дозволити масковані та немасковані переривання.

1.2. Середовище виконання програм та програмне забезпечення захищеного режиму

Запустити на виконання програму, яка працювала б у захищеному режимі й мала можливість доступу до системних ресурсів, в середовищі сучасних операційних систем (Windows, Linux та ін.) неможливо – прикладні програми користувачів в цих ОС отримують мінімальний рівень привілею і позбавлені можливості виконання системних команд та доступу до всіх ресурсів процесора.

Щоб програма отримала найвищий рівень пріоритету, необхідний для повноцінної роботи у захищеному режимі, треба щоб вона сама організувала перехід з реального режиму до захищеного. Тому для виконання програми у захищеному режимі треба вибрати програмне середовище, що працює в реальному режимі, наприклад операційну систему MS DOS. Є декілька варіантів її застосування.

Можна під MS DOS виділити логічний диск (підрозділ фізичного жорсткого магнітного диску) і працювати в середовищі реальної MS DOS. Другим варіантом (який більш прийнятний) є застосування сучасної операційної системи, а за допомогою одного з пакетів віртуалізації, наприклад пакету VMware Workstation, створити віртуальну MS DOS. В обох випадках результати роботи програми у захищеному режимі будуть однакові.

Після переходу до захищеного режиму програма працює вже не під BIOS, не під MS DOS і не під Windows. Тому їй недоступні переривання та функції цих програм. Вона працює безпосередньо з апаратурою процесора і по суті є мікрооперацією системою, тому повинна самостійно виконувати такі функції ОС, як розподіл пам’яті, оброблення переривань від зовнішніх пристроїв, робота з відеопам’яттю на рівні адресації комірок пам’яті та ін.

Під час вибору мови програмування треба брати до уваги, що частини програми, які працюють у захищеному режимі, повинні бути написані на мові Assembler, тому що транслятори з мов Pascal та C під MS DOS формують значення сегментних регістрів тільки в форматі реального режиму, несумісному із захищеним режимом. При переході до роботи у захищеному режимі застосування цих значень призводить до виникнення виключень й припинення роботи програми.

Що стосується процедур та функцій, які працюють у реальному режимі й формують структури даних для роботи у захищеному режимі, то вони в прикладах, які наведені в навчально-методичному посібнику, написані не на мові Assembler, а на мові Pascal або C з використанням асемблерних вставок з метою зробити ці процедури менш громіздкими і, таким чином, зосередити увагу не на техніці програмування, а саме на структурах даних захищеного режиму. Ці процедури і функції рівноцінно можуть бути написані також на мові C.

Оскільки організація роботи МП у захищеному режимі є досить складною, для її полегшення було розроблене програмне забезпечення захищеного режиму (ПЗЗР), до складу якого входять базові програми для кожного розділу, де розглядаються особливості захищеного режиму:

– P_MODE – організація захищеного режиму (розділ 1);

– P_INT – обробка переривань у захищеному режимі (розділ 2);

– P_TASK – організація мультізадачної роботи (розділ 3);

– P_USER – організація роботи програм на рівні користувача (розділ 4);

– P_8086 – робота у віртуальному режимі 8086 (розділ 5);

– P_PAGE – робота зі сторінками (розділ 6),

а також програмний модуль **PROT**, призначений для підтримки програм, що працюють у захищеному режимі, який:

а) містить необхідні константи, типи змінних, змінні, процедури й функції;

б) створює базову глобальну дескрипторну таблицю й базову дескрипторну таблицю переривань (опис і текст модуля **PROT** наведені відповідно в підрозділах 1.14 та 1.16).

Розглянемо більш детально означені вище дії з організації захищеного режиму роботи мікропроцесора. При цьому будемо користуватися фрагментами програми **P_MODE** (повний текст програми приведений в підрозділі 4.15), яка написана на мові Паскаль з використанням асемблерних вставок і виконує всі необхідні дії з організації захищеного режиму, а також програмним модулем **PROT**.

1.3. Визначення розрядності мікропроцесора

Оскільки в діях з організації захищеного режиму для 16-розрядних мікропроцесорів 80286 і для 32-розрядних та 64-розрядних МП наступних моделей є суттєві відмінності, то необхідно спочатку визначити розрядність установленого в комп'ютері мікропроцесора.

Для цього застосовується наведена в окремому модулі **CPU** процедура **get_CPU(par:byte;var fam:byte)**, яка має два параметри: один вхідний – **par** і один вихідний – **fam**.

Вхідний параметр може прийняти значення 0 та 1. Якщо **par** = 0, то процедура **get_CPU** здійснює виведення на екран назви моделі МП та його бренд-рядка (якщо він підтримується даним МП). При виконанні програми **P_MODE,** яка викликає процедуру **get_CPU** з параметром **par** = 0 на екран виводяться такі повідомлення:

– для МП 80286 – «**i80286**»;

– для МП, на якому розроблювався цей текст (рис. 1.1):

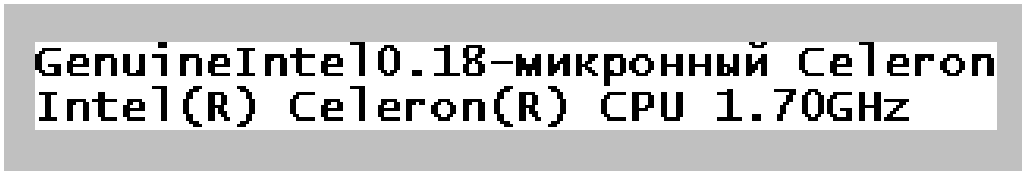

Рисунок 1.1 – Назва моделі МП та його бренд-рядку

При **par** = 1 визначаються та виводяться на екран також такі параметри мікропроцесора, як його функціональні можливості, характеристики кеш-пам'ятей та ін. (детальніше описання процедури **get_CPU** дивиться в розділі «Визначення параметрів мікропроцесора»).

Вихідний параметр **fam** повертає номер сімейства мікропроцесора, наприклад для МП 80286 – 2, для МП 80386 – 3 і так далі. У програмі **P_MODE** він застосовується для визначення розрядності мікропроцесора: якщо **fam** = 2, – це 16-розрядний МП, в іншому випадку – це 32-розрядний або 64-розрядний МП.

Додамо, що визначення розрядності МП є обов'язковим лише в тому випадку, якщо програма повинна використовуватися для організації захищеного режиму як для 32-розрядних або 64-розрядних мікропроцесорів, так і для 16-розрядних. Зараз 16-розрядні МП практично не використовуються. Тому визна-

чення розрядності МП і роз'яснення особливостей програмування для 16-розрядних МП наведені тільки в програмі **P_MODE.** Наступні програми – **P_INT**, **P_TASK**, **P_USER** та **P_PAGE** розроблені тільки для роботи в 32-розрядних або 64-розрядних МП і модуль **CPU** не застосовують.

1.4. Формування глобальної дескрипторної таблиці

Сегментація пам'яті, тобто розбивка пам'яті на окремі блоки, що називаються сегментами, здійснюється по-різному в реальному та захищеному режимах.

У захищеному режимі атрибути сегментів, що задають місцеположення сегмента в загальному адресному простору, його розмір і особливості доступу до нього, задаються у вигляді 8-байтної структури даних, яка називається дескриптором сегменту.

Дескриптори зберігаються в пам'яті у вигляді дескрипторних таблиць. Дескриптори сегментів коду, стека, даних, а також системні дескриптори, знаходяться в глобальній дескрипторній таблиці – *GDT* (Global Descriptor Table).

При мультізадачній роботі (див. розділ 6) кожна задача може мати свої сегменти коду, стека і даних, які є недоступними для інших задач. У цьому випадку дескриптори цих сегментів поміщаються в локальну дескрипторну таблицю – *LDT* (Local Descriptor Table), яка може бути сформована, якщо це необхідно, для кожної задачі.

Дескриптори оброблювачів переривань (див. розділ 5) зберігаються в дескрипторній таблиці переривань – *IDT* (Interrupt Descriptor Table).

У реальному режимі для опису атрибутів сегмента дескриптор не вимагається, бо базова адреса сегмента, що указана в 16-байтових одиницях – параграфах, зберігається у відповідному сегментному регістрі, границя сегмента має фіксоване значення 0FFFFh (64 КБ - 1) байтів, щодо атрибутів доступу, то немає ніякої заборони на доступ (виконання, запис чи читання) до будь-якого сегмента.

1.4.1. Структура дескриптора сегмента

На рис. 1.2 наведений формат дескриптора сегмента пам'яті для 32-розрядних МП, що має такі поля:

Рисунок 1.2. – Формат дескриптора сегмента

Границя сегмента займає в дескрипторі два поля: перше поле (байти 0 та 1) задає молодші розряди (15-0) значення границі сегмента, а друге поле (чотири молодших біта байта 6) – розряди 19-16 значення границі сегмента.

Границя сегмента визначається по різному в залежності від того, як розширюються дані в сегменті: якщо дані розширюються в напряму збільшення адрес – це можуть бути сегменти коду, даних чи системні сегменти, то границя сегмента дорівнює розміру сегмента, зменшеному на 1.

Якщо дані розширюються в напряму зменшення адрес (сегмент стека), то границя сегмента дорівнює різниці верхньої границі сегмента та його розміру, зменшеному на 1. Значення верхньої границі сегмента визначається бітом *B* дескриптора і дорівнює: якщо *B* = 0, то 64 КБ - 1, інакше – 4 ГБ - 1.

При кожному звертанні до сегменту мікропроцесор здійснює перевірку, чи не перевершує відносна адреса даних або команди границю сегменту. І якщо таке трапляється, генерує внутрішнє апаратне переривання (виключення) загального захисту з номером 13 – ***#GD*** (Global Defence).

Базова адреса сегмента теж займає в дескрипторі два поля: байти 2-4 задають розряди 23-0 базової адреси сегмента, а байт 7 – розряди 31-24 базової адреси сегмента, значення якої указуються в байтах.

Наявність двох полів для границі сегмента і його базової адреси пов’язана із забезпеченням сумісності програм, написаних для МП 80286, з програмами для наступних мікропроцесорів: при виконанні програм на МП 80286 молодші шість байтів дескриптора сегмента в цьому випадку повністю співпадають з

форматом дескриптора сегмента для МП 80286, а два старших байти повинні заповнюватися нулями.

Байт доступу дескриптора визначає правила доступу до сегмента, що вибирається, і в залежності від типу сегмента має різні формати, які представлені на рис. 1.3. Біти і поля байта доступу мають наступне призначення:

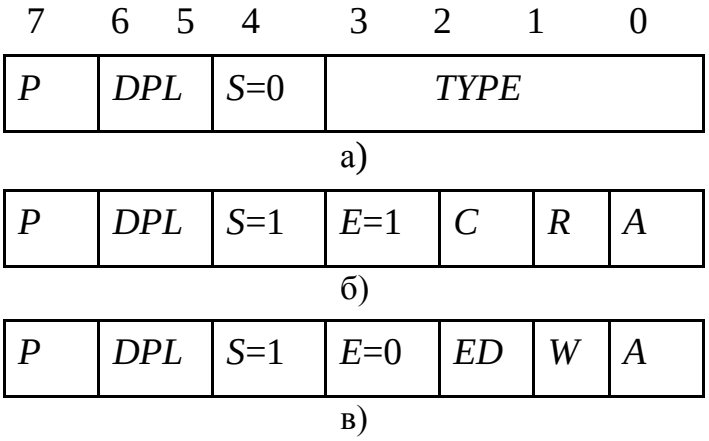

Рисунок 1.3 – Формат байта доступу:

а) дескриптора системного сегмента;

б) дескриптора сегмента коду;

в) дескриптора сегмента даних і стека.

Біт *P* (Present) присутності визначає наявність відповідного сегмента в пам'яті (*P* = 1), або його відсутність в пам'яті (*P* = 0), в цьому разі сегмент знаходиться у віртуальній пам'яті на жорсткому магнітному диску.

Якщо в регістр сегмента занесений селектор дескриптора, що має *P* = 0, то при зверненні до цього сегмента виникає переривання 11 «Сегмента немає в пам'яті» (для сегмента стека – це переривання 12).

Поле *DPL* (Descriptor Privilege Level) рівня привілею дескриптора вказує на ступінь захисту сегмента при доступі до нього (для сегментів даних, стека та системного сегмента), чи на ступень привілею програми – для сегмента коду. Може одержувати значення від 0 (найбільш захищений сегмент чи програма з найбільшим привілеєм) до 3 (незахищений сегмент чи програма з найменшим привілеєм).

Біт *S* (System) визначає, чи є сегмент системним: *S* = 0, чи ні: *S* = 1. Якщо сегмент несистемний, він має бути сегментом коду, стеку або даних. Якщо він є системним, тоді в полі **TYPE** вказується його тип (рис. 1.3, а).

Біт *E* (Execute) виконання визначає, чи можна сегмент виконати: *E* = 1

означає, що це сегмент коду (рис. 1.3, б), $E = 0$ – що це або сегмент даних, або стека (рис. 1.3, в).

Біт *A* (Accessed) доступу встановлюється апаратно (тобто вузлами МП) в "1" при доступі до сегмента (скидається тільки програмно).

Біт *R* (Readable) дозволу зчитування використовується для сегмента коду і дозволяє при $R = 1$ зчитувати його вміст. При $R = 0$ спроба зчитування даних призводить до виникнення переривання 13 *#GD* (те ж саме відбувається при спробі запису в сегмент коду незалежно від значення біта *R*).

Біт *C* (Conforming) підпорядкування визначає додаткові правила звернення до сегмента коду (див. розділ 7 «Захист пам'яті»).

Біт *W* (Writable) дозволу запису використовується для сегментів стека та даних і дозволяє (при $W = 1$), або забороняє (при $W = 0$) зміну вмісту сегмента даних. При порушенні виникає переривання 13 *#GD*.

Дескриптор сегмента стека обов'язково повинен мати $W = 1$ (при порушенні виникає переривання 12 – помилка при роботі зі стеком).

Біт *ED* (Expansion Direction) напряму розширення даних визначає, що при $ED = 0$ дані в сегменті містяться в напряму зростання адреси від базової адреси сегмента до його границі, означеної в дескрипторі (це реалізується в сегментах даних).

При $ED = 1$ (розширення вниз) дані в сегменті розташовані в напрямі зменшення адрес. Це реалізується в сегментах стека, де дані містяться починаючи з комірки, адреса якої дорівнює верхній границі сегмента (64 КБ - 1 чи 4 ГБ - 1 в залежності від значення біта *B*), до границі сегмента, зазначеної в дескрипторі сегмента.

Біти *P* і *A* байта доступу дескриптора сегмента можуть бути використані операційною системою (ОС) для організації віртуальної пам'яті. ОС періодично перевіряючи біт *A* дескрипторів всіх сегментів та скидаючи його, якщо він встановлений, визначає час останнього доступу до кожного сегмента. Якщо сегмента, до якого виконується звернення, немає в пам'яті, тобто $P = 0$, виробляється відповідне переривання, й операційна система, обробляючи це переривання, зчитує цей сегмент з магнітного диска до пам'яті. І якщо в пам'яті немає для цього вільного місця, з неї усувається на диск саме той сегмент, до якого, як це визначила операційна система, довше всього не було доступу.

Біти старшої частини байта 6 дескриптора мають таке призначення:

Біт *G* (Granularity) дрібності вказує, в яких одиницях задана границя сегмента:

– при *G* = 0 – в байтах,

– при *G* = 1 – в сторінках об'ємом 4 КБ.

Таким чином сегмент може мати розмір до 1 МБ (2^{20}) при *G* = 0 і до 4 ГБ (2^{32}) при *G* = 1.

Біт *D/B* має неоднакове значення для різних сегментів:

Для сегмента коду він має назву ***D*** (Default size) – біт розміру за замовчуванням, і визначає розрядність операнда та розрядність відносної адреси за замовчуванням:

– при *D* = 0 – 16 розрядів;

– при *D* = 1 – 32 розряди.

Розрядність, що приймається за замовчуванням, може бути програмно змінена за допомоги префіксу розрядності даних (**66*h***) або адреси (**67*h***). Відзначимо, що для реального режиму роботи МП значення розрядності за замовчуванням дорівнює 16.

Для сегмента стека цей біт називається ***B*** (Big) і визначає таке:

– якщо сегмент стека визначений як сегмент даних, тобто *ED* = 0, то при *B* = 1 розміри комірки стека і регістра *ESP* дорівнюють 32 розряди (при *B* = 0 - 16);

– якщо сегмент стека поширюється вниз (*ED* = 1), то біт *B* крім цього визначає також верхню границю сегмента стека: при *B* = 0 вона дорівнює 0FFFFh (64 КБ - 1), при *B* = 1 – 0FFFFFFFFh (4 ГБ - 1).

Біт *X* може бути використаний системою або користувачем за своїм розсудом (цей біт мікропроцесором не обробляється).

1.4.2. Виконання доступу до сегментів пам'яті

Доступ до необхідного сегмента пам'яті здійснюється за допомоги селектора, що заноситься до відповідного сегментного регістра. На рис. 1.4 наведена структура сегментних регістрів:

– коду – *CS* (Code Segment);

– стека – *SS* (Stack Segment);

– даних – *DS* (Data Segment), ES (Extra Segment), *FS* і *GS* (регістри *FS* і *GS* з'явились у мікропроцесорах, починаючи з МП 80386).

	Регістри сегментів	Регістри дескрипторів сегментів (“тіньові регістри”)		
	Селектор (16 розрядів)	Базова адреса сегмента (32 розряди	Границя сегмента (32 розряди)	Атрибути доступу
CS				
SS				
DS				
ES				
FS				
GS				

Рисунок 1.4 – Структура сегментних регістрів

Сегментні регістри містять значення селекторів сегментів пам’яті. З кожним із сегментних регістрів пов’язаний програмно недосяжний ("тіньовий") дескрипторний регістр сегмента, де зберігаються 32-розрядна базова адреса сегмента, його 32-розрядна границя та атрибути доступу до сегменту – *DPL* та *C, R* чи *W* в залежності від типу сегмента.

Відзначимо, що саме з дискрипторного регістра, а не з сегментного регістра, мікропроцесор бере дані про сегмент пам’яті при доступі до нього. Це здійснюється як в реальному, так і у захищеному режимах роботи МП.

Селектор сегмента представляє собою 16-розрядний покажчик, що має три поля (рис. 1.5):

Поле *RPL* (Requested Privilege Level) визначає рівень привілею запиту, тобто вказує той допустимий рівень захисту сегмента, при якому сегмент може бути вибраний за допомогою даного селектора.

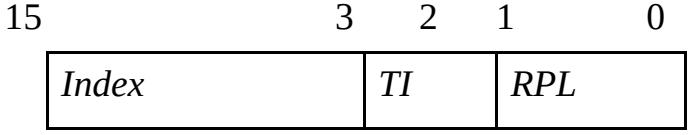

Рисунок 1.5 – Формат селектора сегмента

Поле *TI* (Table Indicator) служить для вибору дескрипторної таблиці:

– *TI* = 0: вибирається *GDT*;

– *TI* = 1: вибирається локальна дескрипторна таблиця *LDT* (Local

Descriptor Table).

Поле *Index* служить індексом (номером) для вибору одного з дескрипторів, що містяться в дескрипторній таблиці (*GDT* чи *LDT*).

Кожного разу при завантаженні селектора в сегментний регістр мікропроцесор знаходить по його індексу необхідний дескриптор сегмента (в *GDT* чи в *LDT*) та завантажує дані з цього дескриптора (базову адресу, границю та атрибути) у відповідний дескрипторний регістр.

Якщо зміщення дескриптора перевищить границю дескрипторної таблиці, то виробляється переривання 13 – «Порушення загального захисту» (#*GD*) .

Першим дескриптором в таблиці *GDT* завжди вказується дескриптор, який називається нуль-дескриптором, – він містить нулі в усіх своїх полях.

Завантаження нуль-дескриптора в сегментні регістри коду і стека зразу викликає переривання 13.

При завантаженні нуль-дескриптора в сегментні регістри даних переривання не виникає. Проте воно виникає при спробі програми звернутися після цього до цих сегментів. Ця обставина може бути використана для відмови доступу до тих або інших сегментних регістрів даних.

Максимальне число дескрипторів таблиці визначається форматом селектора й дорівнює 8192 (2^{13}).

Число дескрипторних таблиць, доступних задачі (*GDT* та *LDT*), їхній максимальний розмір, а також максимальний розмір сегментів визначають розмір віртуальної пам’яті МП: $2 \cdot 8192 \cdot 4$ ГБ = $2^1 \cdot 2^{13} \cdot 2^{32}$ байт = 2^{46} байт = 64 ТБ.

При звертанні до пам’яті мікропроцесор на етапі сегментації формує лінійну адресу операнда або команди, а після цього на етапі сторінкової організації перетворює її в фізичну адресу. Якщо сторінкова організація не використовується, то лінійна адреса буде одночасно і фізичною.

Лінійна адреса операнда або команди дорівнює сумі базової адреси сегмента, в якому знаходяться операнд або команда, та їхньої відносної адреси. При цьому базова адреса сегмента МП береться саме з дескрипторного регістра, а не з дескриптора сегмента в *GDT* (якщо це не зроблено навмисно в програмі), що зажадало би звернення до пам’яті й додаткових витрат часу.

Мікропроцесор також перевіряє, чи не виходить значення відносної адреси операнда чи команди за границю відповідного сегмента та, чи не порушуються права доступу до сегментів (ці значення теж беруться з дескрипторного регістру). Якщо таке трапляється, то виробляється переривання 13.

На рис. 1.6 зображена схема формування лінійної (фізичної при відсутності сторінкового перетворення) адреси, на якій не показані моменти завантаження і використання дескрипторного регістра.

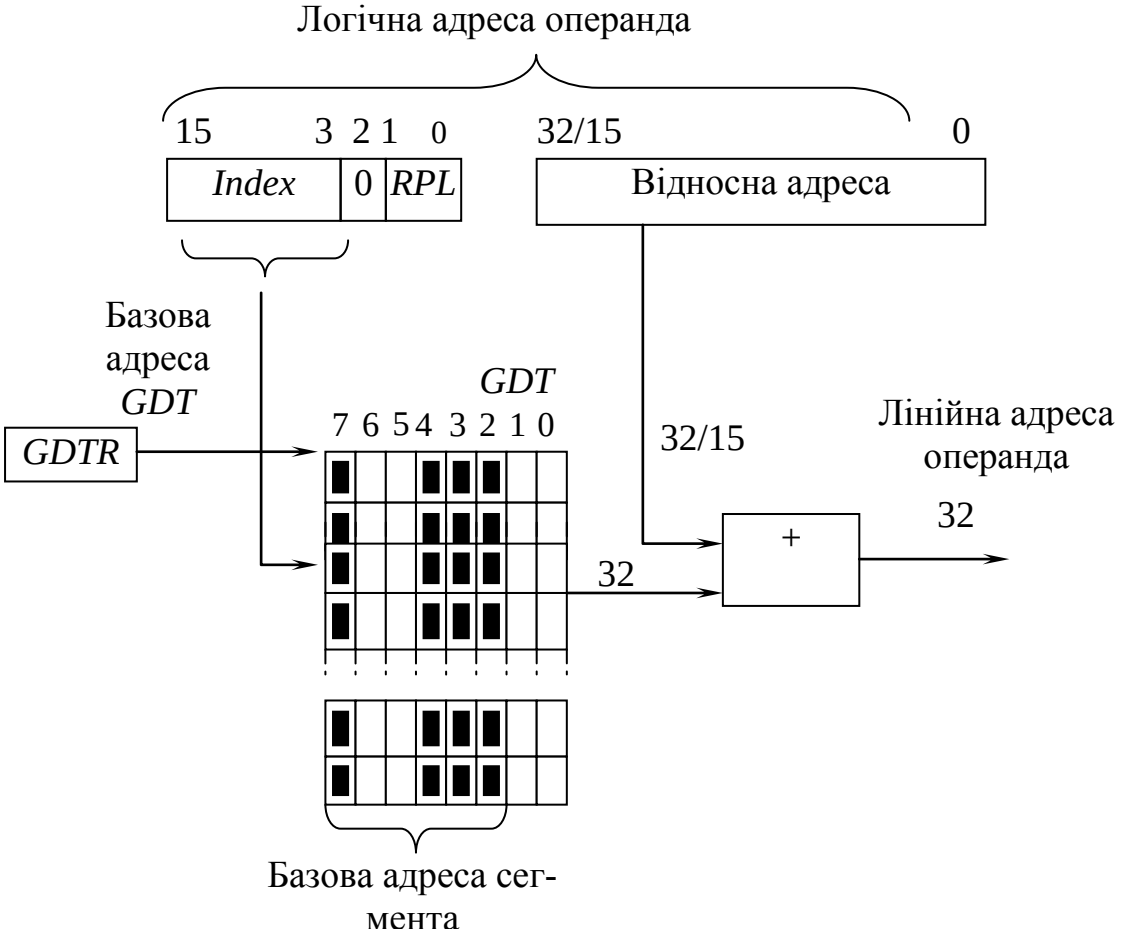

Рисунок 1.6 – Формування лінійної адреси

1.4.3. Формування дескрипторів сегментів пам’яті

К моменту запуску програми **P_MODE** системними програмами MS DOS вже здійснена сегментація пам’яті для роботи в реальному режимі:

– виділено місце в пам’яті для сегментів коду, стеку та даних;

– в сегмент коду записана програма в вигляді машинних команд (коду програми);

– в сегмент даних записані значення змінних та типізованих констант програми;

– в сегментні регістри коду, стека та даних занесені базові адреси цих сегментів (задані в параграфах).

У програмі **P_MODE** при роботі у захищеному режимі застосовуються

такі сегменти: сегмент коду, сегмент стеку, сегмент даних, сегмент відеопам'яті та сегмент розширеної пам'яті. Причому базові адреси та розмір сегментів коду, даних та стека повинні у захищеному режимі мати ті самі значення, що і в реальному режимі, якщо програма має використовувати при роботі у захищеному режимі дані, занесені в ці сегменти в реальному режимі.

Загальним правилом є те, що для кожного сегмента пам'яті, який буде застосовувати програма у захищеному режимі, повинен бути розроблений відповідний дескриптор для завдання параметрів сегмента. Усі розроблені дескриптори сегментів пам'яті повинні бути збережені в таблиці *GDT*.

Але у випадку, коли сегменти в реальному та захищеному режимах мають однакові параметри, можна дескриптори цих сегментів не формувати – при доступі до них МП буде брати їх параметри із дескрипторних регістрів.

У цілому цей підхід не може бути рекомендований, так як містить обмеження – у захищеному режимі не можна завантажувати сегментні регістри, для яких не розроблені та не занесені до таблиці *GDT* дескриптори.

У програмі **P_MODE** не формуються дескриптори двох сегментів:

а) сегмента даних – в навчальних цілях (щоб показати, що така можливість існує);

б) сегмента коду – для спрощення програми (щоб не торкатися питань обробки переривань, які детально розглядаються в розділі 2 "РОБОТА З ПЕРЕРИВАННЯМИ У ЗАХИЩЕНОМУ РЕЖИМУ").

Тому при роботі програми **P_MODE** у захищеному режимі заборонені переривання – при обробці переривання здійснюється виклик оброблювача переривання з завантаженням сегментного регістру *CS*.

Крім того, треба розробити дескриптор сегмента даних реального режиму, який має параметри, які є обов'язковими для сегментів даних реального режиму: границя дорівнює 0FFFFh, *DPL* = 0, *ED* = 0, *W* = 1. Селектор цього дескриптора використовується для завантаження сегментних регістрів даних та стека при повертанні мікропроцесора з захищеного режиму до реального.

1.4.3.1. Завдання селекторів сегментів пам'яті

Усі дескриптори сегментів програми можуть розташовуватися в таблиці *GDT* на будь-якому місці, тобто мати будь-який номер (індекс), крім нуль-дескриптора – він завжди повинен мати індекс 0.

Тому для однозначного звертання до того чи іншого сегмента пам’яті для кожного сегмента треба задати його індекс. При цьому з метою зменшення числа помилок при розробці програми вирішено:

1. Замість індексів дескрипторів сегментів при звертанні до глобальної дескрипторної таблиці застосовувати селектори сегментів. Це пов’язане з двома обставинами: по-перше, в сегментні регістри завантажуються значення саме селекторів, а по-друге, селектор дескриптора сегмента відрізняється від його індексу тільки полями *TI* та *RPL* (див. рис. 1.4), які в більшості програм, і в програмі **P_MODE** зокрема, мають значення: *TI* = 0, *RPL* = 0, тобто значення індесу дорівнює значенню селектора, зсунутого на три розряди праворуч.

2. Значення селекторів пам’яті для всіх програм, що працюють у захищеному режимі (**P_MODE, P_INT**, **P_TASK** та ін.), зробити єдиними (хоча це не є обов’язковим) та зберігати в модулі **PROT**.

У програми **P_MODE** застосовуються такі селектори:

```
stack_sel: word  =$28;     { стека, }
real_sel :  word  =$38;    { даних реального режиму, }
text_sel:  word   =$40;    { відео-пам’яті текстового режиму, }
mem_sel  :word =$48;       { розширеної пам’яті. }.
```

1.4.3.2. Формування полей дескрипторів сегментів пам’яті

Згідно зі структурою дескриптора сегмента, що наведена рисунках 1.1 та 1.2, був розроблений та приведений в модулі **PROT** тип змінної **t_dt** для опису дескрипторів сегментів пам’яті в вигляді запису:

```
t_dt=record
lim_l,            { границя сегмента (біти 15-0) }
base_l :word;     { базова адреса сегмента (біти 15-0) }
base_h,           { базова адреса сегмента (біти 23-16) }
acc,              { байт доступу }
lim_h,            { G,D,0,X та границя сегмента (біти 19-16) }
base_hh:byte      { базова адреса сегмента (біти 31-24) }
end;.
```

Тип **t_dt** може бути застосований для формування дескрипторів обох дескипторних таблиць – *GDT* і *LDT*, тому що вони мають однаковий формат дескрипторів.

Оскільки всі програми, що працюють у захищеному режиму, мають єдину нумерацію селекторів сегментів (див. попередній підрозділ), то ці програми можуть мати і єдину таблицю *GDT*. Тоді з урахуванням приведеної вище структури дескриптора сегмента може бути сформована глобальна дескрипторна таблиця *GDT* в вигляді масиву (модуль **PROT**):

gdt:array[0..21] of t_gdt;.

де число 21 показує кількість дескрипторів, що застосовуються в усіх розроблених на даний час програмах, та містить таблиця *GDT*. Тоді доступ до значення і-го дескриптора (елемента масива) таблиці *GDT* здійснюється таким чином:

gdt[i].

Для всіх дескрипторів при *TI* = 0, (таблиця *GDT*), та *RPL* = 0 (запит здійснюється на рівні привілею 0), індекс дескриптора сегмента буде дорівнювати значенню його селектора, яке зсунуте на три біта праворуч.

Застосовуючи вище наведені описи структури дескриптора сегмента та таблиці *GDT*, покажемо, як робиться формування дескрипторів сегментів пам'яті таблиці *GDT*. Як приклад, візьмемо один з сегментів, що застосовують всі програми захищеного режиму, в тому числі і програма **P_MODE**, – сегмент даних реального режиму.

1. Завдання границі сегмента.

Оскільки розмір всіх сегментів в реальному режимі становить 64 КБ, то поля молодшої та старшої частин границі сегмента даних реального режиму з індексом **real_sel shr 3** будуть дорівнювати відповідно

gdt[real_sel shr 3].lim_l:=$FFFF; *{ 64 КБ - 1}*

та

gdt[real_sel shr 3].lim_h:=0;.

2. Завдання базової адреси сегмента.

У реальному режимі базова адреса сегментів зберігається в сегментних регістрах і задана в параграфах. Це значення для дескриптора сегмента треба визначити у байтах. Для формування 20-розрядного значення базової адреси сегмента, спочатку треба привести значення 16-розрядного регістра *DS* до 32-

розрядного формату, а потім виконати зсув на чотири розряди вліво для перетворення значення базової адреси з 16-байтних одиниць (параграфів) в значення в байтах. При цьому застосовується стандартна функція мови Pascal **Dseg**, що повертає 16-розрядне значення регістру *DS*:

gdt[real_sel shr 3].base_l:=longint(Dseg) shl 4;.

У поле base_h заносяться чотири старших розряди значення регістру *DS*:

gdt[real_sel shr 3]. base_h:= Dseg shr 12;.

Так як максимальне значення базової адреси в реальному режимі не може перевершувати 1 МБ (точніше 10FFDFh), що менш ніж 16 МБ, то значення самої старшої частини базової адреси в дескрипторі дорівнює нулю:

gdt[real_sel shr 3]. base_hh:=0;

3. Формування байта доступу.

Для формування байта доступу дескриптора сегмента даних реального режиму маємо такі дані:

P = 1 – сегмент знаходиться в пам'яті;

DPL = 0 – рівні привілеїв всіх сегментів в реальному режимі дорівнюють 0;

S = 1 – сегмент даних не є системним сегментом;

E = 0 – це не сегмент, що виконується, тобто не сегмент коду;

ED = 0 – дані розширюються в напрямку зростання адреси (не сегмент стека);

W = 1 – в реальному режимі в будь-який сегмент можна записувати дані;

A = 0 – цей біт встановлюється апаратно процесором.

Таким чином байт доступу дескриптора побітно дорівнює значенню 10010010_2 = $92:

gdt[real_sel shr 3].acc:= $92;.

У реальному режимі працюємо з 16-розрядними даними та зміщеннями, тому біти *G* та *D/B* дорівнюють 0. Таким чином значення 6-го байта дескриптора остається незмінним (**gdt[real_sel shr 3].lim_h:=0**).

Для зручності в модулі **PROT** значення окремих бітів і полів байта доступу у відповідності з рис. 1.2 задані у вигляді констант. Наприклад, біт *P* представлений константою **present = $80** і так далі (див. текст модуля в підрозділі 1.15). Тоді байт доступу для кожного сегмента визначається як логічна (чи арифметична) сума констант.

Для сегмента даних байт доступу буде дорівнювати:

acc_data: = present OR no_sys OR wr_ena,

тобто він знаходиться в пам'яті (**present**), не є системним (**no_sys**), в нього можна писати (**wr_ena**).

1.4.3.3. Застосування процедури **init_gdt** для формування *GDT*

Для більш зручного та ефективного формування дескрипторів сегментів таблиці *GDT* можна використовувати процедуру

init_gdt (sel:byte; limit, base:longint; access, byte_6:byte)

модуля **PROT**, параметри якої мають такі значення:

– sel – селектор дескриптора сегмента (в процедурі **init_gdt** по його значенню визначається індекс дескриптора);

– limit – границя сегмента;

– base – базова адреса сегмента;

– access – байт доступу сегмента;

– byte_6 – значення байта 6 дескриптора сегмента з урахуванням тільки чотирьох старших розрядів *G*, *D/B* та *X*).

При роботі процедури **init_gdt** перевіряється, чи не перевершує параметр **limit** (границя сегмента) значення 0FFFFFh (1 МБ - 1). Якщо перевершує, то значення границі сегмента треба задати не в байтах, а в 4 КБ одиницях (сторінках). Для цього параметр **limit** зсувається на 12 розрядів праворуч. Крім цього встановлюється значення біта ***G*** = 1. Це здійснюється шляхом додавання через логічну операцію "АБО" до параметра **byte_6** значення 80h.

Після цього значення параметрів процедури **limit, base, access** та **byte_6** заносяться у відповідні поля дескриптора сегмента згідно з його структурою.

З метою спрощення розробки програм, що працюють у захищеному режимі, та скорочення їх вихідних тестів в секції ініціалізації модуля **PROT** за допомоги процедури **init_gdt** здійснюється створення базової таблиці *GDT*, яка містить дескриптори сегментів, що застосовуються більшістю програм у захищеному режимі:

```
                                        { Нуль-дескриптор: }
init_gdt(0,0,0,0,0);
                                { Дескриптор сегмента коду модуля PROT: }
init_gdt(PROT_sel,$ffff,longint(CSeg) shl 4,acc_code,0);
                                        { Дескриптор сегмента стека: }
init_gdt(stack_sel,0,longint(Sseg) shl 4,acc_stack,0);
                                        { Дескриптор сегмента даних:}
```

init_gdt(data_sel,$ffff,longint(Dseg) shl 4,acc_data,0);

{ Дескриптор сегмента даних реального режиму: }

init_gdt(real_sel,$ffff,longint(Dseg) shl 4,acc_data,0);

{ Дескриптор сегмента відеопам'яті: }

init_gdt(video_sel,4000-1,$b8000,acc_data,0);

{ Дескриптор сегмента розширеної пам'яті: }

init_gdt(mem_sel,$fffff,1024*1024,acc_data,$80);.

При формуванні цих дескрипторів сегментів було прийнято до уваги наступне.

Базові адреси та розміри сегментів стека та даних при формуванні відповідних дескрипторів задані ті ж самі, які вони мали в реальному режимі (див. підрозділ 1.2.3).

При формуванні дескриптора сегмента стека, який має розмір 64 КБ, параметр, що задає його границю дорівнює 0 – тому що дані в сегменті стека розширюються донизу (*ED* = 1).

Щоб показати, що відеосистема працює в текстовому режимі й для виведення даних застосовується один екран відеопам'яті, при формуванні дескриптора сегмента відеопам'яті в процедурі **init_gdt** параметр базової адреси сегмента дорівнює $b8000 байтів, а параметр границі сегмента – 4000 - 1 байтів.

Таким чином кожна програма при підключенні модуля **PROT** вже має сформовану базову таблицю *GDT*. Їй залишається тільки сформувати дескриптори для тих сегментів пам'яті, які не задані у базовій таблиці, але мають використовуватися в конкретній програмі.

Програма **P_MODE** з дескрипторів базової таблиці *GDT* не використає лише дескриптор сегмента коду модуля **PROT** – тому що не застосовує функції та процедури цього модулю при роботі у захищеному режимі.

Єдиним сегментом, який використає програма **P_MODE** і дескриптора якого нема в базовій GDT, є сегмент розширеної пам'яті, який застосовується в програмі для визначення розміру установленої на комп'ютері пам'яті (див. підрозділ 1.8).

Для формування його дескриптора, здійснюється виклик процедури **init_gdt** з такими параметрами, які вказують, що цей сегмент пам'яті займає адресний простір від адреси 1 МБ до адреси 4 ГБ - 1:

init_gdt(mem_sel,$ffefffff,1024*1024,acc_data,0).

Але можливо викликати для цього процедуру **init_gdt** і таким чином:

init_gdt(mem_sel,$ffeff,1024*1024,acc_data,$80);.

Значення $80 остатнього параметра процедури означає, що біт G 6-го байта дескриптора сегмента дорівнює 1, тобто границя сегмента задана в 4 КБ одиницях. Тому об'єм сегмента розширеної пам'яті становить ($ffeff + 1)* 4 КБ = 4 ГБ - 1 МБ.

1.5. Завдання адреси та розміру GDT

Після формування GDT необхідно указати процесору місцеположення цієї таблиці в пам'яті та її розмір.

Це робиться за допомогою 48-розрядного регістра *GDTR* (Global Descriptor Table Register), що містить 32-розрядну базову адресу таблиці *GDT* та її 16-розрядну границю. Структура *GDTR* представлена на рис. 1.7.

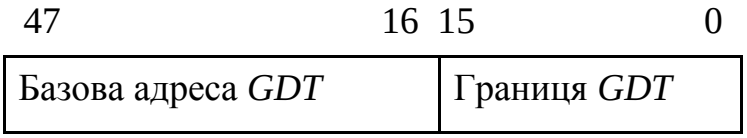

Рисунок 1.7 Структура регістра GDTR

Формат даних регістра *GDTR* в модулі **PROT** представлений у вигляді наступного запису:

```
t_dtr=record        { Структура регістрів GDTR та IDTR:   }
lim:word;           { границя дескрипторної таблиці,   }
base:longint;       { базова адреса дескрипторної таблиці   }
end;.
```

Вміст регістра *GDTR* описується змінною **gdtr** типу **t_dtr**.

Для формування даних регістра *GDTR* та завантаження їх в цей регістр використовується процедура **init_gdtr(sel:word)** модуля **PROT**. По-перше, вона визначає границю і базову адресу таблиці *GDT* і записує їх в відповідні поля

змінної **gdtr:**

gdtr.lim:=:sel+7;
gdtr.base:=lin_adr(seg(gdt),ofs(gdt));.

Завдання границі таблиці *GDT* через значення параметра **sel** процедури **init_gdtr**, а не за допомоги функції, що визначає розмір змінної **gdt** – **sizeof(gdt)** - 1, пов'язане з тим, як це було вирішено відносно таблиці *GDT* – ця таблиця буде єдиною для всіх програм, що працюють у захищеному режимі (**P_MODE**, **P_INT**, **P_TASK** та ін.). При цьому кожна з цих програм може використовувати лише частину таблиці *GDT*. Тому, щоб правильно вказати границю тієї частини таблиці *GDT*, яку використовує кожна окрема програма, вона при виклику процедури **init_gdtr** вказує значення селектора останнього дескриптора в *GDT*, серед тих, що застосовуються в цій програмі.

Для програми **P_MODE** таким селектором є селектор сегмента розширеної пам'яті **mem_sel**. Тому виклик процедури в цій програмі здійснюється таким чином:

init_gdtr(mem_sel);.

По-друге, процедура **init_gdtr** завантажує регістр *GDTR* сформованими в пам'яті даними. Для цього застосовується команда **LGDT gdtr**. Оскільки Турбо Паскаль не підтримує в асемблерних вставках команди системного характеру (в тому числі і команду **LGDT**), виникає потреба в формуванні й представленні цієї команди в асемблерних вставках у машинному коді.

Машинний код команди **LGDT** має такий вигляд:

00001111 00000001 mod 010 r/m.

Поля *mod* та *r/m* третього байту команди визначають тип адресації операнда. При значеннях *mod* = 00 та *r/m* = 110 це буде пряма адресація, і тоді 4-й та 5-й байти команди будуть задавати зміщення змінної **gdtr** відносно базової адреси сегмента даних в пам'яті (6-й та 7-й байти команди застосовуються тільки в випадку 32-розрядної адресації). З урахуванням того, що транслятор в Турбо Паскалі в асемблерних вставках замість ім'я змінної підставляє її зміщення, сформований машинний код команди **LGDT** має такий вигляд:

asm
db 0fh,01h,16h
dw gdtr
end;.

Команда **SGDT**, що виконує занесення вмісту регістра *GDTR* до пам'яті, відрізняється від команди **LGDT** тільки третім байтом:

00001111 00000001 mod 000 r/m.

При роботі програми **P_MODE** у захищеному режимі значення регістра *GDTR* не змінюється і немає потреби в зберіганні поточного значення цього регістра. Тому команда **SGDT** в програмі **P_MODE** та інших програмах захищеного режиму не застосовується.

1.6. Підготовка до скидання мікропроцесора

Для МП 80286 повернення в реальний режим із захищеного здійснюється тільки шляхом скидання мікропроцесора, що виконується командою контролера клавіатури. Для МП наступних моделей можливі два варіанти повернення в реальний режим: за допомогою команди MOV або також через скидання МП.

У випадку повернення через скидання мікропроцесора необхідно в реальному режимі підготувати структури даних для здійснення повернення в задану точку програми після скидання МП.

Завжди після виконання скидання процесор переходить в реальний режим і управління передається програмі BIOS. Ця програма аналізує вміст комірки *CMOS*-пам'яті з адресою 0Fh, в якій зберігається байт стану відключення мікропроцесора.

Байт стану відключення використовується *BIOS* для визначення подальших дій після апаратного скидання МП. У табл. 1.1 наведені можливі значення байта стану відключення.

Таблиця 1.1–Значення байта стану відключення

Значення	Причина відключення
0	Програмне скидання при натисненні комбінації клавіш CTRL-ALT-DEL або неочікуване скидання. Виконується звичайний перезапуск системи, але процедури тестування, які працюють при ввімкнені живлення, не виконуються.
1	Скидання після визначення об'єму пам'яті.
2	Скидання після тестування пам'яті.
3	Скидання після знаходження помилки в пам'яті (контроль парності).
4	Скидання із запитом перезавантаження.
5	Після скидання перезавантажується контролер переривань, потім управління передається на адресу, яка знаходиться в області даних BIOS 0040h:0067h.
6,7,8	Скидання після виконання тесту роботи процесора у захищеному режимі.
9	Скидання після виконання пересилання блоку пам'яті із основної пам'яті в розширену.
0Ah	Після скидання управління негайно передається за адресою 0040h:0067h в області даних BIOS.

Якщо у захищеному режимі не використовуються переривання і, відповідно, немає потреби в перепрограмуванні контролера переривань, в комірку *CMOS*-пам'яті з адресою 0Fh треба записати значення 0Ah. При цьому після скидання МП, управління буде передане за адресою, яка взята з області даних *BIOS* – 0040h:0067h.

Якщо треба перепрограмувати контролери переривань для зміни номерів переривання (див. розділ 5), то в байті стану відключення треба вказати значення 5. У цьому випадку спочатку буде здійснено програмування контролерів переривань для реального режиму, а потім управління буде передане за адресою, яка взята з області даних *BIOS* 0040h:0067h.

Для запису байта даних в комірку *CMOS*-пам'яті необхідно спочатку в порт з адресою 70h записати номер потрібної комірки, а після цього в порт 71h занести дані:

Port[$70]:=$F;
Port[$71]:=$A;.

Якщо після скидання процесора програма повинна працювати починаючи з мітки **real:**, то спочатку треба визначити зміщення цієї мітки. Це можна зробити за допомогою оператора **offset** на асемблері (на жаль Паскаль не дає такої можливості):

asm mov ofs_real,offset real end;.

Після цього потрібно занести за адресою 0040h:0067h значення повної адреси (зміщення і сегмент) мітки:

memw[$40:$67]:= ofs_real;
memw[$40:$69]:=Cseg;.

Після виконання цих дій, в разі скидання мікропроцесора управління в програмі буде передано на мітку **real:**.

1.7. Заборона маскованих та немаскованих переривань

Перед переходом у захищений режим необхідно заборонити усі зовнішні апаратні переривання – як масковані, так і не масковані.

Обробка маскованих переривань мікропроцесором не виконується, якщо скинуто прапор *IF* регістра *FLAFS/EFLAFS*. Скидання прапора *IF* виконує команда *CLI* (clear interrupt-enable flag):

asm cli end;.

Значення прапора *IF* також може бути змінено командами **PUSHF** та **POPF**.

Для заборони немаскованих переривань необхідно в порт 70h занести байт даних, який містить в старшому розряді одиницю:

port[$70]:=$80;.

1.8. Зберігання в пам’яті регістрів мікропроцесора

Оскільки при переході у захищений режим мікропроцесора значення його регістрів змінюються, то необхідно заздалегідь виконати зберігання в пам’яті

тих значень сегментних регістрів коду, даних, стека, *ES* та регістра покажчика стека, які вони мали в реальному режимі.

```
real_cs:=Cseg;
memw[0:$60*4]:=Dseg;
real_ss:=Sseg;
asm mov real_es,es end;
real_sp:=SPtr;.
```

Занесення значення сегментного регістра даних *DS* в пам'ять по абсолютній адресі, а не через змінну, як це зроблено для інших регістрів, пов'язано з тим, що повна адреса комірки пам'яті складається зі значення сегментного регістра даних і зміщення, яке є значенням змінної. При загубленні значення сегментного регістра даних стає неможливим визначити повну адресу комірок пам'яті.

Вибір адреси комірки 0:4*60h пов'язаний з тим, що вона призначена для зберігання вектора переривання користувача і зберігається операційною системою недоторканою.

1.9. Переведення мікропроцесора у захищений режим

Регістр CR0 (рис. 1.8), який вперше з'явився в складі мікропроцесора моделі Intel 80386, включає 10 розрядів (біти 6-15 та 19-28 – зарезервовані) для управління роботою процесора і визначення його стану (біти *CD*, *NW*, *AM*, *WP* та *NE* з'явилися в МП починаючи з i486):

Біт PE (Protection Enable) дозволу захисту встановлюється в стан '1' для переключення МП у захищений режим. Якщо біт *PE* скинутий – МП працює в реальному режимі;

31	30	29	28 … 19	18	17	16	15 … 6	5	4	3	2	1	0
PG	*CD*	*NW*		*AM*		*WP*		*NE*	*1*	*TS*	*EM*	*MP*	*PE*

Рис. 1.8. Формат регістра управління CR0

Біт MP (Monitor coProcessor) управління співпроцесором FPU (Floating Point Unit) використовується разом з бітом *TS* для генерації переривання 7 (”FPU недоступний”) при обробці команди WAIT: якщо *MP* = 1 та *TS* = 1, то команда WAIT викликає переривання. Починаючи з МП i486 цей біт завжди встановлений;

Біт EM (Emulation) емуляції співпроцесора: коли біт встановлений викликається переривання 7 при виконанні команд FPU або переривання 6 при виконанні команд MMX (для МП, що виконує набір команд MMX). Значення EM не впливає на команду WAIT;

Біт TS (Task Switched) переключення задач: біт *TS* встановлюється при переключенні задач. Якщо *TS* = 1 і виконується команда FPU або MMX (якщо *MP* = 1, то і команда WAIT), то викликається переривання 7;

Біт 4 регістра CR0 мав назву *ET* (Extension Type) і застосовувався тільки в МП 80386 для завдання типу FPU, який встановлений на материнській платі комп’ютера: якщо *ET* = 0, то 80287, інакше – 80387. У МП, починаючи з i486, має апаратно встановлене значення 1.

Біт NE (Numeric Error) управління обробкою помилок FPU:

якщо *NE* = 0, то встановлюється схема обробки помилок, що використовується в МП 80286 і 80386 – через зовнішнє переривання IRQ13 (переривання від співпроцесора). При цьому на вхід IRQ13 контролера переривань надходить від МП сигнал FERR#, що приймає активний (низький) рівень при виникненні помилки FPU. Вхідний сигнал МП IGNNE# низьким рівнем забороняє обробку помилок FPU.

Якщо *NE* = 1, помилки співпроцесора обробляються як внутрішні переривання (виключення) з номером 16.

При скиданні процесора цей біт установлюється в 0, тому програмне забезпечення, яке має обробляти помилки FPU, як внутрішні, повинно встановити цей біт в 1;

Біт WP (Write Protection) біт захисту від запису – забороняє запис в сторінки, призначені тільки для читання для програм всіх рівнів привілеїв;

Біт AM (Alignment Mask) маски вирівнювання: при *AM* = 0 забороняється перевірка того, чи вирівняні операнди в програмах користувачів;

Біт NW (Not Write through) заборони запису в кеш-пам’ять;

Біт CD (Cache Disable) заборони кешування, тобто заповнення рядка кеш-пам’яті;

Біт PG (PaGing) дозволу сторінкової організації пам'яті. Установка *PG* = 1 дозволена тільки у захищеному режимі, тобто при *PE* = 1.

При роботі в реальному режиму в регістрі CR0 встановлюється значення 10h, тобто біт *PE* очищений. Для того, щоб перевести МП у захищений режим, необхідно встановити біт *PE* регістра управління CR0 в одиницю.

Занесення даних в регістр CR0 для мікропроцесорів починаючи з 80386, виконується спеціальною командою MOV, яка здійснює обмін даними між системними регістрами та регістрами загального призначення.

У МП 80286 замість 32-розрядного регістра CR0 є 16-розрядний регістр MSW (Machine Status Word). Занесення даних в MSW виконується командою LMSW. Для сумісності молодші 16 розрядів CR0 співпадають з MSW і запис в них може здійснюватися також і командою LMSW.

У програмі **P_MODE** установка біта *PE* MSW здійснюється для МП 80286 за допомогою команд SMSW та LMSW, які задані в машинних кодах:

```
db 0fh,01h,0e0h                    { SMSW AX }
or ax,1
db 0fh,01h,0f0h                    { LMSW AX }.
```

Для МП, починаючи з 80386, установка біта *PE* в регістрі CR0 здійснюється за допомоги двох спеціальних команд MOV, одна з яких здійснює передачу даних з одного з управляючих регістрів CRi (CR0, CR2, CR3 та CR4) в один з регістрів загального призначення РЗП (EAX, ECX, EDX, ESP, EBP, EBX, ESI та EDI) і має такий машинний код.

00001111 00100000 11 CRi РЗП.

Друга команда, навпаки, здійснює передачу даних з одного з регістрів РЗПі в один з управляючих регістрів CRi і має такий машинний код.

00001111 00100010 11 CRi РЗП.

Безпосередньо ж змінити значення регістра CR0 неможливо, так як управляючі регістри не підтримують логічні чи арифметичні операції.

Тому, враховуючи на те, що регістри CR0 та EAX мають код 000, встано-

влення біту PE в регістрі CR0 може бути реалізоване таким чином:

```
db 0fh,20h,0c0h        { MOV EAX,CR0 }
or al,1
db 0fh,22h,0c0h        { MOV CR0,EAX }.
```

Відразу після переходу МП у захищений режим необхідно виконати програмний перехід для очищення буфера передвиборки від команд, занесених туди в реальному режимі. Перехід здійснюється на мітку **@prot**, з якої починається робота МП у захищеному режимі:

jmp @prot.

Помітимо, що оскільки в програмі **P_MODE** не вимагалося перевантажувати сегментний регістр коду – у захищеному режимі використовувались ті ж базова адреса та розмір сегмента коду і ті ж права доступу до нього, замість міжсегментного переходу, який для цього вимагається, використовується короткий перехід. Міжсегментний перехід використовується в програмах **P_INT** та **P_TASK.**

Після цього в сегментні регістри SS, ES і GS завантажуються відповідні селектори сегментів:

```
@prot:
    mov ss,stack_sel
    mov es,video_sel
    db 8eh,2eh                { MOV GS, mem_sel }
    dw mem_sel.
```

Відзначимо, що асемблерні вставки в Паскалі не підтримують роботу з регістрами FS та GS, тому застосована машинна форма команди завантаження регістра GS. Крім того, в зв’язку з тим, що параметри сегмента даних у захищеному режимі не змінюються, регістр DS не треба завантажувати.

На цьому переведення мікропроцесора з реального у захищений режим завершується.

1.10. Робота мікропроцесора у захищеному режимі

У захищеному режимі роботи мікропроцесора виконуються дві процедури програми **P_MODE**: **out_screen** та **know_mem_size.**

Після виклику **call out_screen** процедура **out_screen** шляхом прямого звернення до відеопам'яті здійснює виведення на екран таких повідомлень:

– для МП 80286 – «**Работа в защищенном режиме**»;

– для МП наступних моделей – «**Работа в защищенном режиме: размер памяти** – ».

Повідомлення, яке виводиться на екран задається (на мові Pascal) в вигляді типізованої константи:

const s:string='Работа в защищенном режиме: размер памяти – **'.**

Виведення на екран повідомлення здійснюється шляхом прямого доступу до відеопам'яті. При цьому символи рядка **s** з області пам'яті сегмента даних передаються в сегмент відіеопам'яті. Пряма передача із застосуванням рядкової команди **movs** неможлива через те, що символи рядка в сегменті даних зберігаються в вигляді 8-розрядних ASCII-кодів, а в відеопам'яті вони повинні зберігатися в вигляді 16-розрядних даних – до ASCII-коду символа додається 8-розрядний відеоатрибут. Тому застосовуються дві рядкові команди:

lodsb – для читання символів рядка з пам'яті;

stosw – для виведення символу рядка та відеатрибуту до відеопам'яті.

При цьому адреси символів пам'яті задаються парою регістрів DS:SI (сегмент:зміщення), а адреси символів в відеопам'яті – регістрами ES:DI. Число символів, які треба вивести на екран можна визначити таким чином:

mov cl, [offset s],

при цьму другий операдн команди задає значення першого байту рядка **s**, в якому зберігається кількість символів рядка. Це значення заноситься до регістру CL, щоб можна було застосувати команду **loop** для організації циклу по всім символам рядка.

Адреса першого (а далі поточного) символу рядка заноситься в регістр SI:

mov si,offset s+1.

Адреса комірки відеопам'яті, куди заноситься перший символ з відеоатрибутом, задається в регістрах ES:DI, де ES містить селектор сегменту відеопам'яті, DI – зміщення 16-розрядної комірки відеопам'яті, яке обчислюєься з

урахування номерів рядка та стовбця екрану, куди буде виведений перший (а далі поточний символ разом з відеоатрибутом).

Відеоатрибут символу (його колір та колір фону) заноситься до регістру АН. Тоді цикл виведення символьного рядка на екран буде виглядати так:

@wxy:lodsb
stosw
loop @wxy.

Розмір установленої на комп'ютері пам'яті визначається за допомогою процедури **know_mem_size**, яка для кожної комірки розміром в один байт сегмента розширеної пам'яті, починаючи з адреси 1МБ (щоб при цьому не зіпсувати саму програму **P_MODE**), виконує такі дії:

– значення комірки зберігається (щоб не знищити програми або дані, що знаходяться в розширеній пам'яті);

– в комірку заносяться фіксовані дані (в програмі – код AAh);

– здійснюється зчитування даних з цієї комірки і порівняння їх з кодом, що записується;

– відновлюється початкове значення комірки;

– якщо порівняння показало рівність значень, що були записані і що були зчитані, то здійснюється перехід до наступної комірки пам'яті, якщо – нерівність, то попередня комірка була останньою коміркою пам'яті, встановленою на даному комп'ютері. При цьому в змінній **mem_size** зберігається знайдене значення розміру пам'яті в байтах, яке вже в реальному режимі виводиться на екран.

Зауважимо, якщо при виконанні програми **P_MODE** у захищеному режимі виникне помилка, то це приведе до зависання комп'ютера та його повторного запуску. Це відбувається тому, що в програмі **P_MODE** не застосовуються засоби оброблення виникших при виконанні програми помилок. Питання обробки переривань розглядаються в наступному розділі 2 "РОБОТА З ПЕРЕРИВАННЯМИ У ЗАХИЩЕНОМУ РЕЖИМУ". Приклади розроблення програмних засобів обслуговування переривань наведені в програмі **P_INT**.

1.11. Повернення в реальний режим

Для повернення в реальний режим у програмі **P_MODE** для МП 80286 використовується спеціальна команда контролера клавіатури, яка здійснює

скидання МП:

mov al, 0feh
out 64h, al.

Після цього треба очікувати завершення скидання мікропроцесора:

@wait:
hlt
jmp @wait.

Після скидання МП BIOS виконує перехід на точку у програмі, адреса якої зберігається в комірці $40: $67, тобто на мітку **real**.

Для МП, починаючи з 80386, (при **cpu_type**>2) повернення у реальний режим виконується за допомогою команди MOV, яка здійснює скидання біта PE регістра CR0. Перед цим згідно до рекомендацій фірми виробника МП Intel необхідно виконати такі дії:

a) заборонити апаратні масковані переривання командою CLI;

b) передати управління сегменту коду з R = 1 і границею 0FFFFh;

c) завантажити в сегментні регістри DS, SS, ES селектор сегмента, який має атрибути, що відповідають реальному режиму: границя сегмента, дорівнює 0FFFFh, ED = 0 і W = 1;

d) за допомогою команди LIDT завантажити в IDTR базову адресу (0) і границю (1КБ - 1) таблиці векторів реального режиму;

e) очистити біт PE регістра CR0;

f) здійснити міжсегментний перехід.

З указаних дій в даній програмі вимагається виконати тільки пункти "c", і "e". Пункти "a" та "d" не потрібні, тому, що програма не здійснювала обробку переривань. Пункт "b" не вимагає виконання, оскільки сегмент коду вже має означені параметри. Нарешті, пункт "f" не треба виконувати із-за того, що не було завантаження регістра CS при вході у захищений режим і при роботі у захищеному режимі.

Для завантаження сегментних регістрів SS і ES використовується селектор сегмента даних реального режиму real_sel, що задовольняє вказаним в пункті "c" умовам:

mov ss,real_sel
mov es,real_sel
db 8eh,2eh *{ MOV GS,real_sel }*
dw real_sel.

{---------Скидання біта PE регістра CR0 виконується командою MOV:----------}

db $0f,$20,0c0h *{ MOV EAX,CR0 }*
and al,not 1
db $0f,22h,0c0h *{ MOV CR0,EAX }.*

Перехід на мітку **real** необхідний для очистки буфера передвиборки команд:

jmp real.

1.12. Особливості режиму “Unreal”

Крім реального та захищеного режиму процесор може працювати в так званому режимі “Unreal”, який є різновидом реального режиму. Оскільки при будь-якому доступі до сегменту пам’яті і в реальному, і у захищеному режимах базова адреса сегмента та значення його границі процесор бере з дескрипторного регістру (“тіньового” регістру), то стає зрозумілим - якщо у захищеному режимі в дескрипторному регістрі змінити значення базової адреси сегмента або його границі, або і те і інше (в реальному режимі це зробити не можна) і при цьому не відновити ці значення при повертанні до реального режиму на такі, що характерні для цього режиму (хоча фірма Intel рекомендує цього не робити), то в реальному режимі параметри сегменту пам’яті залишаться такими же, як у захищеному режимі. Це дозволяє зняти деякі обмеження реального режиму, наприклад, обмеження на розмір сегмента (64 КБ) та його базову адресу (1 МБ). Такий стан процесора називають режимом “Unreal”.

Для спрощення переходу з реального режиму до режиму “Unreal” розроблена процедура **set_unr(s_reg:byte; limit, base:longint; byte_6:byte)**, що міститься в модулі **PROT**, параметри якої мають таке значення:

– **s_reg –** номер сегментного регістра (ES – 0, DS – 3, FS – 4, GS – 5);

– limit – границя сегмента;

– base – базова адреса сегмента;

– byte_6 – значення байта 6 дескриптора сегмента з урахуванням тільки чотирьох старших розрядів, насамперед бітів **G** і **D/B**.

Приклади застосування процедури **set_unr** для встановлення режиму “Unreal” та відновленню реального режиму дивиться в підрозділі 1.13.

1.13. Робота МП у реальному режимі та режимі “Unreal”

Після повернення у реальний режим треба відновити вміст сегментних регістрів DS, SS, ES та показчику стека SP тими значеннями, які вони мали до переходу до захищеного режиму:

```
real:
        xor ax,ax
        mov ds,ax
        mov ds, [$60*4]                    { DS, }
        mov ss, real_ss                    { SS, }
        mov es, real_es                    { ES ma }
        mov sp, real_sp                    { SP }.
```

Оскільки на час роботи процесора у захищеному режиму всі апаратні переривання (масковані та немасковані) були заборонені, після повернення в реальний режим їх треба дозволити.

Дозвіл маскованих апаратних переривань здійснюється за допомоги команди STI (set interrupt-enable flag), яка встановлює прапор IF в регістрі EFLAGS

sti,

а дозвіл немаскованих апаратних переривань здійснюється шляхом занесення в порт $70 значення $D:

```
mov al,0dh
out 70h,al
```

.

Крім того робиться скидання стану клавіш-перемикачів (які могли бути натиснути у захищеному режимі та при переході до реального режиму):

mem[$40:$17]:=0;.

Після повернення в реальний режим програма **P_MODE** виводить на екран:

– для комп’ютера з МП моделі 80286 – повідомлення
«**Возврат из защищенного режима выполнен по сбросу МП**»;

– для даного комп’ютера – значення розміру пам’яті, яке було отримане при роботі у захищеному режимі і збережене в змінній **m_size** (рис. 1.9).

Рисунок 1.9 – Визначення розміру пам’яті у захищеному режимі

У програмі **P_MODE** також був реалізований режим “Unreal” шляхом виклику процедури **set_unr** з такими параметрами:

set_unr(5,$ffeff,1024*1024,$80).

Тим самим з сегментним регістром даних GS пов’язується сегмент пам’яті, який має такі параметри:

– границя дорівнює FFEFFFFFh (4 ГБ - 1 МБ - 1);

– базова адреса дорівнює 1024*1024 (1 МБ).

Після цього викликається процедура **know_mem_size**, яка вже визначала розмір встановленої на комп’ютері пам’яті у захищеному режимі. Робота процедури в режимі “Unreal” підтверджує дані, отримані у захищеному режимі (рис. 1.10).

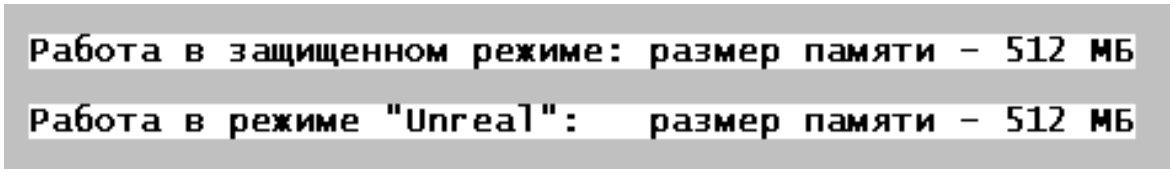

Рисунок 1.10 – Визначення розміру пам’яті в режимах:
захищеному та “Unreal”

Запуск процедури **know_mem_size** в реальному режимі привів би до зависання комп’ютера.

Для повертання з режиму “Unreal”до реального режиму (це необхідно зробити для коректного завершення програми) здійснюється виклик процедури **set_unr** з параметрами, які є допустимими для реального режиму (границя – FFFFFh, базова адреса – 0):

set_unr(5,$ffff,0,0);.

Додамо, що оскільки в режимі “Unreal” використовується сегмент пам’яті з такими же самими параметрами, що і сегмент розширеної пам’яті у захищеному режимі, то в такому випадку можна було би дещо скоротити програму, а саме: вилучити з неї відновлення регістру GS при повертанні до реального режиму і відповідно не застосовувати процедуру **set_unr** для установки режиму “Unreal”, бо він фактично вже діє.

1.14. Опис модуля PROT

Модуль **PROT** розроблений з метою зменшення ємності вхідного тексту програм, що реалізують різні аспекти функціонування мікропроцесора у захищеному режимі, і містить описи тих констант, типів змінних, змінних, процедур і функцій, що використовуються в програмах **P_MODE, P_INT, P_TASK** та **P_PAGE**.

Модуль **PROT** містить опис таких констант:

– завдання значень бітів і полів байтів доступу: **present, nosys, exe, down, wr_ena , rd_ena;**

– завдання значень байтів доступу коду, даних, стеку, обробників переривання (16- та 32-розрядних), обробників пастки (16- та 32-розрядних), TSS (16- та 32-розрядних), шлюзів виклику (16- та 32-розрядних): **acc_code, acc_data, acc_stack, acc_int_16, acc_int, acc_trap_16, acc_trap, acc_TSS_16, acc_TSS, acc_call_16, TSS, acc.**

Модуль **PROT** містить опис таких типів змінних:

t_gdt: структура дескриптора GDT;

t_idt: структура дескриптора IDT;

t_dtr: структура регістрів GDTR та IDTR;

t_tss_386: структура TSS для МП 80386 і наступних моделей МП;

Модуль **PROT** містить опис таких змінних: **gdt, idt, gdtr, idtr, idtr_r, ofs_end_prot, sel_prot, cs_PROT, real_cs, real_ss, real_es, real_sp, excep, ofs_ret, real_ss, real_es, real_sp, semaf, cpu_type, scan, excep**

Модуль **PROT** містить опис таких функцій:

hex: видача значень в 16-річному вигляді.

Модуль **PROT** містить опис таких процедур:

init_gdt: формування дескрипторів GDT;

init_gdtr: формування змісту GDTR;

init_idt: формування дескрипторів IDT;

init_idtr_p: формування змісту IDTR для роботи у захищеному режимі;

init_idtr_r: формування змісту IDTR для роботи в реальному режимі;

pic: програмування 1- і 2-го контролерів переривань для роботи в реальному та захищеному режимах;

init_tss_386: формування TSS для МП 80386 і наступних моделей МП;

set_unr: установлення режиму "Unreal";

exc_00-exc_18: оброблювачі виключень 0-18;

PIC_1, PIC_2: оброблювачі-заглушки апаратних маскованих переривань;

keyb: обробник переривання від клавіатури;

int_30h: обробник програмного переривання з номером 30h: виведення даних на екран в текстовому режимі;

int_32h: обробник програмного переривання з номером 32h: виведення на екран налагоджувальної інформації (номеру виключення, що виникло при виконанні програми та ін.).

1.15 Текст програми P_MODE

```
{==========Организация работы МП в защищенном  режиме=========}
program p_mode;
{--------------Модуль PROT предназначен для поддержки програм,--------------}
{--------------работающих в защищенном режиме:-----------------------------}
{----------------а) содержит необходимые константы, типы переменных,-----}
{-----------переменные, процедуры и функции;-----------------------------------}
{----------------б) создает базовые таблицы GDT и IDT----------------------------}
   uses crt,
        prot,
{-------------------------------Модуль CPU предназначен-------------------------------}
{-----------------для определения параметров микропроцессора------------------}
        cpu;
   label
        real;                                  { Метка возврата в реальный режим }
   const
        s:string='Работа в защищенном режиме. Размер памяти: ';
   var
        cpu_fam:byte;                                    { Номер семейства МП }
        ofs_real:word;                                   { Смещение метки real }
        m_size:longint;                                  { Размер оперативной памяти }
{------------Процедура out_screen выводит на экран одну из строк:-------------}
```

```
{---------- "Работа в защищенном режиме" (для МП молели 80286);--------}
{----------"Работа в защищенном режиме. Размер памяти: "----------------}
{----------------------------(для МП последующи моделей)----------------------------}

   procedure out_screen;assembler;
       asm
            mov ax,3                                  { AX - номер строки экрана }
            mov bx,2                                  { BX - номер столбца экрана }
            xor cx,cx
            cmp cpu_fam,2
            jnz @3
            mov cl,26           { Длина строки 'Работа в защищенном режиме'
}
            jmp @4
      @3:
            mov cl, [offset s]                                  { Длина строки s }
@4:
            mov si,offset s+1                  { DS:SI - адрес строки в памяти }
            mov dl,80                              { DL - число колонок экрана }
            mul dl
            add ax,bx
            shl ax,1
            mov di,ax                        { ES:DI - адрес строки в видеопамяти }
            mov ah, 1eh                          AH - видеоатрибут строки }
   @wxy:
            lodsb                             { Чтение символа строки из памяти }
            stows                               { Вывод символа строки на экран }
            loop @wxy
         end;{out_screen}
{--------------------Процедура know_mem_size определяет объем--------------------}
{---------установленной в компьютере оперативной памяти (в байтах)---------}
   procedure know_mem_size;assembler;
       asm
            db 66h                                                { Очистка EBX }
            xor bx,bx                      { В EBX - адрес текущей ячейки памяти }
            db 66h                                                { Очистка ECX }
            xor cx,cx                  { ECX - счетчик циклов обращения к памяти }
            mov al,0aah                      {  В AL - код, записываемый в память }
    @5:
            db 65h                                  { префикс GS: выбор сегмента GS }
            db 67h                                { префикс AS: разрядность адреса - 32 }
            db 8ah,13h                          { MOV DL,[EBX] - чтение данных ячейки }
            db 65h,67h,88h,03h                    { GS:AS:MOV [EBX],AL - запись кода }
            db 65h,67h,8ah,23h                   { GS:AS:MOV AH,[EBX] - чтение кода }
            cmp ah,al          { Сравнение записываемого и считываемого кода }
```

```
            db 65h,67h,88h,13h          { GS:AS:MOV [EBX],DL - запись данных }
            jnz @6                      { Если коды не равны, т.е. адрес превысил }
                                        { объем памяти компьютера - закончить анализ памяти }
            db 66h,43h                  { INC EBX - увеличение на 1 }
            db 67h                      { адреса текущей ячейки памяти }
            loop @5                     { и переход к ее анализу }
        @6:
            db 66h                      { Запись в m_size из EBX значения }
            mov word ptr m_size,bx      { объема памяти в байтах }
        end;{know_mem_size}
{=================ОСНОВНАЯ ПРОГРАММА=================}
  begin
{=================Работа в реальном режиме=================}
        clrscr;
{---------------------Определение параметров микропроцессора---------------------}

        get_cpu(0,cpu_fam);             { В cpu_fam - номер семейства МП }
{--------------------------Добавление к базовой таблице GDT--------------------------}
{---------------------дескриптора сегмента расширенной памяти---------------------}

        init_gdt(mem_sel,$ffefffff,1024*1024,acc_data,0);
{--------------Формирование данных регистра GDTR и его загрузка--------------}
{----------(параметром является селектор последнего из используемых----------}
{-----------------------------в программе дескрипторов GDT)-----------------------------}

        init_gdtr(mem_sel);
        if cpu_fam=2 then begin
{==========Формирование данных для возврата в реальный режим=======}
{===============микропроцессора 80286 по сбросу===============}

{------------Занесение в КМОП-память по адресу 0Fh байта состояния----------}
{----------отключения (значение 0Ah байта обеспечивает при сбросе МП--------}
{-------------------переход по адресу в ячейках 40h:67h и 40h:69h)-------------------}

            port[$70]:=$F;
            port[$71]:=$A;
{------------Определение смещения точки возврата в реальный режим-----------}

            asm mov ofs_real,offset real end;
{--------------------Занесение в ячейки 40h:67h и 40h:69h адреса--------------------}
{------------(смещение и сегмент) точки возврата в реальный режим------------}

            memw[$40:$67]:=ofs_real;
            memw[$40:$69]:=Cseg;
        end;
```

```
{----------------------Запрет внешних аппаратных прерываний---------------------}

      asm cli end;                                              { маскируемых }
      port[$70]:=$80;                                          { и немаскируемых }
{========Сохранение содержимого сегментных регистров и SP========}

      memw[0:4*$60]:=Dseg;                                        { DS, }
      real_ss:=Sseg;                                               { SS, }
      asm mov real_es,es end;                                      { ES }
      real_sp:=SPtr;                                              { и SP }
{==================Переход в защищенный режим================}

      asm
            cmp cpu_fam,2                          { Анализ разрядности МП }
            jnz @32
{----------Переход в защищенный режим для 16-разрядного МП 80286----------}
{--------------------путем  установки  бита PE в регистре MSW--------------------}
            db 0fh,01h,0e0h                                       { SMSW AX }
            or ax,1
            db 0fh,01h,0f0h                                       { LMSW AX }
            jmp @prot
{-----------------------------Переход в защищенный режим-----------------------------}
{----------------------для МП 80386 и МП последующих моделей----------------------}
{---------------путем установки бита PE в регистре управления CR0-------------}
    @32:
            db 0fh,20h,0c0h                                       { MOV EAX,CR0 }
            or al,1
            db 0fh,22h,0c0h                                       { MOV CR0,EAX }
{----------------------Переход для очистки очереди команд МП----------------------}
            jmp @prot
{----------------------Загрузка сегментных регистров SS, ES и GS-------------------}
{---------------------------соответствующими селекторами-------------------------}

   @prot:
            mov ss,stack_sel
            mov es, text_sel
            db 8eh,2eh                                            { MOV GS,mem_sel }
            dw mem_sel
{================Работа в защищенном режиме==============}

{---------------------------------Вывод на экран строки s--------------------------------}
{----------------------путем прямого обращения к видеопамяти---------------------}

            call out_screen
            cmp cpu_fam,2
```

```
            jnz @mem
{----------Для МП 80286 возврат в реальный режим по команде----------}
{----------контроллера клавиатуры, выполняющей сброс МП----------}

            mov al,0feh                          { Команда сброса }
            out 64h,al                           { микропроцессора }
   @wait:                                { Ожидание во время сброса МП }
            hlt
            jmp @wait
{----------Определение размера оперативной памяти компьютера----------}
{----------для МП 80386 и МП последующих моделей----------}

@mem:
            call know_mem_size
{=======Подготовка к возврату в реальный режим по команде MOV======}
{============для МП 80386 и МП последующих моделей============}

{----------Восстановление параметров сегментов SS, ES и GS----------}
{----------для работы в реальном режиме (Limit=0FFFFh, ED=0, W=1)---------}

            mov ss,real_sel
            mov es,real_sel
            db 8eh,2eh                           { MOV GS,real_sel }
            dw real_sel

{----------Возврат в реальный режим по команде MOV----------}
{----------путем сброса бита PE в регистре управления CR0----------}

            db 0fh,20h,0c0h                      { MOV EAX,CR0 }
            and al,not 1
            db 0fh,22h,0c0h                      { MOV CR0,EAX }
            jmp real
{============Работа после возврата в реальный режим============}

{----------Восстановление регистров после----------}
{----------возврата в реальный режим----------}

      real:
            xor ax,ax
            mov ds,ax
            mov ds,[4*$60]                                { DS, }
            mov ss,real_ss                                { SS, }
            mov es,real_es                                { ES }
            mov sp,real_sp                               { и SP }
{----------Разрешение внешних аппаратных прерываний----------}
```

```
            sti                                          { маскируемых }
            mov al,0dh
            out 70h,al                                   { и немаскируемых }
        end;
        mem[$40:$17]:=0;          { Сброс состояния клавиш-переключателей }
        gotoXY(46,4);
        if cpu_fam=2
        then writeln('Возврат из защищенного режима выполнен',
        'по сбросу МП ')
        else begin
            writeln(m_size shr 20+1,' МБ');
            gotoXY(5,6);
            write('Работа в режиме "Unreal": ',
            'размер памяти:  ');
{------------------------------Работа в режиме "Unreal"------------------------------}

            set_unr(5,$ffeff,1024*1024,$80);
            know_mem_size;
            writeln(m_size shr 20+1,' МБ');
{--------------------------Восстановление реального режима-------------------------}

            set_unr(5,$ffff,0,0);
        end;
        readkey
    end.
```

1.16. Текст модуля PROT

```
{---------Модуль содержит константы, типы переменных, переменные,---------}
{-------------процедуры и функции для работы в защищенном режиме----------}
unit prot;
  interface
    const
        aa:longint=$123;
        hex_tabl:array[0..15] of char='0123456789ABCDEF';
        scan:byte=0;              { Значение скан-кода нажатой клавиши }
        save_on:byte=0;                          { Экран не сохранен }
{==============Селекторы дескрипторов сегментов==============}

        code_sel          =$08;                    { кода программы, }
        PROT_sel          =$10;                  { кода модуля PROT, }
        stack_sel:word    =$18                                { стека }
        real_sel :word    =$20;          { данных реального режима, }
```

```
        data_sel :word      =$28;                          { данных, }
        text_sel:word =$30;                    { видеопамяти текст. режима }
        graph_sel:word      =$38;              { видеопамяти граф. режима }
        mem_sel  :word      =$40;                  { расширенной памяти, }
        CONF_sel            =$48;                   { кода модуля CONF, }
        USER_sel            =$50;                   { кода модуля USER, }
        DEB_sel             =$58;                    { кода модуля DEB, }
        data3_sel:word      =$60;                  { данных пользователя, }
{==============Селекторы дескрипторов TSS задач==============}
        main_sel            =$68;                                { MAIN, }
        v_main_sel:word=$68;                                     { MAIN, }
        time_sel      =$70;                                  { TIME_TASK, }
        color_sel           =$78;                                { COLOR_TASK, }
        sound_sel           =$80;                                { SOUND_TASK, }
        duration_sel  =$88;                                  { DURATION_TASK }
        date_sel      =$90;                                  { DATE_TASK, }
        keyb_sel            =$98;                                { KEYB_TASK }
        user_check_sel      =$A0;                          { USER_CHECK_TASK }
        vm86_sel            =$A8;                          { SWITCH_VM86_TASK }
        stack_TSS_sel:word        =$B0;                          { стека TSS, }
{------------------------Селектор дескриптора шлюза вызова-------------------------}
        gate_sel      =$B8;
{-------------------------------Селектор дескриптора LDT--------------------------------}
        LDT_sel:word        =$C0;
{==============Биты и байты доступа сегментов==============}

{--------------------------------Биты доступа сегментов---------------------------------}
        present       =$80;          { P=1 :          сегмент есть в памяти }
        no_sys        =$10;          { S=1:           сегмент несистемный }
        exe           =$08;          { E=1:           сегмент выполняемый }
        down          =$04;          { ED=1:          сегмент расширяется вниз }
        wr_ena        =$02;          { W=1:           разрешена запись в сегмент }
        rd_ena        =$02;          { R=1:   разрешено чтение из сегмента }
{-------------------------------Байт доступа сегмента кода----------------------------}

        acc_code =present OR no_sys OR exe OR rd_ena;
{-----------------------------Байт доступа сегмента данных----------------------------}

        acc_data=present OR no_sys OR wr_ena;
{----------------------------Байт доступа сегмента стека ----------------------------}

        acc_stack=present OR no_sys OR wr_ena OR down;
{---------------------Байты доступа обработчиков прерывания---------------------}

        acc_int_16=present OR $06;                          { 16-разрядного }
```

```
        acc_int   =present OR $0E;                    { 32-разрядного }
{----------------------Байты доступа обработчиков ловушки-------------------------}

        acc_trap_16=present OR $07;                   { 16-разрядного }
        acc_trap   =present OR $0F;                   { 32-разрядного }
{-------------------------------------Байты доступа TSS-----------------------------------}

        acc_TSS_16=present OR  $01;                   { 16-разрядного }
        acc_TSS   =present OR  $09;                   { 32-разрядного }
{-----------------------------Байты доступа шлюзов вызова----------------------------}

        acc_call_16=present OR $04;                   { 16-разрядного }
        acc_call   =present OR $0C;                   { 32-разрядного }

{----------------------------Байт доступа шлюза задачи-------------------------------}

        acc_task=present OR $05;
{-----------------------------------Байт доступа LDT-------------------------------------}
        acc_LDT=present OR 2;
    type
{-----------------------Структура дескриптора таблицы GDT----------------------}
        t_gdt=record
            lim_l,                          { граница сегмента (биты 15-0) }
            base_l :word;                   { базовый адрес сегмента (биты 15-0) }
            base_h,                         { базовый адрес сегмента (биты 23-16) }
            acc,                                                { байт доступа }
            lim_h,                          { G,D,0,X,граница сегмента (биты 19-16) }
            base_hh:byte                    { базовый адрес сегмента (биты 31-24) }
        end;
{---------------------Структура данных регистров GDTR и IDTR--------------------}

        t_dtr=record
            lim  :word;                                         { граница и }
            base :longint;                          { базовый адрес таблицы }
        end;

{--------------Структура дескриптора таблицы IDT и шлюза вызова--------------}
        t_idt=record
            ofs_l,                                  { смещение  (биты 15-0) }
            sel   :word;                                        { селектор }
            par,                                    { число параметров }
            acc   :byte;                                        { байт доступа }
            ofs_h :word                             { смещение (биты 31-16) }
        end;
{--------------Структура TSS для МП 80386 и последующих моделей--------------}
```

```
        t_tss_386=record
            link,                                   { селектор возврата }
{-----------------------------------Указатели стека-----------------------------------}

            esp0,ss0,                                       { кольца 0 }
            esp1,ss1,                                       { кольца 1 }
            esp2,ss2,                                       { кольца 2 }
{-------------------------------Регистры микропроцессора-------------------------------}
            cr3,eip,eflags,eax,ecx,edx,ebx,esp,ebp,esi,edi,es,cs,ss,ds,fs,gs,ldtr:
            longint;
            bit_t,                                  { бит ловушки задачи }
            adr_BKVV:word;                          { смещение БКВВ }
            sys_inf:string;                 { поле для системной информации }
            BKVV: array[0..7] of byte;                              { БКВВ }
            byte_end:byte;                  { байт завершения TSS }
        end;
{--------------Структура TSS для задачи виртуального режима VM86-----------}

        t_tssVM=record
            link,                                   - { Селектор возврата }
{-----------------------------------Указатели стека-----------------------------------}

            esp0,ss0,                                       - { кольца 0 }
            esp1,ss1,                                       - { кольца 1 }
            esp2,ss2,                                       - { кольца 2 }
{-------------------------------Регистры микропроцессора-------------------------------}

            cr3,eip,eflags,eax,ecx,edx,ebx,esp,ebp,esi,edi,es,cs,ss,ds,fs,gs,ldtr:
            longint;
            bit_t,                                  { Бит ловушки задачи }
            adr_BKVV:word;                          { Смещение БКВВ }
{------------Битовая карта перенаправления программных прерываний----------}
          BK_VM86:array[0..31] of byte;
{-----------------------Битовая карта ввода-вывода (БКВВ)-----------------------}
          BKVV:array[0..1023*8] of byte;
          byte_end:byte;                            { Байт завершения TSS }
        end;
    var
        screen:                                 { Массив для сохранения значений }
        array[0..1999] of word;                         { экрана видеопамяти }
        di_,cx_:word;                   { Переменные для хранения значений }
        dh_,dl_:byte;                           { регистров DI, CX, DH та DL }
        gdt:array[0..16] of t_gdt;                              { Таблица GDT }
        idt:array[0..$33] of t_idt;                             { Таблица IDT }
```

```
        gdtr,                                          { Содержимое GDTR }
        idtr,          { Содержимое IDR для работы в защищенном режиме }
        idtr_r              { Содержимое IDTR для работы в реальном режиме }
        :t_dtr;
        s:string;
        eip_:longint;
        ofs_end_prot,                                          { Смещение }
        sel_end_prot,                                { и селектор метки end_prot }
{----------------------Переменные для хранения значений регистров-------------------}
{--------------------------------------реального режима:---------------------------------}
        real_cs,                                                    { CS, }
        real_ds,                                                    { DS, }
        real_ss,                                                    { SS, }
        real_es,                                                   { ES и }
        real_sp:word;        { SP }
        mode,              { Режим работы МП: 0/1 (реальный/защищенный) }
        left_shift:byte;                          { Состояние клавиши Left Shift:  }
                                                        { 0/1 (отпущена/нажата) }
{============Объявление функций и процедур модуля PROT===========}

    function hex(p:longint):string;
    procedure wait_;
    procedure get_time;
    procedure init_gdt(sel:byte;limit,base:longint;acc_,byte_6:byte);
    rocedure init_gdtr(sel_GDT:word);
    procedure init_idt(i:byte;ofs_:longint;sel_:word;acc_:byte);
    procedure init_idtr;
    procedure pic(mode:byte);
    procedure init_tss (var tss:t_tss; cs,ds,es,ip,sp:word);
    procedure set_unr(base,limit:longint;kseg,byte_6h:byte);
    procedure save_scr;
    procedure exc_00;
    procedure exc_01;
    procedure exc_02;
    procedure exc_03;
    procedure exc_04;
    procedure exc_05;
    procedure exc_06;
    procedure exc_07;
    procedure exc_08;
    procedure exc_10;
    procedure exc_11;
    procedure exc_12;
    procedure exc_13;
    procedure exc_14;
```

```
    procedure exc_16;
    procedure exc_17;
    procedure exc_18;
    procedure PIC_1;
    procedure PIC_2;
    procedure keyb;
    procedure int_30h;
    procedure int_32h;
    procedure int_33h;

  implementation

    const
        s_err:string='Runtime error: исключение ';
        s_reg:array[0..7] of string=
        ('  EAX  ','  ECX  ','  EDX  ','  EBX  ',
        '  ESP  ','  EBP  ','  ESI  ','  EDI  ');
        s_sreg:array[0..6] of string=('  EIP  ',' CS ',' SS '
        ,' ES ',' DS ',' FS ',' GS ');
        s_CR0:array[0..1] of string=
        ('PCN          A W          NETEMP',
        'GDW  резерв  M P  резерв  ETSMPE');
        s_CR0n:string='CR0:';
        s_CR4n:string='CR4:';
        s_CR4:array[0..2] of string=
        ( '                    PPMPP TPV',
        ' Зарезервировано      CGCASDSVM',
        '                    EEEEEEDIE');
        s_EFLAGSn:string='EFLAGS:';
        s_EFLAGS:array[0..2] of string=
         ('          VV                ',
        '         IIIAVR NIPODITSZ A P C',
              '  резерв  DPFCMF0TOLFFFFFF0F0F1F');
    var i:byte;
{==============Описание функций и процедур модуля PROT==============}

{-----------------------Преобразование данных в 16-ричную форму----------------------}

    function hex(p:longint):string;
      var   s:string;
      begin
        s:='h';
        repeat
        s:=hex_tabl[p and $f]+s;
        p:=p shr 4
```

```
            until p=0;
            hex:=s
          end;{hex}
{------------Ожидание нажатия клавиши (в scan - скан-код нажатия)------------}
      procedure wait_;assembler;
          asm
              mov scan,0
        @w:
              cmp scan,0
              jz @w
              test scan,80h
              jz @w
          end;
      procedure get_time;assembler;
          asm
              mov ax,100h
              mov bx,4800h
              int 30h                { Установка маркера в точку экрана (72,0) }
              mov cx,3               { Три цикла определения и вывода времени }
              mov al,4                                  { в формате чч:мм:сс }
@2:
              push cx
              push ax
              out 70h,al
              in al,71h                { В AL - значение часов/минут/секунд }
              mov cl,1dh                                { Видео-атрибут }
              mov dl,al
              mov ax,300h
              int 30h            { Вывод на экран значения часов/минут/секунд }
              pop cx
              pop ax
              cmp cx,0
              jz @1
              push ax
              push cx
              mov ah,2
              sub di,2
              mov cl,1dh
              mov dl,3ah
              int 30h                                { Вывод на экран символа ":" }
              pop ax
              sub al,2
              pop cx
              loop @2
        @1:
```

```
            db 66h          { Использовался  32-разрядный вызов; команды RETF }
        end;                  { нет, так как она подставляется транслятором }
{----------------------Формирование дескриптора таблицы GDT----------------------}

    procedure init_gdt(i:byte;limit,base:longint;acc_,byte_6h:byte);
        begin
            with gdt[i] do begin
                lim_l :=limit;
                base_l :=base;
                base_h :=base shr 16;
                acc    :=acc_;
                lim_h  :=limit shr 16 or byte_6h;
                base_hh:=base shr 24;
            end
        end; {init_gdt}
{-----------------Формирование данных регистра GDTR и его загрузка---------------}

    procedure init_gdtr(sel_GDT:word);
        begin
            gdtr.lim:=sel_GDT+7;
            gdtr.base:=longint(seg(gdt)) shl 4+ofs(gdt);
            asm
                db 0fh,01h,16h                                  { LGDT gdtr: }
                dw gdtr              { загрузка атрибутов GDT в GDTR из gdtr }
            end
        end; {init_gdtr}
{----------------------Формирование дескриптора таблицы IDT----------------------}

    procedure init_idt(i:byte;ofs_:longint;sel_:word;acc_:byte);
        begin
            with idt[i] do begin
                ofs_l:=ofs_;
                sel:=sel_;
                par:=0;
                acc:=acc_;
                ofs_h:=ofs_ shr 16;
            end
        end; {init_idt}
{---------------Сохранение данных регистра IDTR реального режима----------------}
{--------------и формирование данных регистра IDTR с его загрузкой--------------}
{-----------------------------для работы в защищенном режиме-----------------------}

    procedure init_idtr;
        begin
            asm
```

```
            cl                                  { Запрет внешних маскируемых }
            mov al,80h                          { и внешних немаскируемых }
            out 70h,al                          { прерываний }
            db 0fh,1,0eh                        { SIDT idtr: }
            dw idtr_r                { Сохранение атрибутов IDT в idtr_r }
        end;
        idtr.lim:=sizeof(idt)-1;
        idtr.base:=longint(seg(idt)) shl 4+ofs(idt);
        asm
            db 0fh,01h,1eh                      { LIDT idtr: }
            dw idtr                { Загрузка атрибутов IDT в IDTR из idtr }
end;
end;{init_idtr}

{------------------------Программирование ведущего и ведомого----------------------}
{------------------------контроллеров прерываний  для работы-----------------------}
{----------в реальном (mode=0) и защищенном (mode=1) режимах----------}

    procedure pic(mode:byte);
        var  k1,k2:byte;
        begin
            if mode=0 then begin
                k1:=8;   k2:=$70
            end else begin
                k1:=$20; k2:=$28
            end;
            port[$20]:=$11;                     { 1-й ПКП: ICW1 }
            port[$21]:=k1;                      { 1-й ПКП: ICW2 }
            port[$21]:=4;                      { 1-й ПКП: ICW3 }
            port[$21]:=1;                       { 1-й ПКП: ICW4 }
            port[$a0]:=$11;                     { 2-й ПКП: ICW1 }
            port[$a1]:=k2;                      { 2-й ПКП: ICW2 }
            port[$a1]:=2;                       { 2-й ПКП: ICW3 }
            port[$a1]:=1;                       { 2-й ПКП: ICW4 }
        end;{pic}
{----------------------Формирование TSS для 32-разрядных МП-----------------------}
    procedure init_tss(var tss:t_tss; cs,ds,es,ip,sp:word);
        begin
            tss.cs:=cs;
            tss.ds:=ds;
            tss.es:=es;
            tss.ss:=ds;
            tss.eip:=ip;
            tss.esp:=sp;
            tss.eflags:=$200;
```

```
            tss.bit_t:=0;
            tss.byte_end:=$ff
        end;{init_tss}
{---------------------------Для работы в режиме "Unreal"----------------------------}
{---------для заданного сегментного регистра указываются параметры-------}
{-------связанного с ним сегмента памяти, которые могут отличаться------}
{---------от параметров реального режима: граница сегмента и его----------}
{----------базовый адрес могут быть увеличены до величины 4 ГБ-1-----------}
    procedure set_unr(s_reg:byte;limit,base:longint;byte_6:byte);
        begin
{-----------------------------Формирование таблицы GDT-----------------------------}

                                                        { нуль-дескриптор: }
            init_gdt(0,0,0,0,0);
                                                    { дескриптор сегмента данных: }
            init_gdt(8,limit,base,$92,byte_6);
{--------------------Создание данных и загрузка регистра GDTR---------------------}
            init_gdtr(15);
{-----------------------------------Запрет прерываний-----------------------------------}
            asm
                mov al,80h                                      { маскируемых }
                out 70h,al                                    { и немаскируемых }
{-----------------------------Переход в защищенный режим-----------------------------}

                db 0fh,20h,0c0h                               { MOV EAX,CR0 }
                or al,1
                db 0fh,22h,0c0h                               { MOV CR0,EAX }
{----------------Загрузка селектора в заданный сегментный регистр---------------}
                mov ax,8                                    { Селектор сегмента }
                cmp s_reg,0                                                 { ES? }
                jnz @3
                mov es,ax                                     { MOV ES,AX }
                jmp @k
                @3: cmp s_reg,3                                             { DS? }
                jnz @4
                mov ds,ax                                     { MOV DS,AX }
                jmp @k
            @4:
                cmp s_reg,4                                                 { FS? }
                jnz @5
                db 8eh,0e0h                                   { MOV FS,AX }
                jmp @k
            @5:
                cmp s_reg,5                                                 { GS? }
                jnz @k
```

```
                db 8eh,0e8h                                    { MOV GS,AX }
{------------------------------Возврат в реальный режим-------------------------------}

          @k:
                db 0fh,20h,0c0h                                { MOV EAX,CR0 }
                and al,not 1
                db 0fh,22h,0c0h                                { MOV CR0,EAX }
 {---------------------------Разрешение прерываний---------------------------------}
                sti                                            { маскируемых }
                mov al,0Dh
                out 70h,al                                     { и немаскируемых }
            end;
        end;{ set_unr }
    procedure save_scr;
        var
            save_scr:string;
            ekr:file of word;
            k:integer;
        begin
            if boolean(save_on) then begin
                writeln('Введите имя массива для сохранения экрана');
                readln(save_scr);
                if save_scr<>'' then begin
                    save_scr:='EKRAN/'+save_scr;
                    assign(ekr,save_scr);
                    rewrite(ekr);
                    for k:=0 to 1999 do write(ekr,screen[k]);
                end;
            end; { asm  mov ah,1 end;}
        end;{ save_scr }
{----------------------Обработчики исключений 0-8, 10-14, 16-18----------------------}
{-------------(процедуры exc_00-exc_08, exc_10-exc_14, exc_116-exc_18)-----------}
{-----------------выводят на экран номер исключения, адрес команды---------------}
{--------------------------и завершают выполнение программы-----------------------}

    procedure exc_00;assembler;                    { Обработчик исключения 0: }
        asm                                                  { деление на 0 }
            mov ah,1                               { Функция 1 прерывания 32h: }
            mov dx,100h                  { вывод на экран номера исключения (0) }
            int 32h                                        { и адреса команды }
            db 0ffh,2eh                             { Межсегментный переход }
            dw ofs_end_prot                                { на метку end_prot: }
        end;{ exc_00}
    procedure exc_01;assembler;                    { Обработчик исключения 1: }
        asm                                  { - при TF=1 регистра EFLAGS;   }
```

```
            mov ah,1                    { - при T=1 сегмента TSS;  }
            mov dx,101h               { - по контрольным точкам программы; }
            int 32h                     { - по контрольным точкам данных;  }
            db 0ffh,2eh                 { - по контрольным точкам УВВ; }
            dw ofs_end_prot             { - при защите регистров отладки  }
        end;{ exc_01}
    procedure exc_02;assembler;         { Обработчик исключения 2: }
        asm                           { немаскируемое прерывание (NMI) }
            mov ah,1
            mov dx,102h
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
        end;
    procedure exc_03;assembler;         { Обработчик исключения 3: }
        asm                                       { по команде INT 3 }
            mov ah,1
            mov dx,103h
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
        end;
    procedure exc_04;assembler;         { Обработчик исключения 4: }
        asm                           { по команде INTO при OF=1 }
            mov ah,1
            mov dx,104h
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
        end;
    procedure exc_05;assembler;         { Обработчик исключения 5: }
        asm                           { выход за пределы диапазона }
            mov ah,1                  { при выполнении команды BOUND }
            mov dx,105h
            ont 32h
            db 0ffh,2eh
            dw ofs_end_prot
        end;
    procedure exc_06;assembler;         { Обработчик исключения 6: }
        asm                                 { неверный код операции }
            mov ah,1                                { или адресации }
            mov dx,106h
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
```

```
    end;
  procedure exc_07;assembler;          { Обработчик исключения 7: }
    asm                                { недоступно устройство FPU }
      mov ah,1
      mov dx,107h
      int 32h
      db 0ffh,2eh
      dw ofs_end_prot
    end;
  procedure exc_08;assembler;          { Обработчик исключения 8: }
    asm                                { двойная ошибка }
      mov ah,1
      mov dx,108h
      int 32h
      db 0ffh,2eh
      dw ofs_end_prot
    end;
  procedure exc_10;assembler;          { Обработчик исключения 10: }
    asm                                { недоступен TSS }
      mov ah,1
      mov dx,10Ah
      int 32h
      db 0ffh,2eh
      dw ofs_end_prot
    end;
  procedure exc_11;assembler;          { Обработчик исключения 11: }
    asm                                { недоступен сегмент }
      mov ah,1
      mov dx,10Bh
      int 32h
      db 0ffh,2eh
      dw ofs_end_prot
    end;
  procedure exc_12;assembler;          { Обработчик исключения 12: }
    asm                                { ошибка доступа к сегменту стека }
      mov ah,1
      mov dx,10Ch
      int 32h
      db 0ffh,2eh
      dw ofs_end_prot
    end;
  procedure exc_13;assembler;          { Обработчик исключения 13: }
    asm                                { нарушение общей защиты }
{----------------------Анализ вызова обработчика прерывания 13:----------------------}
{-----------------это исключение 13 или возврат из режима VM8086?-------------}
```

```
            push ax
            push ds
            mov ax,28h
            mov ds,ax
            mov bp,sp
            mov ax, [bp+12]
            cmp ax,real_cs          { Если селектор CS в стеке является }
            pop ds                  { базовым адресом реального режима - }
            pop ax
            jnz @err
            db 0eah                 { тогда: переключение на задачу MAIN }
            dw 0                    { для выхода из режиима VM8086, }
            dw main_sel
  @err:
            mov ah,1                { иначе: вывод на экран сообщения о том, }
            mov dx,10Dh             { что возникло исключение 13 }
            int 32h
            db 0ffh,2eh             { и завершение программы }
            dw ofs_end_prot
      end;
procedure exc_14;assembler;         { Обработчик исключения 14: }
      asm                           { недоступна страница }
            mov ah,1
            mov dx,10Eh
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
      end;
procedure exc_16;assembler;         { Обработчик исключения 16: }
      asm                           { ошибка FPU при NE=1 регистра CR0 }
            mov ah,1
            mov dx,110h
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
      end;
procedure exc_17;assembler;         { Обработчик исключения 17: }
      asm                           { ошибка выравнивания данных }
            mov ah,1
            mov dx,111h
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
      end;
procedure exc_18;assembler;         { Обработчик исключения 18: }
```

```
        asm                         { ошибка функционирования узлов МП и МПС }
            mov ah,1
            mov dx,112h
            int 32h
            db 0ffh,2eh
            dw ofs_end_prot
        end;
{------------------Обработчики-заглушки аппаратных прерываний,------------------}
{--------------------поступающих на 1-й контроллер прерываний--------------------}

    procedure PIC_1;assembler;
        asm
            push ax
            mov al,20h
            out 20h,al              { Сброс бита регистра ISR 1-го ПКП }
            pop ax
            db 66h
            iret
        end;
{-----------------Обработчики-заглушки аппаратных прерываний,-----------------}
{-------------------поступающих на 2-й контроллер прерываний--------------------}

    procedure PIC_2;assembler;
        asm
            push ax-
            mov al,20h                  { Сброс бита регистра ISR }
            out 20h,al                                  { 1-го и }
            out 0A0h,al                 { и 2-го контроллеров прерываний }
            pop ax
            db 66h
            iret
        end;
    procedure keyb;assembler;           { Обработчик прерываний }
        asm                                             { от клавиатуры: }
            pusha
            mov al,20h
            out 20h,al
            in al,60h       { Чтение скан-кода нажатия/отжатия клавиши }
            cmp al,2Ah
            jnz @k1
            mov left_shift,1            { Клавиша "Left Shift" нажата }
            cmp al,0AAh
            jnz @k1
            mov left_shift,0            { Клавиша "Left Shift" отпущена }
            @k1:
```

```
            cmp left_shift,1              { Если нажаты клавиши "Left Shift" }
            jnz @k2
            cmp al,0C5h                   { и "Pause/Break", тогда выполняется }
            jnz @k2
            db 0ffh,2eh                   { межсегментный переход на метку end_prot }
            dw ofs_end_prot               {  для срочного завершения программы }
        @k2:
            cmp scan,0E0h                 { Если два первых скан-кода клавиши }
            mov scan,al
            jnz @k3                       { являются кодами E0h и 2Ah, }
            cmp al,2Ah                    {  т.е. нажата клавиша "Print Screen", - }
            jnz @k3                       { то установить признак сохранения }
            mov save_on,1                 { содержимого экрана на магнитном диске }
            jmp @end
        @k3:
            cmp al,0E0h                   { Если скан-код нажатой клавиши равен E0h }
            jnz @end
            cmp save_on,0                 { и сброшен признак сохранения экрана - }
            jnz @end
            mov ax,700h                   { с помощью функции 7 INT 30h сохранить }
            mov bx,0                      { текущее содержимое области экрана }
            mov dx,4F18h                  { с координатами: (0,0) и (79,24), }
            int 30h                       { т.е. весь экран в массиве screen }
      @end:
            popa
            db 66h
            iret
        end;{keyb}
{----------------------Обработчик программного  прерывания 30h----------------------}
{----------------------Вывод данных на экран в текстовом режиме----------------------}
procedure int_30h;assembler;
                                          { Назначение регистров:}
            { AH - номер функции          }
            { Al - номер подфункции       }
            { BL - номер строки экрана    }
            { BH - номер столбца экрана   }
            { DL/DX/EDX – значение данных (байта/слова/двойного слова) }
            { CL - видео-атрибут символа  }
            { SI - смещение строки символов }
            { DI - смещение видеопамяти   }
        asm
            cmp ah,1
            jz @f1
            cmp ah,2
            jz @f2
```

```
            cmp ah,3
            jz @f3
            cmp ah,4
            jz @f4
            cmp ah,5
            jz @f5
            cmp ah,6
            jz @f6
            cmp ah,7
            jz @f7
            cmp ah,8
            jz @f8
            jmp @end
{--------------Функция 1: установка маркера в заданную точку экрана-------------}
                                        { AL=0 - без изменения экрана          }
                                        { AL=1 - с очисткой до конца строки    }
                                        { AL=2 - с очисткой до конца экрана    }
                                        { AL=3 – с подсветкой маркера          }
      @f1:
            push ax
            push cx
            push dx
            push bx
            push si
            cmp al,3
            jz @13
            mov cx,0
            cmp al,0
            jz @11
            mov cl,80
            sub cl,bh                   { Число стираемых символов в строке }
            cmp al,1
            jz @11
            mov al,24
            sub al,bl
            mov dl,80
            mul dl
            add ax,cx
            mov cx,ax                   { Число стираемых символов на экране }
      @11:
            mov al,bl
            mov dl,80
            mul dl
            shr bx,8
            add ax,bx
```

```
            shl ax,1
            mov di,ax
            mov si,ax
            cmp cx,0
            jz @end
            mov ax,720h
    @wxy_:
            stosw
            loop @wxy_
            mov di,si
            jmp @end
            @13:
            mov al,bl
            mov dl,80
            mul dl
            shr bx,8
            add ax,bx                    { В AX - номер символа на экране }
            mov bx,ax
            mov dx,3d4h
            mov al,0fh
            out dx,al
            mov dx,3d5h
            mov al,bl
            out dx,al
            mov dx,3d4h
            mov al,0eh
            out dx,al
            mov dx,3d5h
            mov al,bh
            out dx,al
      @pop:
            pop si
            pop bx
            pop dx
            pop cx
            pop ax
            jmp @end
{-------------------------Функция 2: вывод символа на экран-------------------------}
       @f2:
```

```
                mov al,dl
                mov ah,cl
                stosw
                add di,2
                jmp @end
{----------------Функция 3: вывод данных на экран в 16-ричной форме---------------}
                                                { AL=0 - вывод байта            }
                                                { AL=1 -вывод слова             }
        @f3:                                     { AL=2 - вывод двойного слова  }
                lea bx,hex_tabl
                mov ah,cl
                cmp al,0
                jz @8
                cmp al,1
                jz @16
                add di,14
                mov cx,8
                jmp @1
        @8:
                add di,2
                mov cx,2
                jmp @1
        @16:
                add di,6
                mov cx,4
        @1:
                push cx
        @lp:
                mov al,dl
                and al,0fh
                db 66h
                shr dx,4
                xlat
                stosw
                sub di,4
                loop @lp
                pop cx
                shl cx,1
                add cx,4  {2}
                add di,cx
                jmp @end
{----------------------Функция 4: вывод строки символов на экран---------------------}
        @f4:
                mov ah,cl
                mov bx,si
```

```
                inc si
                mov cl,[bx]
                cmp cl,0
                jz @end
                xor ch,ch
        @wxy:
                lodsb
                stosw
                loop @wxy
                jmp @end
{----------------Функция 5: вывод данных на экран в двоичной форме----------------}
                                                { AL=0 - вывод байта            }
                                                { AL=1 - вывод слова            }
                                                { AL=2 - вывод двойного слова   }
          @f5:
                lea bx,hex_tabl
                mov ah,cl
                cmp al,0
                jz @8_2
                cmp al,1
                jz @16_2
                mov cx,8
                jmp @2
       @8_2:
                mov cx,2
                jmp @2
     @16_2:
                mov cx,4
          @2:
                push cx
                push cx
                shl cx,3
                sub cx,2
                add di,cx  { di+cx*8-2 }
                pop cx
      @lp_1:
                push cx
                xor ah,8
                mov cx,4
       @lp_2:
                mov al,dl
                and al,01h
                db 66h
                shr dx,1
                xlat
```

```
            stosw
            sub di,4
            loop @lp_2
            pop cx
            loop @lp_1
            pop cx
            shl cx,3
            add cx,2
            add di,cx
            jmp @end
{----------Функция 6: вывод данных на экран в десятичной форме----------}
                                                { AL=0 - вывод байта       }
                                                { AL=1 - вывод слова       }
                                              { AL=2 - вывод двойного слова }
      @f6:
            push bp
            push cx
            mov bp, sp
            db 66h
            mov cx, 0CA00h                    { Заносим 1 000 000 000 }
            dw  3B9Ah                                        { в ECX }
            lea bx, hex_tabl
            cmp al,1
            jne @6_1
            db 66h
            and dx, 0FFFFh
            dw 0000h
            db 66h
            mov cx, 2710h                       { Заносим 10 000 }
            dw  0000h                                        { в ECX }
      @6_1:
            cmp al,0
            jne @6_b
            db 66h
            and dx, 000FFh
            dw 0000h
            db 66h
            mov cx, 0064h                         { Заносим 100 }
            dw 0000h                                         { в ECX }
            jmp @6_b
      @osn:
            dw 0Ah
            dw 0h
      @6_b:
            db 66h
```

```
                mov ax, dx
@procsym:
                db 66h
                xor dx, dx
                db 66h
                div cx
                mov ah, [ss:bp]
                xlat
                stosw
                db 66h
                mov ax,cx
                db 66h
                mov cx,dx
                db 66h
                xor dx, dx
                db 66h
                div word [@osn]
                db 66h
                xchg ax, cx
                cmp cx, 0
                jne @procsym
                db 66h
                mov dx,bx
                pop cx
                pop bp
                jmp @end
{----------------Функция 7: Работа с прямоугольной частью экрана,----------------}
{------------------------------ограниченной координатами:----------------------------}
{----------------(X1,Y1) - левый верхний угол; (X2,Y2) - правый нижний--------------}
                                                { AL=0: сохранить в памяти       }
                                                { AL=1: восстановить на экране  }
                                                { BX - (X1,Y1)  }
                                                { DX - (X2,Y2)  }
        @f7:
                push ax
                push bx
                push dx
                mov al,bl
                mov dl,80
                mul dl
                shr bx,8
                add ax,bx
                shl ax,1
                mov di,ax               { B DI - адрес левого верхнего угла области }
                mov si,di
```

```
                xor cx,cx
                pop dx
                pop bx
                pop ax
                sub dh,bh                                   { X2-X1 }
                inc dh
                mov dh_,dh
                sub dl,bl                                   { Y2-Y1 }
                inc dl
                mov dl_,dl
                mov cl,dl
                mov cx_,cx
                mov bx,offset screen
          @l2:
                push cx
                mov cl,dh
          @l1:
                push cx
                cmp al,0
                push ax
                jz @sto
                mov ax,[bx]
                mov es:[di],ax
                jmp @m
         @sto:
                mov ax,es:[di]    { }
                mov [bx],ax
           @m:
                pop ax
                add di,2
                add bx,2
                pop cx
                loop @l1
                add si,160
                mov di,si
                pop cx
                loop @l2
      @end:
                iret
            end;
{----------------------Обработчик программного прерывания 31h----------------------}
{--------------------Вывод данных на экран в графическом режиме------------------}

procedure int_31h;assembler;
{--------------------------Функция 1: вывод на экран пиксела-------------------------}
```

```
                                        { BX - номер столбца (x)          }
                                        { CX - номер строки (y)           }
                                        { DX - 32-разрядный цвет пикселя (0RGB) }
        asm
            mov ax,640                                  Число пикселей в строке }
            db 66h
            and bx,0FFFFh
            dw 0
            db 66h
            push dx
            mul cx                                  { Произведение - в DX:AX }
            db 66h
            shl dx,16
            mov dx,ax                               { Произведение - в EDX }
            db 66h
            add dx,bx
            db 66h
            shl dx,2
            db 66h
            mov di,dx                       { В DI - адрес пикселя в видеопамяти }
            db 66h
            pop dx
            db 66h,67h,26h,89h,17h                  { MOV ES:[EDI],EDX }
            db 66h
            iret
        end;
{----------------------Обработчик программного прерывания 32h---------------------}
{---------------Вывод на экран служебной и отладочной информации-------------}

procedure int_32h;assembler;
                    { Назначение регистров: }
                    { AH - номер функции }
                    { BH/BL - номер столбца/строки экрана }
                    { Для функции 1 координаты курсора заданы по умолчанию: }
                    { BL=18, BH=0 }
        asm
{---------------Задание в переменной mode режима работы процессора-----------}
            db 66h
            push ax
            db 0fh,20h,0c0h                                 { MOV EAX,CR0 }
            test al,1                               { Анализ режима работы МП: }
            jnz @pm
            mov mode,0                                      { реальный режим }
            jmp @beg
      @pm:
```

```
                mov mode,1                          { защищенный режим }
        @beg:
                db 66h
                pop ax
                cmp ah,1
                jnz @mark
{-------------Задание для функции 1 координат маркера по умолчанию:------------}
                mov bl,18                                          { строка 18, }
                mov bh,0                                           { столбец 0 }
                push dx
                push dx
      @mark:
                pusha
                mov ax,100h                 { Установка маркера в точку экрана с }
                int 30h                     {  координатами, заданными в BH/BL }
                mov bp,sp
                mov [bp],di
                popa
                cmp ah,1
                jz @f1
                cmp ah,2
                jz @f2
                cmp ah,3
                jz @f3
                cmp ah,4
                jz @f4
                cmp ah,5
                jz @f5
                cmp ah,6
                jz @f6
                cmp ah,7
                jz @f7
                jmp @end
{-----------------------------------------Функция 1:-----------------------------------------}
{---------------Вывод на экран сообщения "Runtime error: исключение",------------}
{-----------номера исключения и (при необходимости) адреса команды,----------}
{-------------при выполнении которой (при отказе) или после выполнения--------}
{------------------------которой (при ловушке) возникло исключение-------------------}
                                                    { Назначение регистров:}
                                                    { DL - номер исключения }
                                                    { DH - 1/0: с выводом/  }
                                                    { без вывода адреса команды }
        @f1:
                mov ah,4                                           { Функция 4: }
                mov si, offset s_err
```

```
                mov cl,1bh
                int 30h                         { Вывод на экран строки s_err }
                mov ax,300h                     { Функция 3, подфункция 0: }
                mov cl,1ch
                pop dx
                int 30h                         { Вывод на экран номера исключения }
                pop dx
                mov bp,sp
{------------------------Анализ номера возникшего исключения:-----------------------}
{--------------если он равен одному из номеров 0,1,2,3,4,5,6,7,16 или 18,----------}
{--------------------то код ошибки в стек не заносился и поэтому------------------}
{-------------------------------очистки стека делать не нужно-------------------------}
                cmp dl,0
                jz @3
                cmp dl,1
                jz @3
                cmp dl,2
                jz @3
                cmp dl,3
                jz @3
                cmp dl,4
                jz @3
                cmp dl,5
                jz @3
                cmp dl,6
                jz @3
                cmp dl,7
                jz @3
                cmp dl,16
                jz @3
                cmp dl,18
                jz @3
{-------------------------------Очистка стека от кода ошибки-------------------------}
                db 66h
                mov ax,[bp+8]
                db 66h
                mov [bp+12],ax
                db 66h
                mov ax,[bp+4]
                db 66h
                mov [bp+8],ax
                db 66h
                mov ax,[bp]
                db 66h
                mov [bp+4],ax
```

```
                add sp,4
                mov bp,sp
{--------------Если при обработке возникшего исключения нужен анализ----------}
{----------------кода ошибки, то вместо вызова прерывания INT 32h--------------}
{--------------необходимо использовать собственные средства анализа,----------}
{--------------например, как это сделано в обработчике исключения 11-----------}
{-----------------------------программы P_INT (раздел 5)---------------------------------}
@3:
                cmp dh,0
                jz @end
{---------------Вывод на экран адреса (селектор:смещение) команды,--------------}
{--------------------------------которая вызвала исключение----------------------------}

                mov ah,2
                mov cl,1bh
                mov dl,28h
                int 30h                                 { Вывод на экран символа "(" }
                sub di,2
                mov ax,301h
                mov cl,1eh
                db 66h
                mov dx,[bp+16]                          { Чтение из стека CS }
                int 30h                                 { Вывод на экран значения CS }
                sub di,2
                mov ah,2
                mov cl,1eh
                mov dl,3ah
                int 30h                                 { Вывод на экран символа ":" }
                sub di,2
                mov ax,302h
                mov cl,1eh
                db 66h
                mov dx,[bp+12]                          { Чтение из стека EIP }
                int 30h                                 { Вывод на экран значения EIP }
                sub di,2
                mov ah,2
                mov cl,1bh
                mov dl,29h
                int 30h                                 { Вывод на экран символа ")" }
                jmp @end
{-------------------------------------Функция 2:-------------------------------------------}
------------------------------Вывод на экран значений РОН:-----------------------------}
{--------------------EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI----------------------}

{------------Особенность этой функции: перед ее вызовом необходимо----------}
```

```
{------------------сохранить в стеке значения регистров AX и BX,-----------------}
{--------------------а после вызова функции - их восстановить---------------------}
{--------------------(см. обработчик ислючения 0 модуля PROT)--------------------}
                                        { В AL  - режим вывода значений регистров }
                                        { AL=0: вывод в одну строку }
                                        { AL=1: вывод в две строки }

@f2:
{----------------Сохранить параметры вызова прерывания INT 32h:----------------}
            push ax                                          { режим вывода и }
            push bx                                      { координаты вывода }
            mov bp,sp
            cmp mode,0
{----------------Восстановить исходные значения регистров AX и BX-------------}
{------------------для реального и защищенного режимов работы МП-------------}
            jnz @pm1
            mov ax,[bp+12]
            mov bx,[bp+10]
            jmp @@1
    @pm1:
            mov ax,[bp+18]
            mov bx,[bp+16]
@@1:
            db 66h
            pusha                        { Сохранить РОНы для вывода на экран }
            mov bp,sp
            mov bx,[bp+32] { Восстановить координаты места экрана }
            mov ax,100h
            int 30h                                      { Установить маркер }
{----------------Вывод массива строк наименований регистров (s_reg)------------}
            lea bx,s_reg
            mov cx,8
      @as:
            push cx
            push bx
            mov ah,4
            mov cl,1ah
            mov si,bx
            int 30h
            add di,2
            pop bx
            add bx,256
            pop cx
            cmp cx,5
            jnz @l
```

```
        mov ax,[bp+34]                  { Анализ режима вывода: }
        cmp al,0                          { AL=0: в одну строку }
        jz @l
        add di,408                        { AL=0: в две строки }
@l:
        loop @as
        mov ax,[bp+34]
        cmp al,0                        { Анализ режима вывода }
        jnz @l3
        add di,142
        jmp @l4
@l3:
        add di,142
@l4:
        mov cx,8
@l1:
        db 66h
        pop dx          { Чтение из стека значения очередного РОНа }
        push cx
        push di
        mov cl,1bh
        mov ax,302h
        int 30h          { Вывод на экран значения очередного РОНа }
        pop di
        sub di,18
        pop cx
        cmp cx,5
        jnz @l2
        mov bp,sp
        mov ax,[bp+18]
        cmp al,0                        { Анализ режима вывода }
        jz @l2
        sub di,408
@l2:
        loop @l1
        pop bx
        pop ax
        jmp @end

{-----------------------------------------Функция 3:-----------------------------------------}
{----------------------Вывод на экран значений счетчика команд EIP----------------}
{----------------------и сегментных регистров CS, SS, ES, DS, FS и GS---------------}
        { Назначение регистров: AL - номер подфункции }
        {------AL=0: вывод значений собственно регистров EIP и CS------}
        {------AL=1: вывод "теневых" значений регистров EIP и CS--------}
```

```
                {----В случае AL=1 перед вызовом функции необходимо задать:---}
                { в CX  - значение CS  }
        @f3:
                cmp al,0
                jz @01
                push cx
                cmp mode,0
                jz @31
                db 66h
        @31:
                push dx
        @01:
                push ax
{--------------------------------Вывод массива строк s_sreg-----------------------------}
                lea bx,s_sreg
                mov cx,7
      @as1:
                push cx
                push bx
                mov ah,4
                mov cl,1ah
                mov si,bx
                int 30h
                add di,2
                pop bx
                add bx,256
                pop cx
                loop @as1
                add di,82
                pop ax
                cmp al,0
                jnz @02
                mov bp,sp
                cmp mode,0
                jz @32
                db 66h
      @32:
                mov dx,[bp]
                mov cl,1bh
                mov ax,302h
                int 30h                          { Вывод на экран значения EIP }
                cmp mode,0
                jnz @pm2
                mov dx,[bp+2]
                jmp @33
```

```
@pm2:
        mov dx,[bp+4]
@33:
        mov cl,1bh
        mov ax,301h
        int 30h                          { Вывод на экран значения CS }
        jmp @ss
@02:
        cmp mode,0
        jz @34
        db 66h
@34:
        pop dx
        mov cl,1bh
        mov ax,302h
        int 30h                  { Вывод на экран "теневого" значения EIP }
        pop dx
        mov cl,1bh
        mov ax,301h
        int 30h                   { Вывод на экран "теневого" значения CS }
 @ss:
        mov dx,ss
        mov cl,1bh
        mov ax,301h
        int 30h                          { Вывод на экран значения SS }
        mov dx,es
        mov cl,1bh
        mov ax,301h
        int 30h                          { Вывод на экран значения ES }
        mov bp,sp
        mov dx,ds
        mov cl,1bh
        mov ax,301h
        int 30h                          { Вывод на экран значения DS }
        db 8ch,0e2h                                        { MOV DX,FS }
        mov cl,1bh
        mov ax,301h
        int 30h                          { Вывод на экран значения FS }
        db 8ch,0eah                                        { MOV DX,GS }
        mov cl,1bh
        mov ax,301h
        int 30h                          { Вывод на экран значения GS }
        jmp @end
```

{----------Функция 4: Вывод на экран наименований и значений----------}

```
{----------------------битов (флагов) регистра управления CR0-----------------------}
                    { Назначение регистров: AL - номер подфункции }
                    {-------AL=0: вывод в двоичном виде значений битов CR0--------}
                    {-------AL=1: вывод  наименований и значений битов CR0--------}
          @f4:
                    cmp al,0
                    jz @val
{---------------------------------Вывод массива строк s_CR0-----------------------------}
                    lea bx,s_CR0
                    mov cx,2
          @as2:
                    push cx
                    push bx
                    mov ah,4
                    mov cl,1dh
                    mov si,bx
                    int 30h
                    add di,96
                    pop bx
                    add bx,256
                    pop cx
                    loop @as2
                    sub di,8
          @val:
                    mov ah,4
                    mov cl,1ah
                    mov si,offset s_CR0n                    { Вывод строки  "CR0:" }
                    int 30h
                    db 0fh,20h,0c2h                         { MOV EDX,CR0 }
                    mov cl,1bh
                    mov ax,502h
                    int 30h                  { Вывод на экран значений разрядов CR0 }
                    jmp @end

{-----------------Функция 5: Вывод на экран наименований и значений---------------}
{----------------------битов (флагов) регистра управления CR4-----------------------}
@f5:
{---------------------------------Вывод массива строк s_CR4-----------------------------}
                    lea bx,s_CR4
                    mov cx,3
        @as3:
                    push cx
                    push bx
                    mov ah,4
                    mov cl,1dh
```

```
                mov si,bx
                int 30h
                add di,96
                pop bx
                add bx,256
                pop cx
                loop @as3
                sub di,8
                mov ah,4
                mov cl,1ah
                mov si,offset s_CR4n                    { Вывод строки  "CR4:" }
                int 30h
                db 0fh,20h,0e2h                         { MOV EDX,CR4 }
                mov cl,1bh
                mov ax,502h
                int 30h                 { Вывод на экран значений разрядов CR4 }
                jmp @end

{----------------Функция 6: Вывод на экран наименований и значений---------------}
{--------------------------битов (флагов) регистра EFLAGS----------------------------}
@f6:
{-----------------------------Вывод массива строк s_EFLAGS---------------------------}
                lea bx,s_EFLAGS
                mov cx,3
        @as4:
                push cx
                push bx
                mov ah,4
                mov cl,1dh
                mov si,bx
                int 30h
                add di,96
                pop bx
                add bx,256
                pop cx
                loop @as4
                sub di,14
                mov ah,4
                mov cl,1ah
                mov si,offset s_EFLAGSn
                int 30h                                 { Вывод строки  "EFLAGS:" }
                db 66h
                pushf
                db 66h
                pop dx
```

```
            mov cl,1bh
            mov ax,502h
            int 30h                 { Вывод на экран значений разрядов EFLAGS }
            jmp @end
{-------------------Функция 7: Вывод на экран содержимого стека-------------------}
                                    { Назначение регистров: AL - номер подфункции }
                                    { AL=0: 16-розрядный стек }
                                    { AL=1: 32-розрядный стек }
                                    { CX -  число элементов стека }
      @f7:
            mov bp,sp
            mov si,0
            cmp al,0
            jnz @732
      @716:
            push cx
            push si
            mov dx,[bp+si]
            mov ax,301h
            mov cl,1bh
            int 30h
            add di,150
            pop si
            add si,2
            pop cx
            loop @716
            jmp @end
      @732:
             push cx
            push si
            db 66h
            mov dx,[bp+si+12]
            mov ax,302h
            mov cl,1bh
            int 30h
            add di,142
            pop si
            add si,4
            pop cx
            loop @732
            jmp @end
     @end:
            cmp mode,0
            jz @iret
            db 66h
```

```
        @iret:
                iret

          end;{int_32nh}
{===============Секция инициализации модуля PROT===============}
    begin
        asm  mov ax,ax end;
{--------------Формирование дескриптора сегмента кода модуля PROT-----------}
{-------------------------------------( таблица GDT)-------------------------------------}

        init_gdt(PROT_sel,$ffff,longint(CSeg) shl 4,acc_code,0);

{=================Формирование таблицы IDT=================}

{----------------Дескрипторы шлюзов обработчиков исключений 0-18:-------------}
                init_idt(0,ofs(exc_00),PROT_sel,acc_trap);
                init_idt(1,ofs(exc_01),PROT_sel,acc_trap);
                init_idt(2,ofs(exc_02),PROT_sel,acc_trap);
                init_idt(3,ofs(exc_03),PROT_sel,acc_trap);
                init_idt(4,ofs(exc_04),PROT_sel,acc_trap);
                init_idt(5,ofs(exc_05),PROT_sel,acc_trap);
                init_idt(6,ofs(exc_06),PROT_sel,acc_trap);
                init_idt(7,ofs(exc_07),PROT_sel,acc_trap);
                init_idt(8,ofs(exc_08),PROT_sel,acc_trap);
                init_idt(10,ofs(exc_10),PROT_sel,acc_trap);
                init_idt(11,ofs(exc_11),PROT_sel,acc_trap);
                init_idt(12,ofs(exc_12),PROT_sel,acc_trap);
                init_idt(13,ofs(exc_13),PROT_sel,acc_trap);
                init_idt(14,ofs(exc_14),PROT_sel,acc_trap);
                init_idt(16,ofs(exc_16),PROT_sel,acc_trap);
                init_idt(17,ofs(exc_17),PROT_sel,acc_trap);
                init_idt(18,ofs(exc_18),PROT_sel,acc_trap);
{-----------------------Дескрипторы шлюзов обработчиков-----------------------}
{-----------------------внешних аппаратных прерывания:-------------------------}

{-----------Дескриптор шлюза обработчика прерывания от таймера:---------}
                init_idt($20,ofs(PIC_1),PROT_sel,acc_int);
{---------Дескриптор шлюза обработчика прерывания от клавиатуры:-------}
                init_idt($21,ofs(keyb),PROT_sel,acc_int);
{------------Дескрипторы шлюзов обработчиков прерываний IRQ2-IRQ7:----------}
for i:=2 to 7 do
init_idt($20+i,ofs(PIC_1),PROT_sel,acc_int);
{----------Дескрипторы шлюзов обработчиков прерываний IRQ8-IRQ15:---------}
                for i:=8 to 15 do
```

```
            init_idt($20+i,ofs(PIC_2),PROT_sel,acc_int);
{----------Дескрипторы шлюза обработчика программного прерывания:--------}

            init_idt($30,ofs(int_30h),PROT_sel,acc_trap_16);
            init_idt($31,ofs(int_31h),PROT_sel,acc_trap);
            init_idt($32,ofs(int_32h),PROT_sel,acc_trap);
            for i:=$33 to 255 do init_idt(i,0,0,0);
    end.
```

1.17 Індівідуальні завдання

При виконанні кожного із нижчеперелічених індивідуальних завдань необхідно:

а) перевести мікропроцесор у захищений режим;

б) здійснити дії, передбачені індивідуальним завданням;

в) повернутися у реальний режим.

1. Вивести на екран значення всіх сегментних регістрів.

2. Зберегти в області пам’яті вище 1 МБ поточний зміст відеопам’яті в текстовому режимі з наступним відновленням зображення.

3. Створити в пам’яті вище 1МБ сегмент коду розміром 512 байт, захищений від читання, і перевірити роботу з ним.

4. Зберегти в області пам’яті вище 1Мб поточний зміст відеопам’яті в графічному режимі з наступним відновленням зображення.

5. Створити в пам’яті вище 4 МБ сегмент даних розміром 4 МБ, захищений від запису, і перевірити роботу з ним.

6. Розробити функцію для виводу на екран значень змінних типів Byte та Word.

7. Встановити розрядність даних за замовчуванням, рівну 32, і перевірити роботу з ними.

8. Забезпечити можливість модифікації програм у захищеному режимі.

9. Встановити розрядність адреси по умовчанню, рівну 32, і перевірити роботу з нею.

10. Розробити процедуру читання сектора із жорсткого магнітного диска.

11. Використовуючи процедуру читання сектора, зберегти в пам’яті вище

1 Мбайта зміст FAT розділу жорсткого диска.

12. Використовуючи процедуру читання сектора, зберегти в пам’яті вище 1 Мбайта зміст кореневого каталога розділу жорсткого диска.

13. Виконати тестування пам’яті вище 1 Мбайта.

14. Організувати в області пам’яті вище 1 Мбайта зберігання набору шрифтів із можливістю їхнього завантаження у відеопам’ять.

15. Розробити текстовий редактор, що працює з даними в області пам’яті вище 1 Мбайта.

16. Визначити розмір вбудованої кеш-пам’яті МП і кеш-пам’яті другого рівня персонального комп’ютера.

2. РОБОТА З ПЕРЕРИВАНННЯМИ У ЗАХИЩЕНОМУ РЕЖИМІ

2.1 Види переривань та виключень

Переривання – це припинення виконання програми мікропроцесором за наявністю запиту та дозволу на переривання, передача управління спеціальним процедурам – оброблювачам переривання з подальшим поверненням до виконання перерваної програми.

По джерелу й характеру виникнення запитів переривання можна поділити на апаратні та програмні.

Програмні переривання викликаються за командою INT n, де n – номер переривання. При виконанні цих команд мікропроцесор здійснює ті ж самі дії, що і при обробленні апаратних переривань. Використання програмних переривань дозволяє підвищити гнучкість програмного забезпечення.

Апаратні переривання є асинхронними по відношенню до роботи МП і поділяються на внутрішні та зовнішні. Під перериванням в вузькому сенсу слова розуміється реакція на запит від зовнішнього пристрою (ЗП) системи.

Зовнішні переривання, в свою чергу, поділяються на ті, що маскуються й на ті, що не маскуються.

Переривання, що маскуються, названі так, оскільки можуть бути заборонені (замасковані) скиданням прапора IF в регістрі прапорів EFLAGS. При застосуванні внутрішньої шини APIC (Advanced Programmable Interrupt Controller), а це трапляється при роботі МП в складі багатопроцесорної системи, вони надходять на вхід LINT0 мікропроцесора від зовнішніх пристроїв, таких як таймер, клавіатура, послідовний порт, годинник реального часу, магнітні диски та ін., для обміну даними з МП. При однопроцесорній роботі (без застосування шини APIC) вхід МП LINT0 конфігурується у вхід INTR.

При роботі із ЗП необхідні спеціальні апаратні засоби для того, щоб визначити пристрій, який викликав переривання, встановити пріоритет на обслуговування поміж ЗП, замаскувати при необхідності деякі пристрої, встановити номер переривання. Усі ці функції в мікропроцесорних системах виконує пристрій PIC (Programmable Interrupt Controller) – програмувальний контролер переривань (ПКП).

Мікропроцесор, відповідаючи на запити переривань, що маскуються, виконує два блокованих цикли підтвердження переривання і наприкінці другого

циклу читає 8-розрядний номер переривання від зовнішнього пристрою, що генерується ПКП.

Переривання, що не маскуються, надходять на вхід мікропроцесора NMI (NonMaskable Interrupt) (при застосуванні шини APIC – на вхід LINT1) внаслідок помилок в роботі зовнішнього обладнання, наприклад, від схем контролю пам'яті по паритету. Вони можуть бути заборонені тільки зовнішніми по відношенню до МП схемами комп'ютера.

Внутрішні апаратні переривання надходять від вузлів МП і свідчать про виникнення такої ситуації при виконанні команди мікропроцесором, яку він не в змозі самостійно вирішити, наприклад, при звертанні до сегмента, якого немає в пам'яті, надходженні неприпустимого коду команди та ін. Такі запити на переривання називаються виключеннями (exceptions).

Виключення поділяються на відмови (faults), пастки (traps) та виходи з процесу (aborts).

Відмови – це виключення, що виникають під час виконання команди, наприклад, при діленні на 0. Для того, щоб забезпечити прозорість механізму оброблення переривань, при відмовах здійснюється збереження стану процесора (вміст необхідних регістрів, вказівника стеку), який він мав до виникнення виключення.

Пастки – це виключення, що виникають після виконання команди, наприклад, після виконання команди налагоджування INT 3.

Виходи з процесу – це виключення, коли адреса команди, що викликала переривання, надійно не може буди визначена. У цьому випадку метою оброблювача переривання є виведення налагоджувальної інформації та завершення роботи програми. Прикладом виходу з процесу є виключення, що викликане подвійною помилкою.

Робота процесора також може бути перервана появою активних сигналів на його контактах RESET#, FLUSH#, STPCLC#, SMI# та INIT#. Але при цьому не застосовується механізм оброблення переривань та виключень, який розглядається в даному розділі.

Мікропроцесори сімейства x86 здатні обробити до 256 різних переривань. Перші 32 номери переривань зарезервовані фірмою Intel для обробки виключень та зовнішнього переривання NMI. Інші надані для нумерації маскованих апаратних переривань та програмних переривань.

При роботі процесора можуть виникнути такі зарезервовані перериван-

ня (виключення).

Переривання 0 *(тип:**відмова**)* ***#DE*** **(**Divide Error**)**: виключення ділення на 0 виникає при виконанні команд DIV, IDIV, коли дільник дорівнює нулю, або коли число бітів результату операції перевищує число бітів операнда призначення.

Переривання 1 #DB (Debug Error): виключення, яке застосовується в процесі налагоджування програм і виникає:

– в покроковому режиму роботи мікропроцесора (при встановленому прапорі TF = 1 регістра EFLAGS) після виконання кожної команди (*тип:**пастка***);

– у разі звертання до сегмента TSS, що має біт T = 1 при переключенні задач (*тип:**пастка***);

– при зупинці по контрольних точках команд, що встановлюються за допомоги регістрів налагоджування DR0-DR3, DR6-DR7 (*тип:**відмова***);

– при зупинці по контрольних точках даних, що встановлюються за допомоги регістрів налагоджування DR0-DR3, DR6-DR7 (*тип:**пастка***);

– при зупинці по контрольних точках при звертанні до пристроїв введення/виведення, що встановлюються за допомоги регістрів налагоджування DR0-DR3, DR6-DR7 (*тип:**пастка***);

– при порушенні захисту в разі звертання до регістрів налагоджування DR0-DR3, DR6-DR7 (*тип:***відмова**).

Переривання 2 NMI (NonMaskable Interrupt): зовнішнє апаратне немасковане переривання виникає при надходженні активного сигналу NMI# на вхід NMI мікропроцесора.

Переривання 3 *(тип:**пастка**)* ***#BP*** (Breakpoint): виключення контрольних точок виникає при виконанні однобайтової команди INT 3 (0CCh), яка використовується для встановлення контрольних точок при налагоджуванні програми.

Операційна система чи налагоджувач повинні замінити перший байт команди, перед виконанням якої треба зупинити проходження програми, наприклад, з метою виведення на екран поточної налагоджувальної інформації (вмісту регістрів МП, областей пам’яті та ін.), командою INT 3, яка при виконанні визиває виключення 3. При цьому застосовується сегмент даних, що накладається на сегмент коду для внесення змін до програми, оскільки запис до сегменту коду у захищеному режимі заборонений. Обробник виключення 3 повинен відновити перший байт команди і декрементувати значення лічильника

команд у стеку.

Переривання 4 *(тип:**пастка)*** ***#OF*** (Overflow): виключення знакового переповнення виникає, коли МП виконує команду INTO при встановленому прапорі переповнення OF = 1 в регістрі EFLAGS. Деякі арифметичні команди, наприклад, ADD або SUB, виконують операції як із знаковими так і з беззнаковими операндами, встановлюючи прапори знакового чи беззнакового переповнення – відповідно OF і CF в регістрі EFLAGS.

Оскільки для беззнакових операндів знакове переповнення OF не є помилкою, а для знакових – це помилка, то після виконання операції зі знаковими операндами застосовується команда INTO для виявлення помилок при виконання операцій саме із знаковими операндами.

При розробці програми на Асемблері при необхідності команду INTO після арифметичної команди додає програміст. Коли програма розробляється на мові високого рівня (Pascal, C та ін.), цю команду додає чи не додає транслятор з цієї мови в залежності від типу даних. Наприклад, при програмуванні на мові Pascal після арифметичних операцій з даними типів Shortint, Integer та Longint команда INTO додається, а для типів Byte чи Word – ні.

Переривання 5 *(тип:**відмова)*** ***#BR*** (BOUND Range Exceeded): виключення перевищення діапазону має місце при виконанні команди BOUND, якщо вміст регістра-операнда, який задає значення, що перевіряється, виходить за означені межі діапазону, які задані другим операндом команди – адресою першої з двох суміжних комірок пам’яті, що задають відповідно нижню і верхню границю діапазону.

Переривання 6 *(тип: **відмова)*** ***#UD*** **(**Invalid Opcode (Undefined Opcode)**:** виключення, що виникає при роботі мікропроцесора, коли він зустрічається з такими ситуаціями:

– виконання недозволеного або зарезервованого коду команди, або MMX-команди, коли МП не підтримує MMX-технологію;

– виконання MMX-команди, при встановленому біті EM в регістрі CR0;

– виконання команди, тип операнда якої не відповідає коду операції, наприклад, команда міжсегментного переходу, яка застосовує регістровий операнд, або команда LES, вихідний операнд якої не є адресою пам’яті;

– виконання команди UD2 (машинний код: **db 0fh,0bh**);

– виявлення префіксу LOCK перед командою, яка не може здійснити захват шини, або якщо операнд результату не є комірка пам’яті. Відзначимо, що

тільки деякі команди, наприклад, MOVS, INS, та OUTS можуть захватити системну шину для обміну даних;

– спроба виконання команд LLDT, SLDT, LTR, STR, LSL, LAR, VERR, VERW або ARPL в реальному режимі роботи;

– спроба виконати команду RSM в режимі роботи, який не є режимом SMM.

Переривання 7 *(тип:* ***відмова) #NM*** **(**Device Not Available (No Math Coprocessor)): виключення недоступності пристрою FPU виникає в трьох випадках:

– процесор зустрічає команду з плаваючою крапкою, коли встановлений біт EM в регістрі CR0;

– процесор зустрічає команду з плаваючою крапкою або MMX-команду, коли встановлений біт TS в регістрі CR0;

– процесор зустрічає команди WAIT або FWAIT, коли встановлені біти MP і TS в регістрі CR0.

Біт EM встановлюється, коли процесор не має вбудованого пристрою з плаваючою крапкою FPU. У цьому випадку виключення 7 «Недоступний пристрій» генерується кожний раз, коли зустрічається команда з плаваючою крапкою. Це дає змогу програмно здійснювати емуляцію команд FPU.

Біт TS вказує, що відбулося переключення задачі і контекст FPU (зміст регістрів FPU) після останнього виконання команди з плаваючою крапкою або MMX-команди є незбереженим. Тому обробник виключення 7 може зберегти контекст FPU в пам’яті до виконання нових команд з плаваючою крапкою або MMX-команд.

Біт MP застосовується тільки разом з бітом TS для визначення, чи потрібно генерувати виключення 7 при виконанні команд WAIT або FWAIT. Він поширює функції біту TS на команди WAIT або FWAIT і дає змогу обробнику виключення 7 заздалегідь забезпечити зберігання контексту FPU. Біт MP був впроваджений в мікропроцесорах 80286 та 80386DX. Для наступних МП (за виключенням i486SX) цей біт повинен бути завжди встановлений.

Переривання 8 *(тип:* ***вихід з режиму) #DF*** (Double Fault): виключення подвійної помилки показує, що процесор зустрічає друге виключення під час обробки першого. Взагалі може бути два вирішення цієї проблеми: по-перше, виключення можуть бути оброблені послідовно одно за одним, по-друге, може бути сгенероване виключення 8 «Подвійна помилка». Для визначення, як треба

поступати в таких ситуаціях, всі виключення поділяються на такі три класи:

1) клас 1 містить такі виключення – 1, 2, 3, 4, 5, 6, 7, 16, 17, 18 та програмні переривання і апаратні переривання, що маскуються;

2) клас 2 містить такі виключання – 0, 10, 11, 12, 13;

3) клас 3 містить виключення 14.

Виключення подвійної помилки 8 буде генеруватися тільки в трьох випадках, які наведені в таблиці 2.1.

Таблиця 2.1 – Генерація виключення 8

№ п/п	Клас першого виключення	Клас другого виключення
1	Клас 2	Клас 2
2	Клас 3	Клас 2
3	Клас 3	Клас 3

Якщо при обробці виключення 8 виникне ще одно (третє) виключення, то МП переходить до стану виключення – Shutdown, вихід з якого здійснюється за сигналом NMI або через скидання мікропроцесора.

***Переривання 9*:** Зарезервовано.

Було застосовано в МП 80386 та 80387 для індикації порушень доступу до сегментів та сторінок пам’яті при передачі даних від FPU. У наступних МП це виключення зарезервоване, але при виникненні указаних порушень замість виключення 9 генерується виключення 13

Переривання 10 *(тип:* ***відмова) #TS*** (Invalid TSS): виключення невірного сегмента стану задачі TSS (Task State Segment) вказує, що при спробі переключити задачу, була виявлені такі невірні дані в TSS задачі, на яку зроблено переключення:

– розмір TSS менше, ніж 67h байтів;

– невірна LDT, або LDT немає в пам’яті;

– селектор сегмента стеку перевищує розмір дескрипторної таблиці;

– для сегмента стеку заборонений запис;

– для сегмента стеку не виконується DPL = CPL;

– для селектора сегмента стеку не виконується RPL = CPL;

– селектор сегмента коду перевищує розмір дескрипторної таблиці;

– сегмент коду не є сегментом, що виконується;

– для сегмента коду, що не підпорядкований, не виконується: DPL = CPL;

– для сегмента коду, що підпорядкований, не виконується: DPL більше ніж CPL;

– селектор сегмента даних перевищує розмір дескрипторної таблиці;

– для сегмента даних заборонено читання.

Переривання 11 *(тип:* ***відмова) #NP*** (Segment Not Present): виключення відсутності сегмента в пам'яті, вказує, що в дескрипторі сегмента чи шлюзу скинутий біт присутності (P = 0). Процесор може генерувати це виключення під час таких операцій:

– при спробі завантаження регістрів CS, DS, ES, FS та GS ((для сегмента стека це викликає виключення 12));

– при спробі завантаження регістра LDTR застосовуючи команду LLDT;

– коли виконується команда LTR і TSS має ознаку неприсутності;

– при спробі застосувати дескриптор шлюзу або TSS, що мають ознаку неприсутності, але в іншому є вірними.

Переривання 12 *(тип:* ***відмова) #SS*** (Stack Segment Fault): виключення помилки звернення до стека, вказує, що була виявлена одна з пов'язаних з стеком умов:

– виявлено порушення границі сегменту під час команди, що зверталася до регістру SS. Це можуть бути команди POP, PUSH, CALL, RET, IRET, ENTER та LEAVE, так же як і інші звернення до пам'яті, що застосовують явно чи неявно регістр SS, наприклад, MOV AX [BP+6].

– при спробі завантаження регістра SS селектором неприсутнього в пам'яті (P = 0) сегмента стеку.

Переривання 13 *(тип:* ***відмова) #GP*** (General Protection): порушення загального захисту, вказує, що виявлено одно з порушень класу «Загальні порушення захисту», яке не відноситься до інших виключень:

– перевищення границі сегменту при доступі до сегментних регістрів CS, DS, ES, FS та GS;

– перевищення границі сегменту при посиланні до дескрипторної таблиці;

– передача виконання до сегменту, що не є сегментом, який може виконуватися;

– запис до сегмента коду, або до сегмента даних, для якого дозволено тільки читання;

– читання з сегмента коду, для якого дозволено тільки виконання;

– завантаження регістра SS селектором сегмента, для якого дозволено тільки читання;

– завантаження регістрів SS, DS, ES, FS, GS та CS селектором системного сегмента;

– завантаження регістрів DS, ES, FS та GS селектором сегмента, для якого дозволено тільки виконання;

– завантаження регістра SS селектором, який може виконуватися, або нуль-селектором;

– завантаження регістра CS селектором для сегмента даних, або нуль-селектором;

– адресація до пам’яті з застосуванням сегментних регістрів DS, ES, FS та GS, коли в них завантажений нуль-селектор;

– переключення за допомоги команд JMP та CALL до задачі, яка є занятою;

– переключення до доступної (незанятої) задачі при виконання команди IRET;

– застосування селектора сегмента при переключенні задачі, який вказує на дескриптор TSS в поточній таблиці LDT, бо дескриптор TSS повинен знаходитися тільки в таблиці GDT;

– порушення будь-якого правила, пов’язаного з привілеями (див. розділ «Захист пам’яті»);

– перевищення максимальної довжини команди, яка дорівнює 15 байтів;

– завантаження регістра CR0 значенням з встановленим бітом PG (дозволена сторінкова організація пам’яті) і скинутим бітом PE (реальний режим роботи МП);

– завантаження регістра CR0 значенням з встановленим бітом NW і скинутим бітом CD;

– посилання до дескриптора в IDT після виникнення переривання або виключення, який не є дескриптором ні пастки, ні переривання, ні шлюзу задачі;

– спроба доступу до обробника переривання чи виключення через шлюзи переривання або пастки з режиму віртуального 8086, якщо DPL сегмента коду більше ніж 0;

– спроба запису 1 до зарезервованих бітів регістру CR4;

– спроба виконати привілейовану команду, якщо CPL не дорівнює 0;

– запис в зарезервовані біти регістрів MSR;

– доступ до шлюзу, що містить нуль-дескриптор;

– виконання команди INT n, якщо CPL більше ніж DPL шлюзу переривання, пастки або задачі, на який йде посилання;

– селектор сегмента в шлюзі переривання, пастки або задачі не є селектором сегмента коду;

– операнд команди LLDT, що задає селектор сегмента, має локальний тип (TI = 1), або не відповідає типу дескриптора таблиці LDT;

– операнд команди LTR, що задає селектор сегмента, має локальний тип (TI = 1), або вказує на TSS, що не є доступним;

– селектор сегмента коду в командах CALL, JMP та RET є нуль-дескриптором;

– якщо біт PAE в регістрі управління CR4 встановлений, та процесор виявляє будь-які зарезервовані біти, що встановлені в одиницю в таблиці покажчиків на каталоги сторінок.

Переривання 14 *(тип:* ***відмова) #PF*** (Page Fault): виключення помилки сторінки вказує, що при встановленій сторінковій організації пам'яті (біт PG = 1 в регістрі CR0) процесор виявляє одну з таких ситуацій під часу виконання механізму перетворення лінійної адреси в фізичну адресу:

– біт присутності (P) в покажчику каталогу сторінок чи таблиці сторінок є скинутим, вказуючи на то, що таблиці сторінок чи сторінки немає в фізичної пам'яті;

– процедура не має достатнього рівня привілею для доступу до сторінки;

– програма рівня користувача спробує зробити запис до сторінки, яка має ознаку «тільки для читання». Якщо встановлений біт WP в регістрі CR0, то виключення 14 виникає також тоді, коли програма з будь-якім рівнем привілеїв спробує зробити запис до сторінки з ознакою «тільки для читання».

Переривання 15**:** Зарезервовано.

Переривання 16 *(тип:* ***відмова) #MF*** (Floating Point Error(Math Fault)): виключення помилки пристрою FPU, яке генерується при умові, що біт NE регістра CR0 встановлений. Під час виконання команд з плаваючою крапкою пристрій FPU виявляє шість типів помилок:

1) невірна операція (**#I**):

– переповнення або антипереповнення стеку (#**IS**);

– невірна арифметична операція (**#IA**);

2) ділення на нуль (**#Z**);

3) ненормалізований операнд (**#D**);

4) числове переповнення (**#O**);

5) числове антипереповнення (**#U**);

6) неточний результат (**#P**).

Для кожного з цих типів помилок пристрій FPU має прапор в регістрі стану FPU і біт маски в регістрі управління FPU.

Якщо FPU виявляє помилку, виконуючи команди з плаваючою крапкою, і біт маски для цієї помилки встановлений, тобто помилка замаскована, то FPU обробляє помилку, генеруючи спеціальне значення за умовчанням та продовжуючи виконання програми.

Якщо біт маски для цього типа помилки скинутий, і біт NE регістра CR0 встановлений, то FPU робить наступне:

1) встановлює відповідний прапор в регістрі стану FPU;

2) очікує доти в потоці команд не зустрінеться наступна команда з плаваючою крапкою, яка є командою “з очікуванням” (усі команди з плаваючою крапкою за виключенням команд FNINIT, FCNLEX, FNSTSW, FNSTSW AX, FNSTCW, FNSTENV, FNSAVE є саме командами “з очікуванням”);

3) виробляє внутрішній сигнал помилки, по якому МП, в свою чергу, генерує виключення помилки FPU.

Переривання 17 *(тип:**відмова)*** ***#AC*** (Alignment Check): виключення помилки вирівняння вказує, що процесор виявив невирівняний операнд в пам’яті, коли дозволена перевірка вирівняння.

Перевірка вирівняння здійснюється тільки в сегментах даних або стеку, проте не в сегментах коду або системних сегментах. Прикладом порушення вирівняння є слово, яке записано до пам’яті по непарній адреси.

У таблиці 2.2 наведені загальні вимоги до вирівняння даних, що розпізнаються процесором.

Таблиця 2.2 – Вимоги до вирівняння даних

Тип даних	Адреса повинна ділитися на
Слово (Word)	2
Подвійне слово (Double Word)	2
Дійсні одинарної точності (Single Real)	4
Дійсні подвійної точності (Double Real)	4
Дійсні розширені (Extended Real)	8
Селектор сегмента	8
32-розрядний далекий вказівник	8
48-розрядний далекий вказівник	2
32-розрядний вказівник	4
Вміст регістрів GDTR, IDTR, LDTR та TR	4
FSTENV/FLENV область схову (Save Area)	4
FSAVE/FRSTOR область схову	4 або 2 в залежності від розміру операнда
Рядок бітів	4 або 2 в залежності від розміру операнда

Невирівняні дані збільшують число циклів і, таким чином, час обміну даними з пам’яттю.

Для здійснення перевірки вирівняння даних повинні бути виконані такі умови:

– встановлений біт AM в регістрі CR0;

– встановлений біт AC в регістрі EFLAGS;

– рівень привілеїв програми CPL дорівнює 3.

Зазначимо, що посилання до пам’яті, які за умовчанням належать до рівня привілею 0, такі, як завантаження дескрипторних таблиць, не генерують помилки перевірки вирівняння навіть тоді, коли зроблені з найнижчим рівнем привілею 3.

Зберігання вмісту регістрів GDTR, IDTR, LDTR та TR в пам’яті програмою рівня привілею 3 може генерувати помилку вирівняння. Хоча прикладні програми дуже рідко здійснюють зберігання вмісту цих регістрів, помилки вирівняння можна уникнути шляхом вирівняння даних навіть до адреси слова.

Переривання 18 *(тип:* ***вихід з режиму)*** **#MC** (Machine Check): виключення апаратного контролю вказує, що процесор за допомоги своїх схем контролю виявив внутрішню помилку або помилку шини, або помилку шини виявив зовнішній агент (інший процесор в багатопроцесорній системі). Виключення апаратного контролю є специфічним для моделі процесора і відрізняється для МП Pentiom і МП сімейства P6.

Помилка шини, яка виявлена зовнішнім агентом, передається процесору через контакти BINIT# для P6 і BUSCHK# для Pentium, після чого інформація про помилку завантажується в регістри апаратного контролю, і здійснюється генерація виключення апаратного контролю.

Для процесорів сімейства P6: якщо біт EIPV регістра MCG_STATUS, який приналежить до регістрів MSR (Model Specific Register), встановлений, то збережене в стеку значення регістрів CS та EIP буде безпосередньо пов'язане з помилкою, яка викликала виключення апаратного контролю (виключення типу відмова).

Якщо цей біт є скинутим, то адреса команди в стеку не буде обов'язково адресою команди, що викликала це виключення (виключення типу вихід з режиму).

Переривання 19-31**:** Зарезервовано.

Треба зауважити, що номери зарезервованих виключень можуть бути використані в майбутньому. Тому ці номери не можна застосовувати при розробленні програмного забезпечення.

2.2 Пріоритети виключень та переривань

Якщо більш ніж одно виключення або переривання виникає к моменту завершення команди, процесор обслуговує їх в визначеному порядку. У таблиці 2.3 наведені пріоритети серед класів виключень та переривань.

У той час як пріоритети між класами є постійними в межах даної архітектури, то пріоритети між виключеннями одного класу є залежними від впровадження і можуть змінюватися при переході від одної моделі процесора до іншої.

Процесор обслуговує в першу чергу виключення та переривання класу, що має найвищий пріоритет, передаючи виконання першій команді оброблювача виключення або переривання.

Таблиця 2.3 – Пріоритети серед виключень та переривань

Пріоритет	Опис
1	2
1 (найвищий)	Апаратне скидання та апаратний контроль: – RESET (скидання процесора); – виключення апаратного контролю.
2	Виключення типу пастки при переключенні задач: встановлений біт T в TSS.
3	Зовнішні апаратні переривання: – FLUSH; – STOPCLK; – SMI; – INIT
4	Пастки попередньої команди: – виключення контрольних точок (INT 3); – виключення налагоджування та покрокової работи процесора (біт TF та контрольні точки по даних та пристроям введення-виведення).
5	Зовнішні переривання: – NMI переривання; – масковані апаратні переривання.
6	Відмови від вибірки наступної команди: – виключення контрольної точки по командам; – порушення границі коду сегмента[1]; – відмова сторінки коду[1]
7	Відмови від дешифрування наступної команди: – довжина команди > 15 байтів; – невірний код команди; – недоступний FPU.

Закінчення таблиці 2.3

1	2
8	Відмови від виконання команди: – виключення FPU; – знакове переповнення; – помилки перевірки діапазону; – недійсний TSS; – сегмента немає в пам’яті; – помилка стеку; – помилка загального захисту; – помилка сторінки даних; – помилка вирівняння даних.

Виключення більш низького пріоритету відкидаються, переривання більш низького пріоритету залишаються активними. Відкинуті виключення будуть повторно згенеровані, коли оброблювач виключення передасть управління тієї команді, при виконанні якої виникло переривання.

2.3 Формат коду помилки

При обслуговуванні ряду виключень (8, 10-14, 17) процесор формує код помилки, що заноситься в стек слідом за вмістом регістрів EFLAGS, CS та лічильника EIP та може аналізуватися процедурою оброблення переривання.

На рисунку 2.1 приведений формат 32-розрядного коду помилки для виключень 10, 11, 12, та 13, який має такі поля:

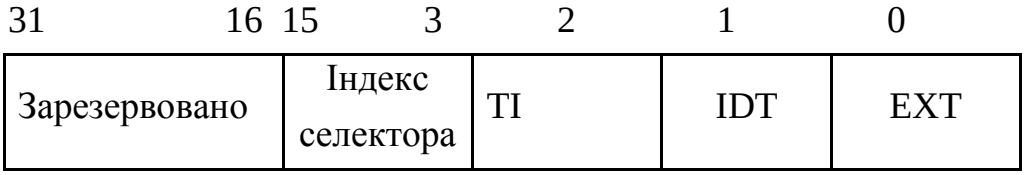

Рисунок 2.1 – Формат помилки для виключень 10-13

Біт EXT (External), якщо встановлений, вказує, що подія є зовнішньою по відношенню до програми, де виникло виключення.

Біт IDT, якщо встановлений, вказує, що поле індексу селектора в коді

помилки відноситься до дескриптора шлюзу в таблиці IDT, якщо біт скинутий, то це вказує, що поле індексу селектора відноситься до дескриптора в таблиці GDT або до дескриптора в поточній таблиці LDT.

Біт TI застосовується тільки коли біт IDT = 0. Якщо біт TI встановлений, то поле індексу селектора відноситься до дескриптора сегмента чи шлюзу в поточній LDT, якщо скинутий – до дескриптора сегмента в GDT.

Поле Індекс селектора вказує на дескриптор сегмента чи шлюзу в таблицях IDT, LDT та GDT, звернення до якого викликало виключення.

Якщо код помилки (його молодші 16 розрядів) дорівнює нулю, то це означає, що помилка не була пов'язана з посиланням до конкретного сегмента, або в команді було здійснене посилання до нуль-дескриптора.

Для виключень 8 та 17 всі поля коду помилки завжди мають нулеві значення.

На рисунку 2.2 приведений формат 32-розрядного коду помилки для виключення 14, який має такі поля:

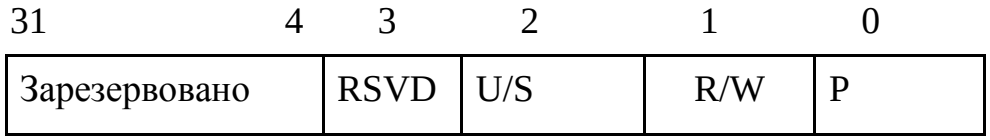

Рисунок 2.2 – Формат помилки для виключення 14

Біт P (Present) вказує, що виключення викликане зверненням до сторінки, яка відсутня в пам'яті (P = 0), або з-за порушення захисту сторінки (P = 1).

Біт W/R (Write/Read) вказує на виникнення виключення з-за порушення заборони читання сторінки (W/R = 0) або запису (W/R = 1) в сторінку;

Біт U/S (User/Supervisor) вказує, що виключення виникло при виконанні системної програми (U/S = 0), або при виконанні програми користувача (U/S = 1);

Біт RSVD (Reserved) вказує, що процесор виявив одиниці в зарезервованих бітах каталогу сторінок при встановлених бітах PSE або PAE регістра CR4 (біт PSE доступний тільки починаючи з процесорів Pentium та сімейства P6, а біт PAE – тільки з процесорів P6).

Усі інші розряди коду помилки мають невизначене значення.

Якщо оброблювач виключення буде викликаний не апаратними засобами

процесора, а за допомоги програмного переривання INT n, де n – номер виключення, то в цьому випадку код помилки не буде занесений в стек для жодного виключення, хоча оброблювач виключення буде виконуватися.

2.4 Формат дескрипторної таблиці переривань

Оброблення переривань у захищеному режиму відрізняється від оброблення переривань в реальному режиму тим, що замість завдання 32-розрядних векторів, які містять 16-розрядні сегмент та зміщення адреси оброблювачів переривань та зберігаються в області пам’яті об’ємом 1 КБ, починаючи з адреси 0, треба для кожного переривання сформувати 64-розрядний дескриптор шлюзу оброблювача переривання й записати його в дескрипторну таблицю переривань IDT (Interrupt Descriptor Table), яка може розташуватися в довільному місці пам’яті.

Дескриптори шлюзів на відміну від дескрипторів сегментів, які були розглянуті в розділі 4, задають параметри не сегментів, тобто областей пам’яті, а параметри (адреса, тип і атрибути доступу) окремих комірок пам’яті (шлюзів), куди буде передано управління.

Дескриптори оброблювачів переривань таблиці IDT мають формат шлюзів виклику з одною відміною – у них в полі кількості параметрів (байт 4 дескриптора) завжди вказуються нулі (рис. 2.3).

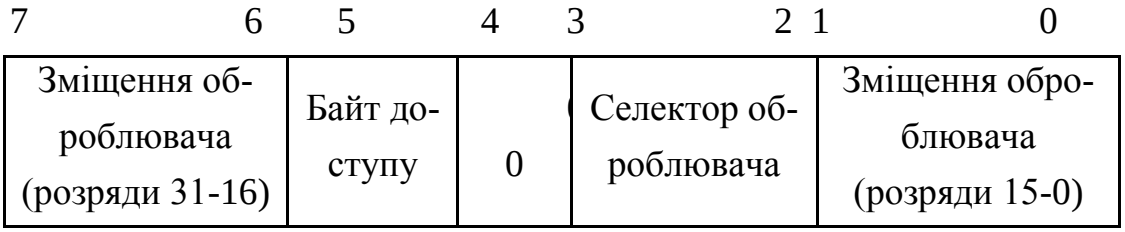

Рисунок 2.3 – Формат дескриптора шлюзу для обробки переривань

У таблиці IDT можуть міститись дескриптори шлюзів трьох типів: шлюзи задач (задачі можуть переключатися через переривання), шлюзи переривань та шлюзи пасток.

Дескриптори шлюзу задач із полів, представлених на рис 2.3, використовують тільки два поля: поле селектора сегмента стану задачі та поле байта доступу. Застосування шлюзів задач дивитесь в розділі 3 “ОРГАНІЗАЦІЯ МУЛЬ-

ТІЗАДАЧНОЇ РОБОТИ МІКРОПРОЦЕСОРА”

На рис. 2.4 наведений формат байтів доступу шлюзів задач (а), переривань (б) і пасток (в), який є байтом доступу типу системного сегмента.

7	6	5	4	3	2	1	0
P	DPL		S=0	0	1	0	1

а)

P	DPL		S=0	D	1	1	0

б)

P	DPL		S=0	D	1	1	1

в)

Рисунок 2.4 – Формат байта доступу шлюзів

а) задачі;

б) переривання;

в) пастки

Біти ***P, S*** та поле ***DPL*** мають те ж саме призначення, що в дескрипторах сегментів пам’яті (дивись підрозділ 1.4.1).

Біт D визначає розрядність шлюзу оброблювача переривань, тобто розрядність лічильника команд EIP, поля для значення регістра CS та регістра EFLAGS, які будуть занесені в стек, а також розрядність коду помилки:

– D = 0: 16 розрядів;

– D = 1: 32 розряди.

Відмінність шлюзів переривання і пастки полягає в наступному:

– при виклику процедури обслуговування через шлюз переривання скидається прапор IF в регістрі EFLAGS (як це робиться в реальному режиму);

– при виклику процедури обслуговування через шлюз пастки прапор IF не скидається (це робиться, коли небажано вимикати механізм розподілу часу, що використовує переривання від таймера).

2.5 Дії процесора при обробленні переривання

При кожному перериванні, незалежно від його типу процесор виконує такі дії:

1. Визначає номер переривання і, згідно з цим номером, зчитує з таблиці IDT дескриптор цього переривання. Номер переривання визначається порізному в залежності від типу переривання:

– при зовнішніх перериваннях, що маскуються, номер переривання по запиту процесора за допомоги двох циклів обміну даними передає програмувальний контролер переривань;

– при зовнішніх перериваннях, що не маскуються, номер переривання дорівнює 2;

– при програмних перериваннях номер переривання n задається в команді INT n;

– при виключеннях номер переривання виробляється внутрішніми схемами МП.

2. Аналізує байт доступу дескриптора переривання й визначає тип шлюзу.

2.1. Якщо це шлюз задачі, то здійснює переключення на задачу-оброблювача переривання. У цьому разі пункти 3 та 4 не виконуються.

2.2. Якщо це шлюз переривання або пастки, то аналізується біт D дескриптора переривання. Якщо він дорівнює 1, що вказує на 32-розрядне переривання, то в стек заносяться три 32-розрядних слова в такому порядку: вміст регістрів EFLAGS і CS та лічильника команд EIP. При цьому два старші байти 32-розрядного слова для 16-розрядного регістра CS не мають значення. Вміст регістрів CS і EIP задає адресу команди, яка буде виконуватись після завершення обробки переривання.

Якщо в дескрипторі переривання в байті доступу біт D = 0, що вказує на 16-розрядне переривання, то в стек заносяться три 16-розрядних слова: вміст FLAGS, CS та IP.

Відзначимо, що виключення типу відмова зберігають в стеку адресу саме тієї команди, яка його викликала (переривання з номерами 0, 5-7 та 10-17). Це дає можливість повторного запуску програми (рестарту програми) після того, як процедура оброблення переривання усуне причину виникнення виключення.

Виключення типу пастки (переривання з номерами 1, 3 та 4) зберігають в стеку адресу наступної команди. Якщо пастка виявлена при виконанні команди

переходу, наприклад, команди JMP, то в стек заноситься адреса не наступної команди, а адреса саме переходу.

Для виключень 8 та 18, які мають тип «вихід з режиму», не завжди звісна адреса команди, при виконанні якої вони виникли, тому оброблювачі цих виключень, як правило, повинні закінчувати виконання поточної програми.

Мікропроцесор аналізує запити на переривання від зовнішніх пристроїв на границях машиних команд і після прийняття їх на оброблення зберігає в стеку адресу наступної команди.

Якщо запит на зовнішнє переривання виник під час виконання команди з префіксом повторювання, то переривання буде прийнято на оброблення в кінці поточної ітерації.

При обробленні програмних переривань МП заносить в стек адресу наступної команди.

Крім того, якщо в дескрипторі оброблювача переривання указаний шлюз переривання, то в регістрі прапорів EFLAGS очищається прапор IF.

3. Для виключень 8, 10-14, 17 додатково в стек заноситься 32-розрядний код помилки, що уточняє причину виникнення виключення (формат коду помилки приведений в підрозділі 2.3). Якщо в байті доступу шлюзу оброблювача переривання біт T = 0 – в стек заноситься 16-розрядний код помилки.

4. По дескриптору переривання процесор визначає адресу (селектор та зміщення) оброблювача цього переривання та передає йому управління.

5. При виконанні 32-розрядної команди IRET (DB 66h IRET), якою завершується робота оброблювача переривання, з стека беруться 32-розрядні значення EIP, CS та EFLAGS та завантажуються у відповідні регістри процесора, при цьому МП повертається до виконання перерваної програми (при виконанні 16-розрядної команди IRET з стека беруться 16-розрядні значення IP, CS та FLAGS).

Якщо оброблювачем переривання є задача, то по команді IRET здійснюється переключення назад на задачу, при виконанні якої виникло переривання.

У процесорах з динамічним (по припущенню) виконанням команд (МП сімейства P6 та наступні) оброблення переривань та виключень здійснюється під час фази вивантаження (retirement), тобто у порядку розташування команд в програмі.

2.6 Організація захищеного режиму з обробленням переривань

Для того щоб здійснити оброблення всіх типів переривань (програмних, виключень та зовнішніх апаратних) у захищеному режиму, необхідно розробити програмні засоби, які повинні виконати такі дії:

1) розробити оброблювачі переривань;

2) сформувати глобальну дескрипторну таблицю, що містить дескриптори всіх сегментів пам'яті, які будуть використовуватися в програмі;

3) задати базову адресу і розмір глобальної дескрипторної таблиці в регістрі GDTR;

4) сформувати дескрипторну таблицю переривань, яка містить дескриптори шлюзів всіх розроблених в програмі оброблювачів переривань (апаратних і програмних);

5) заборонити масковані та немасковані переривання;

6) задати базову адресу і розмір дескрипторної таблиці переривань;

7) перепрограмувати контролери переривань для роботи у захищеному режимі;

8) зберегти в пам'яті вміст регістрів МП;

9) перевести мікропроцесор у захищений режим;

10) виконати у захищеному режимі потрібні дії, в тому числі виконати оброблення виникаючих переривань;

11) повернутися в реальний режим;

12) відновити вміст регістрів МП;

13) перепрограмувати контролери переривань для роботи в реальному режимі;

14) дозволити переривання, що маскуються, та ті, що не маскуються.

Якщо зробити порівняння цих дій з діями з організації захищеного режиму мікропроцесора, які були розглянуті в розділі 1 "ОРГАНІЗАЦІЯ РОБОТИ МІКРОПРОЦЕСОРА У ЗАХИЩЕНОМУ РЕЖИМІ", то можна зробити такі висновки:

1. Частина дій, які виконувалися в програмі **P_MODE**, відсутня – це стосується визначення типу мікропроцесора та формування даних для організації повертання МП до реального режиму шляхом скидання мікропроцесора.

2. Ці дії можна не виконувати через те, що зараз 16-розрядні МП практично не використовуються, і тому питання обробки переривань у захищеному

режимі будуть розглядатися тільки для 32-розрядних мікропроцесорів, для яких повертання МП в реальний режим через його скидання є незручним і необов'язковим.

3. Друга частина дій – такі, як завдання базової адреси і розміру таблиці GDT, заборона та дозвіл переривань, які маскуються та тих, що не маскуються, зберігання в пам'яті та відновлення вмісту регістрів МП, в цілому співпадає і тому не буде розглядатися в даному розділі (дивиться описання цих дій в розділі 4).

4. Третя частина дій – такі, як формування глобальної дескрипторної таблиці, переведення МП у захищений режим та його повертання до реального режиму мають відмінності, які пояснюються в відповідних підрозділах.

5. Всі інші дії, які пов'язані з обробленням переривань у захищеному режимі: розробка оброблювачів програмних та апаратних переривань, формування дескрипторної таблиці переривань, завдання її базової адреси й розміру, перепрограмування контролерів переривань для роботи у захищеному та реальному режимах, є новими і тому детально розглядаються далі в цьому розділі.

Усі вищеописані дії з організації захищеного режиму з обробленням переривань здійснені в програмі **P_INT** (повний текст програми приведений в підрозділі 2.18), яка розроблена з навчальною метою і може бути застосована як базова при створенні програм, призначених для роботи у захищеному режимі.

Програма **P_INT** виконується в такому же програмному середовищі, що і програма **P_MODE** (дивись підрозділ.1.2), теж підключає модуль **PROT** з використанням констант, типів змінних, змінних, функцій й процедур, що задані в модулі (опис та текст модуля **PROT** наведені відповідно в підрозділах 1.14 та 1.16)**.**

2.7 Розроблення оброблювачів переривань

2.7.1 Розподіл номерів переривань

Щоб мати можливість у захищеному режимі здійснити оброблення всіх видів переривань в програмі **P_INT** перш за все був зроблений розподіл всіх 256 номерів переривань:

– перші 32 номери з 0 по 31 згідно з рекомендаціями фірми Intel віддані для нумерації виключень;

– 16 наступних номерів з 32 (20h) по 47 (2Fh) зарезервовані для зовнішніх апаратних переривань;

– решта номерів, починаючи з номера 48 (30h), віддана під програмні переривання.

2.7.2 Розробка оброблювачів програмних переривань

У програмі **P_INT** застосовуються функції двох програмних переривання – INT 30h та INT 32h.

2.7.2.1 Функції переривання INT 30h

За допомоги функцій переривання INT 30h, оброблювач якого реалізований в вигляді процедури **int_30h**, здійснюється робота з відеопам'яттю в текстовому режимі.

Ця процедура поміщена в модуль **PROT**, так як застосовується при роботі з відеопам'яттю не тільки програмою **P_INT**, але також іншими програмами, що працюють у захищеному режимі (крім програми **P_MODE**).

Функції переривання INT 30h реалізують режим прямого доступу до відеопам'яті. Необхідність розроблення цих функцій пояснюється тим, що у захищеному режимі застосовувати функції та переривання BIOS та MS DOS для роботи з відеопам'яттю неможливо (це можна здійснити тільки після переходу до спеціального режиму VM 8086).

Вхідними параметрами при викликах переривання INT 30h є дані, які повинні бути завантажені у відповідні регістри МП і мати такі значення:

– AH – номер функції;

– AL – номер підфункції;

– BH/BL – номер стовпця/рядку екрана;

– DL/DX/EDX – значення даних (байту/слова/подвійного слова), які виводяться на екран;

– CL – відеоатрибут символів;

– SI – зміщення рядка символів в пам'яті;

– DI – зміщення комірки відеопам'яті.

Для переривання 30h розроблені такі функції:

– функція 0 (AH = 0) – установка призначення біта 7 відеоатрибуту (із збереженням вмісту усіх регістрів):

а) AL = 0 – режим мерехтіння символу;

б) AL = 1 – режим 16-розрядного фону;

– функція 1 (AH = 1) – установка маркера в задану точку екрана (номер стовпця, номер рядку) із збереженням вмісту усіх регістрів, окрім DI:

а) AL = 0 – без зміни екрана;

б) AL = 1 – з очищенням екрану від маркера до кінця рядку;

в) AL = 2 – з очищенням екрану від маркера до кінця екрана (в CL – колір фону екрана після очищення);

г) AL = 3 – з підсвічуванням маркера;

– функція 2 (AH = 2) – виведення символу на екран;

– функція 3 (AH = 3) – виведення даних на екран в 16-річній формі:

а) AL = 0 – виведення байту;

б) AL = 1 – виведення слова (16 розрядів);

в) AL = 2 – виведення подвійного слова (32 розряди);

– функція 4 (AH = 4) – виведення рядка символів на екран.

– функція 5 (AH = 5) – виведення даних на екран в двійковій формі:

а) AL = 0 – виведення байта;

б) AL = 1 – виведення слова (16 розрядів);

в) AL = 2 – виведення подвійного слова (32 розряди);

– функція 6 (AH = 6) – виведення даних на екран в 10-річній формі:

а) AL = 0 – виведення байту;

б) AL = 1 – виведення слова (16 розрядів);

в) AL = 2 – виведення подвійного слова (32 розряди);

– функція 7 (AH=7) – робота із прямокутною областю екрана, що обмежена координатами (X1,Y1) – лівий верхній кут області екрана (регістр BX); та (X2,Y2) – правий нижній кут області екрана (регістр DX):

а) AL = 0: зберегти зображення області екрана в пам’яті;

б) AL = 1: відновити збережене зображення на екрані.

Особливістю процедури **int_30h** є те, що вона розроблена для використання як з захищеному режимі, так і в реальному режимі роботи мікропроцесора. Тому дескриптор оброблювача цього переривання має 16-розрядний байт доступу пастки **acc_trap_16.** Крім цього процедура завершується 16-розрядною командою **iret,** в той час як процедури обробки 32-розрядних переривань закінчуються 32-розрядною командою **db 66h iret**.

2.7.2.2 Використання переривання INT 30h в реальному режимі

Щоб використати функції переривання 30h в реальному режимі треба спочатку занести адресу процедури оброблювача цього переривання **(@int_30h)** до таблиці векторів за номером $30:

meml[0:4*$30]:=longint(@int_30h),

так як операція **@int_30h** має тип **pointer**, а ліва частина оператора присвоювання має тип **longint**, то в операторі здійснюється узгодження типів.

При роботі процесора в реальному режиму за допомоги функцій 1 та 4 переривання 30h в програмі **P_INT** здійснено (див. рис. 2.5):

а) очищення всього екрану (замість використовування процедури **clrscr** мови Borland Pascal):

```
mov ax,102h
mov cl                          { встановлюється фон чорного кольору }
mov bx,0
int 30h;
```

б) виведення на екран (2,1) повідомлення "Работа в реальном режиме:" (текст повідомлення заданий за допомоги типізованої константи мови Borland Pascal – **s_real:string='Работа в реальном режиме:';**):

```
mov ax,100h      { Функція 1, підфункція 0: установка маркера }
mov bx,0501h     { (стовпець 5, рядок 1) }
int 30h
mov ah,4              { Функція 4: виведення рядка символів на екран }
mov cl,1eh          { відеоатрибут: жовтий символ на синьому фоні }
mov si,offset s_real                { В SI-  адреса рядка символів s_real }
int 30h                                          { Виведення рядка на екран }.
```

Застосування функцій переривання INT 30h у захищеному режимі описано в підрозділі 2.13.

2.7.2.3 Функції переривання INT 32h

Програмне переривання INT 32h, оброблювач якого реалізований в вигляді процедури **int_32h** модуля **PROT**, застосовується в програмі **P_INT** з метою спростити виведення на екран довідкової та налагоджувальної інформації.

Переривання INT 32h також може бути застосовано як в реальному ре-

жимі, так і у захищеному. Але на відміну від переривання INT 30h механізм реалізації цієї можливості інший.

У процедурі **int_32h** перед виконанням будь-якої функції визначається режим роботи процесора. Це здійснюється шляхом читання вмісту регістра управління CR0. Якщо молодший біт цього регістру (біт PE) дорівнює 0 – процесор працює в реальному режиму, якщо 1 – процесор працює у захищеному режиму. При цьому згідно із значенням біту PE встановлюється значення змінної **mode**, яке в подальшому використовується в функціях в тих випадках, коли їх дії залежать від того, в якому режимі працює процесор.

Наприклад, це відбувається при роботі зі стеком, так як в реальному режимі в стек заносяться. 16-розрядні значення, а у захищеному при завданні 32-розрядного оброблювача (як це зроблено з оброблювачем переривання 32h) – 32-розрядні.

При викликах переривання INT 32h в регістрі AH задається номер функції, а в регістрах BH та BL – відповідно номер стовпця та номер рядку екрана, куди буде виведена задана інформація (крім функції 1, для якої по умовчанню задаються такі значення BL = 18, BH = 0).

Для переривання 32h розроблені такі функції:

1. Функція 1 (AH = 1) застосовується при виявленні в ході роботі програми виключень й здійснює виведення на екран таких даних: повідомлення: "**Runtime error: исключение**", номера виключення та, при необхідності, адреси команди, при виконанні якої виникло виключення (або наступної команди).

При цьому:

– регістр DL задає номер виключення;

– регістр DH, який може мати значення 0 чи 1, вказує, чи потрібно (при DH = 1), чи ні (при DH = 0) виводити на екран окрім номеру виключення також адресу команди, що причинила переривання.

Треба відзначити, що виведення адреси команди можливо тільки в тому випадку, якщо стек після виклику оброблювача переривання не був змінений. Крім того, номери стовбцю та рядку екрана, куди виводиться повідомлення про виключення, що виникло, для цієї функції не задаються (вони фіксовані і дорівнюють відповідно 1 та 17).

2. Функція 2 (AH = 2) здійснює виведення на екран найменувань та вмісту регістрів загального призначення: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. Особливість цієї функції: перед її викликом потрібно зберегти в стеку значення

регістрів AX та BX, а після виклику функції – відновити їх вміст зі стеку.

При цьому регістр AL задає режим виводу найменувань та вмісту регістрів:

– AL = 0: в один рядок екрана;

– AL = 1: в два рядки екрана.

3. Функція 3 (AH = 3) здійснює виведення на екран вмісту лічильника команд EIP та сегментних регістрів CS, SS, ES, DS, FS та GS. У регістрі AL задається номер підфункції:

– AL = 0: виведення безпосередніх значень регістрів CS та EIP;

– AL = 1: виведення "тіньових" значень регістрів CS та EIP, тобто таких, які містяться в стеку.

У випадку AL = 1 потрібно перед викликом функції 3 задати:

– в CX – значення CS;

– в EDX – значення EIP.

4. Функція 4 (AH = 4) здійснює виведення на екран:

– при AL = 0 значень бітів регістру управління CR0;

– при AL = 1 найменувань та значень бітів регістру CR0.

5. Функція 5 (AH = 5) здійснює виведення на екран:

– при AL = 0 значень бітів регістру управління CR4;

– при AL = 1 найменувань та значень бітів регістру CR4.

6. Функція 6 (AH = 6) здійснює виведення на екран:

– при AL = 0 значень бітів регістру прапорів EFLAGS;

– при AL = 1 найменувань та значень бітів регістру прапорів EFLAGS.

2.7.2.4 Використання переривання INT 32h в реальному режимі

Аналогічно перериванню 30h адреса процедури оброблювача переривання 32h перед використанням повинна бути занесена до таблиці векторів за номером $32:

meml[0:4*$32]:=longint(@int_32h).

При роботі процесора в реальному режиму за допомоги функцій 3 та 4 переривання 32h в програмі **P_INT** здійснено:

1) виведення на екран (3,1) назви та вмісту регістрів EIP CS, SS, DS, ES,

FS, GS:

```
mov ax,300h      { Функція 3 }
mov bx,103h
int 32h
```

2) виведення на екран (5,6) найменувань та значень бітів регістра управління CR0:

```
mov ax,401h      { Функція 4 }
mov bx,506h
int 32h.
```

Зображення екрану після виконання функцій 3 та 4 переривання 32h приведене на рис 2.5.

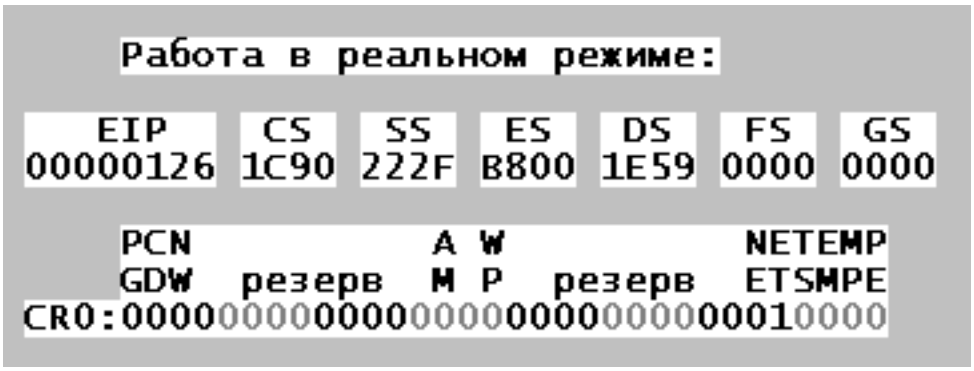

Рисунок 2.5 – Дані, виведені на екран в реальному режимі

Як це видно з рисунку, процесор дійсно працює в реальному режимі, оскільки значення сегментних регістрів є базовими адресами сегментів пам’яті, біт PE регістра управління CR0 дорівнює 0.

Застосування функцій переривання INT 32h у захищеному режимі описано в підрозділі 2.13.

2.7.3 Розроблення оброблювачів виключень

Оскільки при роботі програм у захищеному режимі в будь-який момент може виникнути непередбачене виключення, необхідно мати заздалегідь сформовані оброблювачі всіх без винятку виключень.

У модулі **PROT** містяться процедури **exc_00-exc_08, exc_10, exc_11,**

exc_13, exc_14 та **exc_16-exc_18** та задача **exc12_task,** які розроблені для оброблення виключень з номерами 0-8, 10-14 та 16-18 (виключення з номерами 9 та 15 в сучасних процесорах не застосовуються), і можуть бути використані будь-якою програмою у захищеному режимі. Кожна з процедур при виникненні виключення виконує такі дії:

1. Виводить на екран за допомоги функції 1 програмного переривання 32h номер виключення та (при DH = 1) адресу команди в форматі селектор:зміщення, при виконанні якої виникло це виключення (для виключень відмов) або наступної команди (для виключень пасток). Наприклад, для процедури **exc_00**, що обробляє виключення 0, це виглядає так:

mov ah,1
mov dx,100h
int 32h.

2. Завершує роботу програми у захищеному режимі шляхом виконання команди непрямого міжсегментного переходу до мітки **end_prot**, якою завершуються дії процесора у захищеному режимі та починається перехід до реального режиму.

Саме ця команда застосована тому, що адресу мітки в програмі, до якої підключається модуль, він може отримати тільки через змінну, в яку програма повинна адресу мітки занести. Команда непрямого міжсегментного переходу реалізована в вигляді машинного коду

db 0ffh,2eh
dw ofs_end_prot,

де **ofs_end_prot** – змінна, в яку програма **P_INT** до переходу у захищений режим записує зміщення мітки **end_prot**:

asm mov ofs_end_prot,offset end_prot end;,

а в наступну змінну **sel_end_prot** заносить селектор сегмента коду цієї мітки, тобто селектора коду програми **P_INT**:

sel_end_prot:=code_sel;.

Для правильної роботи команди непрямого міжсегментного переходу необхідно, щоб в пам'яті ці дві змінні стояли обов'язкового поруч (як це об'явлено в модулі **PROT**).

Оскільки виключення 12 ***#SS*** (Stack Segment Fault) виникає при порушенні нормальної роботи стека (див. підрозділ 5.1), то викликати процедуру для оброблення цього виключення, як це було зроблено для інших виключень, не можна через те, що при виклику будь-якої процедури в стек заноситься адреса наступної команди, що приведе при непрацездатному стеку до зависання програми.

Єдиним рішенням цієї проблеми є розробка оброблювача виключення 12 не в вигляді процедури, а в вигляді задачі **EXC12_TASK** (модуль **PROT**), при виклику якої здійснюється переключення до нового стеку, який задача буде використовувати в своїй роботі замість старого стеку (про розробку й переключення задач дивись в розділі 6 "ОРГАНІЗАЦІЯ БАГАТОЗАДАЧНОЇ РОБОТИ").

Задача **EXC12_TASK,** як і процедури оброблення інших виключень, за допомоги функції 1 переривання 32h виводить на екран повідомлення про виникнення виключення в роботі програми й номер виключення – 0C без виведення адреси команди (DH = 0), оскільки ця інформація при переключенні задач в стек не заноситься.

Є дві причини виникнення виключення 12: відсутність сегмента стека в пам'яті (біт P дескриптора сегмента стека дорівнює 0) та перевищення розміру стека (регістр ESP вказує на комірку пам'яті, адреса якої дорівнює або менша границі сегмента стека). Тому в процедурі **EXC12_TASK** спочатку з поля SS сегмента стану задачі **MAIN** (програми **P_INT**) зчитується значення селектора сегмента стека, по якому в таблиці GDT знаходиться дескриптор цього сегмента й аналізується значення біта P:

mov bx,[bx+80] *{ з TSS* зчитується *селектор стека }*
test byte ptr [offset gdt+bx+5],80h.

Якщо P = 0, то здійснюється його встановлення в дескрипторі сегмента стека

or byte ptr [offset gdt+bx+5],80h.

і робота програми **P_INT**продовжується.

Якщо P = 1, то це означає, що причиною виникнення виключення 12 є перевищення розміру стека. У цьому випадку і сегментіTSS змінюються значення укажчика стека в полі ESP:

```
mov bx,offset main_tss
mov ax,0fffh                          { Значення 0FFFh }
mov [bx+56],ax              { заноситься в поле ESP сегменту TSS }
```

та значення поля EIP:

```
mov ax,ofs_end_prot                   { Зміщення мітки end_prot }
mov [bx+32],ax              { заноситься в поле EIPсегмента TSS }
```

для завершення роботи у захищеному режимі.

Крім розроблення самої задачі **EXC12_TASK** в секції ініціалізації модулю **PROT** здійснені такі дії:

– формування сегменту стану задачі **EXC12_TASK**;

– формування дескрипторів сегментів стану задач **MAIN** та **EXC12_TASK** та занесення їх до таблиці GDT.

Кожна програма, як буде використовувати оброблювач переривання 12 повинна у захищеному режимі (тому це неможливо зробити в модулі **PROT**) завантажити селектор задачі **MAIN** в регістр задач TR:

```
db 0fh,0,1eh            { LTR main_sel: }
dw v_main_sel.
```

Як повідомлялось вище, тільки при виникненні виключень-відмов мікропроцесор заносить в стек адресу тієї команди, яка саме і викликала це виключення, в той час як при виникненні виключень-пасток та апаратних зовнішніх та програмних переривань в стек заноситься адреса наступної команди.

Тому на розробку оброблювачів виключень-відмов треба звернути особливу увагу, бо якщо оброблювач виключення після виведення на екран налагоджувальної інформації, наприклад, номеру переривання завершить свою роботу

виконанням команди IRET то процесор візьме з стеку адресу наступної команди і передасть їй управління. Оскільки це буде та ж сама команда, що викликала виключення, то вона знову згенерує те ж саме виключення, і таким чином програма зациклиться.

Щоб цього не скоїлось оброблювачі виключень можуть використати один з трьох можливих варіантів оброблення відмов:

1) достроково завершити виконання програми;

2) обійти команду, яка викликала відмову;

3) усунути причину відмови.

У програмі **P_INT** у захищеному режимі здійснюється генерація (зрозуміло в навчальних цілях) та оброблення таких виключень-відмов:

– "Ділення на нуль" (виключення 0);

– "Вихід за мережі діапазону" (виключення 5);

– "Відсутність сегмента даних в пам’яті" (виключення 11).

При цьому застосовуються всі три вище названі варіанти оброблення виключень-відмов:

1. Перший варіант оброблення відмов використовується при розробці оброблювачів-заглушок для всіх можливих в сучасних мікропроцесорах виключень (від виключення 0 до виключення 18) в вигляді стандартних процедур **exc_00-exc_08, exc_10, exc_11, exc_13, exc_14** та **exc_16-exc_18** та задачі **exc12_task**, які після підключення модуля **PROT** при виникненні будь-якого виключення здійснюють автоматичне його оброблення.

Тому в програмі **P_INT** для перевірки варіанту 1 не розроблявся свій оброблювач виключення 0 – для оброблення цього виключення була застосована стандартна процедура **exc_00** з модулю **PROT**, яка при виникненні виключення 0 виводить на екран номер виключення та адресу команди, при виконанні якої виникло виключення, в вигляді повідомлення, яке приведено на рисунку 5.9 підрозділу 5.13).

2. Другий варіант оброблення відмов застосовує оброблювач виключення 5, який реалізований в програмі **P_INT** в вигляді процедури **my_exc_05**, що використовується в програмі замість стандартної процедури **exc_05** модуля **PROT** та виконує такі дії:

а) зберігає вміст регістрів загального призначення за допомоги команди **pusha**, оскільки після виконання оброблення виключення 5 управління буде передано програмі **P_INT** для продовження її виконання, і треба забезпечити, щоб

регістрі загального призначення мали ті ж самі значення, які вони мали до виклику оброблювача виключення;

б) виводить на екран з налагоджувальними цілями значення регістрів загального призначення шляхом виклику переривання INT 32h з застосуванням функції 2:

```
push ax          { Необхідне для роботи функції 2 зберігання в стеку }
push bx          { вмісту регістрів AX та BX }
mov ax,201h      { Функція 2/підфункція1 переривання 32h: }
mov bx,113h      { виведення на екран (1,19) }
int 32h          { значень регістрів загального призначення }
pop bx           { у два рядки (для компактності) }
pop ax;
```

в) Змінює в стеку значення лічильника команд EIP на 4 байти, тобто на величину довжини команди BOUND

```
mov bp,sp
add word ptr [bp+16]
```

і, тим самим, забезпечує передачу управління наступній команді після закінчення роботи оброблювача (значення покажчика стека змінено на 16 байтів в зв'язку з виконанням команди **pusha**). Таким чином може бути заборонена перевірка виходу за межі масиву;

г) за допомоги функції 1 переривання 32h виводить на екран повідомлення: "**Runtime error: исключение** " та номер виключення:

```
mov ah,1
mov dx,5h
int 32h;
```

д) очікує натиснення клавіші для продовження роботи програми:

```
@w:cmp scan,0
   jz @w
   test scan,80h
```

jnz @w;

е) завершує оброблення виключення 5:

popa
db 66h
iret.

У підрозділі 2.13 приведене зображення екрану після виведення всіх повідомлень та даних оброблювачем **my_exc_05.**

3. Третій варіант оброблення відмов застосовує оброблювач виключення 11, який реалізований в програмі **P_INT** в вигляді процедури **my_exc_11,** й імітує режим обміну даними між фізичною та віртуальною пам'яттю у захищеному режимі. Тому, коли виконується команда завантаження до сегментного регістру даних DS дескриптора, у якого в байті доступу скинуто біт P (імітація відсутності сегмента в фізичній пам'яті), виникає виключення 11.

Процедура **my_exc_11**, яка є оброблювачем цього переривання, читає із стеку код помилки (формат коду помилки дивиться в підрозділ 2.3) та визначає за його допомоги тип дескрипторной таблиці – GDT і селектор дескриптора (його значення – 30h, тобто це сегмент даних). Після цього процедура в визначеному дескрипторі таблиці GDT встановлює біт P = 1, імітуючи тим самим завантаження цього сегмента з магнітного диску до пам'яті.

Оскільки після оброблення цього виключення управлення передається знову до команди завантаження сегментного регістру DS, то тепер відбувається нормальне виконання команди і в подальшому сегмент даних є доступним до читання даних чи до запису даних.

Таким чином це виключення виявилось непомітним (прозорим) для команди, що завантажувала сегментний регістр. Так само при реальному обміну даними між фізичною та віртуальною пам'яттю відсутній в пам'яті сегмент завантажується з магнітного диска прозоро для програми користувача.

Крім цього в процедурі **my_exc_11** аналізується біт розрядності в байті доступу шлюзу обробника переривання 11, що робить цей обробник незалежним від розрядності шлюзу, і відповідно до його розрядності (16 чи 32 бітів), виконуються очищення стеку від коду помилки та завершення процедури командою IRET.

У ході роботи програми **P_INT** при виникненні виключення 11 процедура **my_exc_11** виводить на екран (1,17) за допомоги функцій 1, 4 та 5 переривання 30h повідомлення "**Исключение:11 | Код ошибки:**", номер виключення та код помилки в двійковій формі (дивись підрозділ 1.13) і очікує натиснення клавіші для продовження роботи програми.

2.7.4 Розробка оброблювачів зовнішніх апаратних переривань

Чипсети комп'ютерів архітектури PC AT в своєму складі мають два програмувальних контролера переривань PIC. Один з них є ведучим, другий – веденим. До кожного з ПКП може бути підключено до 8 зовнішніх пристроїв.

Таким чином комп'ютер має 15 входів для підключення запитів на переривання IRQ (Interrupt Request) від зовнішніх пристроїв: IRQ0, IRQ1, IRQ3-IRQ7 (для ведучого PIC) та IRQ8-IRQ15 (для веденого PIC). Вхід IRQ2 застосовується в комп'ютері для підключення веденого контролера до ведучого.

Для оброблення апаратних переривань у захищеному режимі від можливих 15 зовнішніх пристроїв розроблені оброблювачі-заглушки в вигляді процедури **PIC_1** для семи переривань першого контролера переривань (крім оброблювача переривань від клавіатури) та процедури **PIC_2** для восьми переривань другого ПКП (модуль **PROT**).

Процедура **PIC_1** здійснює скидання першого PIC шляхом використання команди контролера переривань з кодом 20h, яка закінчує оброблення поточного переривання:

```
mov al,20h
out 20h,al.
```

Якщо цю команду не видати, то контролер переривань не зможе далі обробляти запити на будь-яке переривання, чий привілей нижче чи рівний привілею поточного запиту.

Процедура **PIC_2** відрізняється від процедури **PIC_1** тим, що здійснює скидання зразу двох контролерів переривання – першого ПКП та другого ПКП:

```
mov al,20h
out 20h,al
ut 0a0h,al.
```

У програмі **P_INT** не розроблюється свій оброблювач переривань від клавіатури. Замість цього застосовується оброблювач переривань від клавіатури модуля **PROT**, розроблений в вигляді процедури **keyb**, яка викликається при натисненні та відтисненні клавіш клавіатури й виконує такі дії:

– зберігає в стеку вміст РОН;

– здійснює скидання першого контролера ПКП (так же, як в процедурі **PIC_1**);

– читає скан-код клавіші з порту 60h в регістр AL**;**

– визначає стан клавіші “лівий Shift” і зберігає його в змінній **left_shift**: якщо ця клавіша натиснута, то **left_shift** = 1, якщо відтиснута – **left_shift** = 0**;**

– перевіряє натиснення комбінації клавіш “лівий Shift” та “Pause/Break” і, якщо ці клавіші одночасне натиснуті, – завершує виконання програми шляхом непрямого міжсегментного переходу до мітки **end_prot**: (це дозволяє в будь-який момент виконання програми, в тому числі і при непередбачених ситуаціях, наприклад, при зацикленні програми, терміново перервати її роботу);

– якщо встановлений чорно-білий режим виведення на екран (**modeVA**= 1), то по натисненню клавіші "Print Screen"викликає процедуру **save_ekran** (модуль **PROT**) для збереження поточного зображення екрану (див. підрозділ 1.15), інакше – здійснює занесення скан-коду в змінну **scan**.

Оскільки процедура **keyb** міститься в модулі **PROT**, після підключення цього модуля кожна програма має можливість:

– отримати в змінній **scan** значення скан-коду останньої натиснутої клавіші або відтиснутої клавіші (значення скан-коду відтиснутої клавіші відрізняється старшим бітом);

– по комбінації клавіш “лівий Shift” та “Pause/Break” терміново завершити виконання програми;

– по натисненню клавіші “Print Screen” зберегти поточне зображення екрану в змінній s**creen**.

У самій програмі **P_INT** розроблений оброблювач переривання від годинника реального часу RTC (Real Time Clock).

Годинник реального часу RTC в програмі **P_INT** застосовується для формування часових затримок (опис цього пристрою приведений в підрозділі 2.17). Оскільки пристрій RTC не був запрограмований програмами **BIOS** та **DOS** на генерацію переривань, то до початку обробки переривань, треба здійснити іні-

ціалізацію цього пристрою. Це здійснюється після переходу процесора до захищеного режиму (дивись підрозділ 1.9).

Оброблювач переривання від годинника реального часу реалізований в вигляді процедури **RTC**, виконує такі дії:

– скидає біти обслуговування переривання в контролерах PIC:

```
mov al,62h
out 20h,al          { в ведучому }
mov al,20h
out 20h,al
out 0a0h,al ;       { та в веденому }
```

– скидає RTC, шляхом читання його регістра стану з адресою Ch:

```
mov al,0ch
out 70h,al
in al,71h;
```

– по кожному перериванню з періодом в 1мс зменшує на одиницю (до нуля) значення змінних **time** та **pause**.

Таким чином змінні **time** та **pause** можуть бути використані в програмі **P_INT** для формування часових затримок з дискретністю в 1 мс (дивись підрозділ 1.13).

2.8 Формування таблиці GDT та завдання її параметрів

Програма **P_INT,** так же як і програма **P_MODE**, застосовує модуль **PROT** і тому вже має сформовану базову таблицю GDT (формування дескрипторів цієї таблиці дивись в підрозділі 1.4.3.3). Але на відміну від програми **P_MODE** використовує всі дескриптори цієї таблиці:

– дескриптор сегмента коду модуля **PROT**;

– дескриптор сегмента стека;

– дескриптор сегмента даних;

– дескриптор сегмента даних реального режиму;

– дескриптор сегмента відеопам’яті.

Крім того в програмі додатково формується дескриптор сегмента самої програми **P_INT**, який задає для сегмента коду у захищеному режимі ті ж самі параметри, який він мав в реальному режимі:

init_gdt(code_sel,$ffff,longint(Cseg) shl 4,acc_code,0);.

Якщо цей дескриптор не розробити та не завантажити його селектор в сегментний регістр CS зразу після переходу програми до захищеного режиму, то будь-яке переривання, яке виникне при роботі процесора в цьому режимі, приведе до зависання комп'ютера.

Формування дескриптора коду програми в модулі **PROT** в складі базової таблиці GDT неможливо, оскільки модулю недоступне значення селектора коду тієї програми, до якої він буде підключений.

Після формування таблиці GDT потрібно занести її базову адресу та розмір до регістра GDTR з тим, щоб мікропроцесор міг знайти цю таблицю в пам'яті та перевірити при доступі до неї, чи не виходить відносна адреса за межі таблиці. Це робиться за допомоги виклику процедури

init_gdtr(text_sel),

параметром якої є селектор останнього із застосованих в програмі сегментів пам'яті – селектора відеопам'яті. Більш детально це описано в підрозділі 1.5 **"**Завдання адреси та розміру GDT".

2.9 Формування дескрипторної таблиці переривань

Структура дескриптора таблиці IDT згідно з форматом дескриптора шлюзу (див. підрозділ 1.4) розроблена у вигляді запису наступного типу (модуль **PROT**):

```
t_idt=record      { Структура дескриптора IDT: }
ofs_l,            { зміщення оброблювача переривання (біти 15-0) }
sel:word;         { селектор оброблювача переривання }
par,              { число параметрів }
acc:byte;         { байт доступу }
ofs_h :word       { зміщення оброблювача переривання (біти 31-16) }
end;.
```

Формування таблиці IDT здійснюється за допомогою процедури модуля **PROTinit_idt(i:byte;ofs_:longint;sel_:word;acc_:byte)**, яка заносить значення

параметрів оброблювача переривання у відповідні поля заданого дескриптора. Параметри процедури **init_idt** мають такі значення:

i – номер переривання і, відповідно, номер дескриптора шлюзу оброблювача переривання в IDT;

ofs_, sel_ – зміщення та селектор оброблювача переривання;

acc_ – байт доступу.

У модулі **PROT** байти доступу шлюзів переривання та пастки для 16- та 32-розрядних значень задані в вигляді констант – відповідно **acc_int_16**, **acc_int**, **acc_trap_16** та **acc_trap**.

2.9.1 Формування вихідної таблиці IDT

Як вже повідомлялося (дивись підрозділи 1.7.3, 1.7.4 та 1.7.2), в модулі **PROT** містяться процедури-заглушки всіх обрроблювачів виключень в вигляді процедур **exc_00-exc_08, exc_10, exc_11, exc_13, exc_14 exc_16-exc_18** та задачі **exc12_task,** процедури-заглушки всіх зовнішніх апаратних переривань в вигляді процедур **PIC_1** та **PIC_2** а також процедури-оброблювачі програмних переривань **int30h** та **int32h**.

Крім того, в секції ініціалізації модуля **PROT** здійснені виклики процедури **init_idt** для формування вихідної таблиці IDT, яка містить дескриптори шлюзів всіх базових оброблювачів виключень, зовнішніх апаратних переривань та програмних переривань.

Для формування дескрипторів оброблювачів виключень 0-8, 10-14 та 16-18 використовуються виклики процедури **init_idt** з такими параметрами:

```
init_idt(0,ofs(exc_00), code_PROT,acc_trap);
.       .       .
init_idt(8,ofs(exc_08), code_PROT,acc_trap);
init_idt(10,ofs(exc_10), code_PROT,acc_trap);
init_idt(11,ofs(exc_11), code_PROT,acc_trap);
init_idt(12,0,exc12_sel,acc_task);
init_idt(13,ofs(exc_13), code_PROT,acc_trap);
init_idt(14,ofs(exc_14), code_PROT,acc_trap);
init_idt(16,ofs(exc_16), code_PROT,acc_trap);
        .       .       .
```

init_idt(18,ofs(exc_18), code_PROT,acc_trap);,

де:

– exc_00-exc_08, exc_10, exc_11, exc_13, exc_14 exc_16-exc_18 – процедури оброблювачів-заглушок виключень 0-8, 10, 11, 13, 14 та 16-18, що задані в модулі **PROT**;

– exc12_sel – селектор задачі EXC12_TASK;

– code_PROT – селектор сегменту коду модуля **PROT**;

– acc_trap – байт доступу шлюзу пастки для 32-розрядних значень;

– acc_task – байт доступу шлюзу задачі.

Для формування дескрипторів шлюзів оброблювачів-заглушок переривань від зовнішніх пристроїв, а також оброблювача переривання від клавіатури використовуються виклики процедури **init_idt** з такими параметрами:

```
init_idt($20,ofs(PIC_1),PROT_sel,acc_int);
init_idt($21,ofs(keyb),PROT_sel,acc_int);
for i:=2 to 7 do
init_idt($20+i,ofs(PIC_1),PROT_sel,acc_int),
```

де:

PIC_1 – процедура оброблювача-заглушки для входів IRQ0, IRQ2- IRQ7 першого контролера переривань;

keyb – процедура стандартного оброблювача переривання від клавіатури (див. підрозділ 2.7.4);

acc_int – байт доступу шлюзу переривання для 32-розрядних значень.

Для формування дескрипторів шлюзів оброблювачів-заглушок переривань від зовнішніх пристроїв для другого контролера переривань використовуються виклики процедури **init_idt** з такими параметрами:

```
for i:=8 to 15 do
init_idt($20+i,ofs(PIC_2),PROT_sel,acc_int),
```

де:

PIC_2 – процедура оброблювача-заглушки для входів IRQ8-IRQ15 – другого контролера переривань.

Для формування дескрипторів шлюзів оброблювачів програмного переривання використовуються виклики процедури **init_idt** з такими параметрами:

init_idt($30,ofs(int_30h),code_sel2,acc_trap_16);

init_idt($32,ofs(int_32h),code_sel2,acc_trap);.

Дескриптори всіх невикористаних номерів програмного преривання в таблиці IDT заповнюються нулями (важливо, щоб біт присутності (P) був скинутий):

for i:=$33 to 255 do init_idt(i,0,0,0);.

Формування вихідної дескрипторної таблиці переривань IDT здійснюється під час підключення модуля **PROT** до будь-якої програми. Таким чином забезпечується первний рівень надійності програм при виникненні при їх виконанні виключень та переривань.

Сама дескрипторна таблиця переривань IDT задана в вигляді змінної

idt:array[0..255] of t_idt; (модуль **PROT**).

2.9.2 Формування дескрипторів оброблювачів переривань в програмі P_INT

Кожна програма, що застосовує модуль **PROT**, може застосувати як стандартні оброблювачі переривань, що наведені в цьому модулі, так і оброблювачі переривань, що розроблені в самій програмі.

У цьому випадку треба також розробити дескрипторі шлюзів для цих процедур та записати їх в IDT замість вихідних. Це здійснюється, як і при формуванні вихідної таблиці IDT, за допомоги функції **init_idt**.

Наприклад, в програмі **P_INT** для оброблювачів виключень 5 та 11 формуються такі шлюзи, які змінюють шлюзи вихидних процедур **exc_05** та **exc_11** в таблиці IDT**:**

init_idt(5,ofs(my_exc_05),code_sel,acc_trap);

init_idt(11,ofs(my_exc_11),code_sel, acc_trap_16),

де:

– my_exc_05 та **my_exc_11 –** процедури оброблювачів виключень 5 та 11, які розроблені в програмі **P_INT** (див. підрозділ 2.7.4);

– code_sel – селектор коду програми **P_INT.**

Для формування дескриптора шлюза оброблювача переривань від годинника реального часу, який розроблений в програмі **P_INT** (див. підрозділ 2.7.4) використовуються виклик процедури **init_idt** з такими параметрами:

init_idt($28,ofs(RTC),code_sel,acc_int).

2.10 Завдання адреси і розміру IDT

Після формування дескрипторів шлюзів IDT необхідно вказати мікропроцесору місцеположення цієї таблиці в пам'яті та її розмір.

Для завдання адреси й розміру таблиці IDT застосовується регістр IDTR, який містить 32-розрядну базову адресу таблиці IDT та її 16-розрядну границю.

Як це видно з рис. 2.6 регістр IDTR має ту ж саму структуру, що і регістр GDTR (див. підрозділ 1.3). Тому і вміст регістра IDTR в реальному та захищеному режимах має такий же формат (тип даних), що і вміст регістра GDTR.

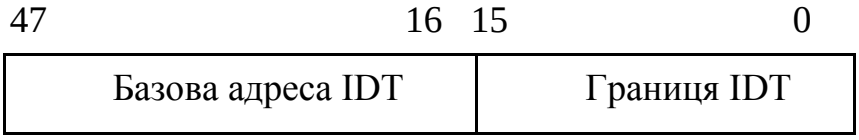

Рисунок 2.6 – Структура регістра IDTR

На відміну від регістра GDTR регістр IDTR застосовується як в реальному, так і у захищеному режимах. У реальному режимі він зберігає базову адресу та розмір області векторів оброблювачів переривань реального режиму, а у захищеному – адресу та розмір дескрипторної таблиці переривань IDT. Тому вміст регістра IDTR описується відповідно змінними **idtr_r** та **idtr** типу **t_dtr**.

У програмі **P_INT** для зберігання поточних даних регістра IDTR (даних реального режиму) та формування даних цього регістра для роботи у захищеному режимі та його завантаження цими даними застосовується процедура **init_idtr** (модуль **PROT**), яка за допомоги команди SIDT заносить до змінної **idtr_r** поточні значення регістра IDTR (атрибути таблиці IDT реального режиму), формує в змінній **idtr** дані захищеного режиму та завантажує за допомоги команди LIDT ці дані до регістра IDTR.

Команди SIDT та LIDT, як і команда LGDT (див. підрозділ 1.3) реалізовані в вигляді машинних команд:

```
db 0fh,1,0eh      { команда SIDT idtr_r: }
dw idtr_r         { Зберігання атрибутів IDT в idtr_r }
db 0fh,1,1eh      { команда LIDT idtr: }
```

dw idtr *{ Завантаження атрибутів IDT в IDTR з idtr }.*

Оскільки після занесення даних в регістр IDTR встановлюється порядок оброблення переривань захищеного режиму, то перед цим в процедурі **init_idtr** забороняються всі зовнішні апаратні переривання: (дивись підрозділ 1.7).

cli *{ ті, що маскуються }*
mov al,80h *{ й ті, що }*
out 70h,al *{ не маскуються }.*

2.11 Перепрограмування контролерів переривань

Оскільки фірма Intel зарезервувала 32 молодших номери переривань для виключень, то для роботи у захищеному режимі потрібно змінити номери переривань від зовнішніх пристроїв, які попадають в діапазон 0-31.

Наприклад, номер переривання від системного таймера, який дорівнює 8 в реальному режимі, повинен бути змінений, так як він збігається з номером виключення "Подвійна помилка".

Номери переривань від зовнішніх пристроїв в комп'ютері задаються за допомоги двох контролерів PIC. Перший контролер PIC запрограмований BIOS для роботи в реальному режимі на номери переривань від 8 до 15, другий PIC – на номери від 75h до 7Fh. Хоч номери переривань 2-го контролера переривань не попадають в діапазон 0-31, в програмі **P_INT** він також програмується з метою упорядкування номерів зовнішніх переривань.

Перепрограмування першого та другого контролерів переривань здійснюється за допомоги процедури модуля **PROT**

pic(mode:byte),

яка в залежності від значення параметра **mode**, що вказує режим роботи мікропроцесора (**mode** = 0: реальний; **mode** = 1: захищений), задає такі номери переривань:

а) для першого PIC, який є ведучим:

– 8 (**mode** = 0) ;

– 20h (**mode** = 1);

б) для другого PIC, який є веденим:

– 75h (**mode** = 0);

– 28h (**mode** = 1).

Номери переривань в PIC задаються тільки для входу IRQ0 (IRQ8). Для

входу IRQ1 номер переривання становить на 1 більше, для входу IRQ2 – на 2 більше і т.д.

Ініціалізація контролера переривань, тобто завдання його режиму роботи та параметрів, в тому числі і номерів переривань для IRQ0, здійснюється за допомоги чотирьох управляючих слів ініціалізації ICW (Initialization Control Word): ICW1-ICW4.

Хоча завдання номера переривання робиться за допомоги одного слова, а саме ICW2, проте організація PIC вимагає, щоб всі управляючі слова ініціалізації видавалися обов'язково разом (ICW1-ICW4). Тому в процедурі **pic** (модуль **PROT**) для програмування першого та другого контролерів переривань (PIC1 і PIC2) застосовані всі чотири управляючі слова, значення та призначення яких наведені в таблиці 2.4:

Таблиця 2.4 – Значення управляючих слів ініціалізації

ICW	Значення ICW				Призначення ICW
	Реальний режим		Захищений режим		
	PIC1	PIC2	PIC1	PIC2	
ICW1	11h	11h	11h	11h	Задає необхідність генерації CW4
ICW2	8	20h	70h	28h	Задає номери переривань для входів IRQ0 та IRQ8
ICW3	4	2	4	2	Задає конфігурування: PIC2 підключається к PIC1через вхід IRQ2
ICW4	1	1	1	1	Задає тип МП: починаючі з 8086 та режим обробки переривань: обов'язкове скидання PIC

У програмі **P_INT** для програмування першого та другого ПКП для роботи у захищеному режимі здійснюється виклик процедури **pic(1)**.

2.12 Перехід до захищеного режиму

Перед переходом до захищеномго режиму треба зберегти в пам'яті:

1) адреси міток **end_prot** та **ret_real** для здійснення непрямих міжсегментних переходів:

– до мітки **end_prot** процедурами-заглушками оброблювачів виключень модуля **PROT** (дивись підрозділ 2.7.3);

– до мітки **ret_real** при повертанні до реального режиму (дивись підрозділ 2.7.14).

При цьму в змінні **ofs_end_prot**, **ofs_ret_real**, **sel_end_prot** та **sel_ret_real** заноситься відповідно зміщення міток **end_prot** та **ret_real** та їх селектори:

```
asm
        mov ofs_end_prot,offset end_prot
        mov sel_end_prot,code_sel
        mov ofs_ret_real,offset ret_real
        mov sel_ret_real,cs
end.
```

2) вміст регістрів DS, SS, ES та покажчика стека SP – оскільки значення сегментних регістрів в реальному та захищеному режимах суттєво відрізняються – в реальному режимі в них зберігаються базові адреси сегментів, а у захищеному – селектори сегментів (дивись підрозділ 1.8 “Зберігання в пам’яті регістрів мікропроцесора”).

Перехід до захищеного режиму в програмі **P_INT** має дві відмінності по відношенню до цих дій в програмі **P_MODE**:

а) не розглядаються особливості переходу для 16-розрядних процесорів;

б) замість переходу на мітку **@prot:** в межах сегменту по команді **jmp @prot** використовується команда міжсегментного переходу на ту ж саму мітку:

```
        db 0eah
        dw offset @prot
        dw code_sel.
```

Зверніть увагу, що при об’явленні констант значення селектора сегмента коду задано в вигляді константи: **code_sel = 18h**, а не в вигляді типізованої константи, як для інших селекторів, наприклад, селектора сегмента даних: **data_sel :word = $30**. Це зроблено тому, що транслятор з мови Паскаль по різному інтерпретує імена констант, типізованих констант та змінних в командах асемблера та в машинних командах, які реалізовані за допомоги операторів DB та DW:

– в командах асемблера замість імен констант, типізованих констант та змінних підставляються їх значення;

– в реалізаціях машинних команд замість імен констант також підставляються їх значення, але замість імен типізованих констант та змінних підставля-

ються їх адреси (зміщення сегмента даних)**.**

При іншому варіанті реалізації міжсегментного переходу, де нема машинних команд, наприклад, такому:

push code_sel
push offset @prot
retf

значення селектора сегмента коду можна було би задати в вигляді типізованої константи, як і для інших селекторів.

Міжсегментний перехід в програмі **P_INT** застосований з метою завантаження в сегментний регістр коду CS селектора захищеного режиму із значенням 18h, бо інакше при обробленні будь-якого переривання поточне значення регістру CS, тобто його значення в реальному режимі – 1C90h, буде занесене в стек, а потім по команді IRET оброблювача переривання завантажено в сегментний регістр як селектор, при цьому виникне виключення 13 із-за перевершення границі таблиці GDT.

Після завантаження інших сегментних регістрів (DS, SS та ES) відповідними селекторами (**data_sel**, **stack_sel** та **text_sel**) розпочинається робота програми **P_INT** у захищеному режимі.

2.13 Робота програми P_INT у захищеному режимі

Після переходу програми **P_INT** до захищеного режиму виконуються такі дії:

1. Здійснюється ініціалізація годинника реального часу RTC (Real Time Clock), яка передбачає установлення дозволу на генерацію цим пристроєм переривань та його скидання. Це дозволяє обробляти зовнішні переривання як від клавіатури, так і від RTC, оскільки при роботі під операційною системою DOS цей пристрій був запрограмований тільки для читання поточної дати та часу.

Для програмування RTC на генерацію переривань з періодом 1мс (дивись підрозділ 2.17) потрібно в його регістр стану за адресою Bh записати код 4ah:

mov al,0bh
out 70h,al
mov al,4ah
out 71h,al,

що означає (див. рис. 2.13):

– 24-годинний режим роботи;

– дозвіл прямокутних імпульсів та дозвіл періодичної генерації переривань з періодом 1мс.

Крім цього для стабільної роботи пристрою треба здійснити начальне скидання RTC, шляхом читання з його регістра стану за адресою Ch:

mov al,0ch

out 70h,al

in al,71h.

2. На екрані монітора до інформації, що була виведена в реальному режимі, додаються такі повідомлення та дані захищеного режиму (див. рисунок 2.7):

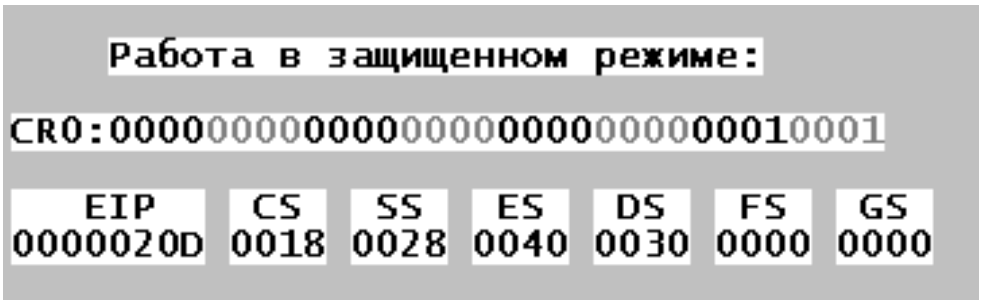

Рисунок 2.7 – Дані захищеного режиму

– за допомоги функцій 1 та 4 переривання 30h – повідомлення "**Работа в защищенном режиме:**"

```
mov ax,103h
mov bx,050Ah                    { Встановлення маркера (5,10) }
int 30h
mov ah,4
mov cl,9eh      { Заданий16-розрядний фон або  режим мерехтіння }
mov si,offset s_prot
int 30h;                        { Виведення рядка s_prot на екран }
```

– за допомоги функції 4 переривання 32h – значення бітів регістру управ-

ління CR0:

mov bx,10Ch
mov ax,400h
int 32h;

– за допомоги функції 3 переривання 32h – вміст регістрів EIP, CS, SS, ES, DS, FS та GS:

mov bx,10Eh
mov ax,300h
int 32h.

Визначимо, що в порівнянні з реальним режимом сегментні регістри містять не базові адреси сегментів пам'яті, а селектори дескрипторів цих сегментів, значення яких співпадають з тими, що були задані в модулі **PROT** (регістри FS та GS мають значення 0 тому, що не були завантажені при роботі програми **P_INT** та не застосовувалися при роботі під DOS).

Крім цього біт PE регістра CR0 має значення 1, що свідчить про роботу процесора у захищеному режимі.

3. Здійснюється завантаження регістру TR селектором задачі **main** для обробки виключення 12:

db 0fh,0,1eh *{ LTR main_sel: }*
dw v_main_sel.

4. Дозволяється оброблення зовнішніх апаратних маскованих переривань за допомоги команди STI. Якщо цього не зробити, то очікувані в програмі **P_INT** запити на переривання від пристрою RTC та клавіатури не будуть оброблені.

5. За допомоги переривання від годинника реального часу реалізується затримка впродовж трьох секунд. Для цього:

а) на екран виводиться повідомлення "**Выполняется трехсекундная задержка**" (рис. 2.8);

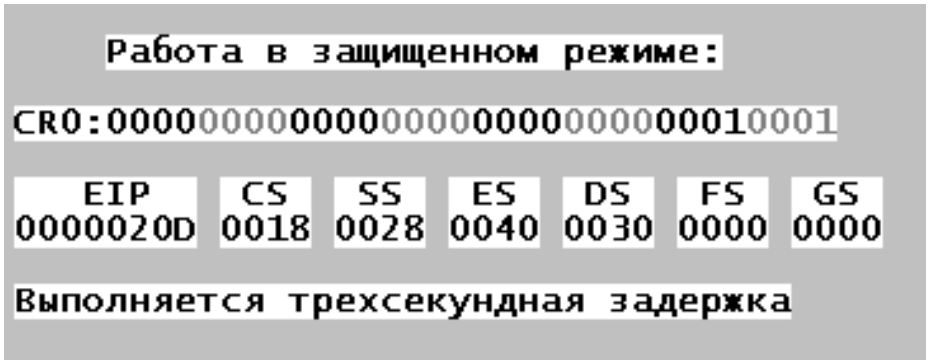

Рисунок 2.8 – Зображення екрану при виконанні затримки

б) виконується трьохсекундна затримка шляхом занесення в змінну **time** значення часу затримки в мс, тобто 3000,

```
db 66h,0c7h,6                    { mov (double) word ptr time,3000 }
dw time
dd 3000
```

та очікування, поки значення цієї змінної не стане дорівнювати 0:

```
@d1:db 66h,81h,3eh               { cmp (double) word ptr time,0 }
    dw time
    dd 0
    jnz @d1;
```

в) з екрану за допомоги функції 1 переривання 30h витирається повідомлення про трьохсекундну затримку.

Застосування при занесенні значення до змінної **time** та порівнянні поточного значення **time** з 0 машинних кодів команд **mov** та **cmp** пов'язано з тим, що змінна **time** має 32-розрядне значення (об'явлена з типом **longint**), а Pascal в асемблерних вставках не підтримує 32-розрядні операції. Якщо об'явити цю змінну з типом **word**, то це обмежить величину затримки до 65 секунд.

6. Задається 16-розрядний колір символів замість мерехтіння (діє тільки при завданні кольорового режиму роботи).

7. З метою перевірки коректності роботи розроблених процедур **exc00**, **exc05** та **exc11 з**дійснюється генерація виключень 0, 5 та 11. Дозвіл чи заборона на генерацію кожного окремого виключення робиться за допомоги типізованих констант **ena_exc00**, **ena_exc05** та **ena_exc11**, які при ініціалізації мають зна-

чення, що забороняють генерацію виключень:

ena_exc00:byte=0; *{ виключення 0 }*
ena_exc05:byte=0; *{ виключення 5 }*
ena_exc11:byte=0; *{ виключення 11 }.*

Щоб дозволити генерацію того чи іншого виключення, потрібно в відповідну типізовану константу занести 1, наприклад, щоб дозволити вироблення виключення 0, потрібно змінити вихідне значення типізованої константи **ena_exc00** на таке:

ena_exc00:byte=1;

а) Генерація виключення 0. Виключення 0 виникає при діленні на нуль:

```
mov ax,1
mov bl,0
div bl.
```

Другим варіантом виникнення виключення 0 є така послідовність команд:

```
mov ax,100h
mov bl,1
div bl.
```

У цьому випадку виключення 0 виникає тому, що кількість розрядів результату (9) перевищує розрядність регістру призначення (AL – 8 розрядів).

При виникненні виключення 0 оброблювач цього переривання процедура **exc00** виводить на екран таке повідомлення:

```
Runtime error: исключение 00 (0018:00000253)
```

Рисунок 2.9 – Повідомлення оброблювача виключення 0

З рисунку 2.9 видно, що помилка ділення на нуль виникла в основній програмі (**P_INT),** тому що селектор сегменту дорівнює 18h, при виконанні команди, яка розташована через 70 байтів після команди, що вивела на екран значення EIP – 20Dh.

б) Генерація виключення 5. Виключення 5 виникає при роботі команди

BOUND, яка реалізована в вигляді машинної команди і генерує переривання 5 при таких заданих параметрах – діапазон значень: 10-20, значення, що перевіряється: 7:

mov lim1,10
mov lim2,20
mov ax,7
db 62h,6 *{ команда BOUND }*
dw lim1.

При виникненні виключення 5 оброблювач цього переривання процедура **exc05** виводить на екран дані, що наведені на рисунку 2.10.

Рисунок 2.10 – Повідомлення оброблювача виключення 5

З рисунку 2.10 видно, що значення регістра EAX дійсно дорівнює 7. Якщо задати значення регістра AX в межах діапазону, наприклад,

mov ax,15,

то виключення 5 не виникне.

в) Генерація виключення 11. Виключення 11 виникає після скидання біту P в байті доступу дескриптора сегмента даних (імітація відсутності сегмента в пам'яті):

mov ax,data_sel *{ Визначення згідно з селектором }*
shr ax,3 *{сегмента даних }*
mov dl,8 *{ зміщення його дескриптора в таблиці GDT }*
mul dl *{з занесенням в дозволений }*
mov bx,ax *{ для завдання зміщення регістр BX }*
and byte ptr [offset gdt+bx+5],07fh, *{ скидання біта 7 в байті 5 }*
{ дескриптора сегмента даних }

та завантаження сегментного регістра даних селектором цього модифікованого

дескриптора:

mov ds,data_sel.

При виникненні виключення 11 оброблювач цього переривання процедура **exc11** виводить на екран таке повідомлення:

Рисунок 2.11 – Повідомлення оброблювача виключення 11

З рисунку 2.11 видно, що код помилки вказує на сегмент пам'яті з селектором 30h в таблиці GDT, тобто на сегмент даних програми **P_INT**.

8. Здійснюється визначення та виведення на екран значень скан-кодів клавіш клавіатури. При цьому розглядалися два варіанти вирішення цієї задачі.

Перший варіант передбачав використання для цієї мети оброблювач переривання від клавіатури, що повинен бути розроблений у програмі **P_INT**.

У другому варіанті передбачалося використати для визначення скан-коду стандартний оброблювач переривань від клавіатури – процедуру **keyb** модуля **PROT**, а додаткові функції по виведенню значень скан-кодів на екран покласти на дії програми **P_INT**.

Для реалізації був вибраний другий варіант, так як у першому варіанті новий оброблювач переривань від клавіатури багато в чому дублював функції процедури **keyb** і був тому явно надлишковим.

Тому в програмі **P_INT** при застосуванні стандартного оброблювача переривань виконуються такі дії:

а) за допомоги функції 1 переривання 30h встановлюється маркер (1,14) з очищенням екрану**;**

б) за допомоги функції 4 переривання 30h на екран виводиться повідомлення:«**Скан-коды нажатия/отжатия клавиши:»**;

в) очікується натиснення чи відтиснення клавіші клавіатури шляхом порівняння скан-коду попередньої клавіші (змінна **old_scan)** з скан-кодом тільки що натиснутої або віджатої клавіші (змінна **scan)**, значення якої змінює тільки оброблювач переривань від клавіатури – процедура **keyb** модуля **PROT** (дивись підрозділ 2.7.4):

@wait:
mov al,scan
cmp al,old_scan
jz @wait;

г) коли значення змінної **scan** зміниться (була натиснута чи відтиснута клавіша), цикл очікування закінчується і змінна **old_scan** знову приймає значення змінної **scan.**

Більшість клавіш клавіатури, у тому числі багато управляючих клавіш, мають тільки два скан-коди: скан-код натиснення та скан-код відтиснення, які відрізняється старшим бітом (в скан-кодах натиснення клавіш він дорівнює 0, в скан-кодах відтиснення – 1).

Наприклад, на рисунку 2.12 наведені скан-коди,клавіші “лівий Shift”.

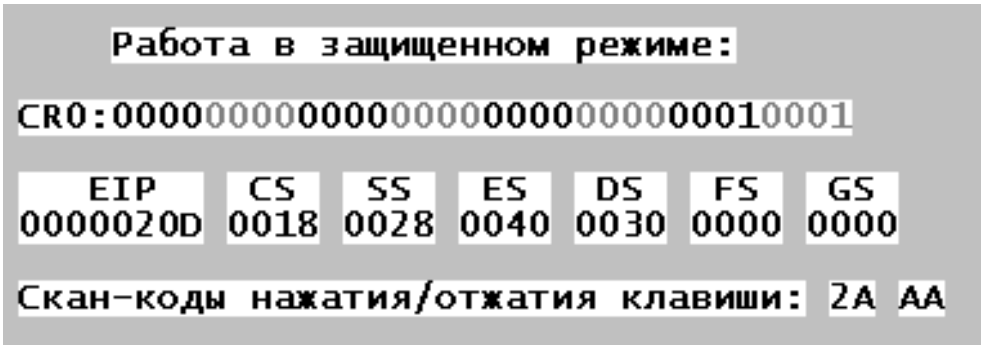

Рисунок 2.12 – Скан-коди натиснення та відтиснення клавіші “лівий Shift”

Але деякі клавіші мають більшу кількість скан-кодів, наприклад, клавіша “Pause/Break” – шість (дивись рис. 2.13),

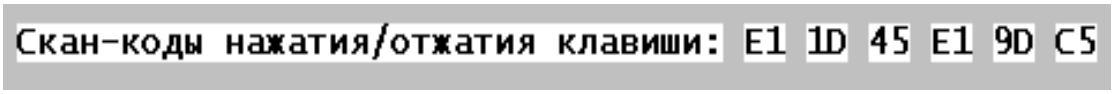

Рисунок 2.13 Скан-коди клавіші “Pause/Break”

а клавіша “Print Screen” навіть вісім: E0 2A E0 37 E0 B7 E0 AA.

Тому для виведення в рядку на екрані всіх скан-кодів але тільки однієї клавіші було запропонований такий підхід. На виведення всіх скан-кодів однієї

клавіші відводиться певний час, і якщо цей час буде вичерпаний, то зміна скан-коду означає що це вже є скан-код іншої клавіші. Скан-коди однієї клавіші генеруються контролером клавіатури впродовж декількох десятків мс. Тому час, призначений для генерації скан-кодів був експериментально вибраний рівним 200 мс (треба щоб з іншої сторони цей час був менше часу між натисканням різних клавіш).

Для завдання часу, потрібного для генерації скан-кодів була використана змінна **pause**, значення якої спочатку дорівнює 200, а потів завдяки оброблювачу переривань від годинника реального часу (див. підрозділ 2.7.4) зменшується на 1 кожну 0,001 секунди.

г) Після розпізнання нового скан-коду значення змінної **pause** порівнюється з 0. Якщо її значення не дорівнює 0, то це означає, що час генерації скан-кодів даної клавіші ще не закінчився і тому потрібно виводити поточний скан-код клавіші на екран.

Якщо значення змінної **pause** дорівнює 0, то це означає, що час генерації скан-кодів клавіші вичерпаний, всі її скан-коди вже виведені на екран. Тому перед виведенням чергового скан-коду на екран потрібно витерти з екрану всі виведені раніше скан-коди та занести в змінну **pause** знову вихідне значення 200 для нового відліку часу на генерування скан-кодів нових клавіш.

д) Виведення скан-кодів клавіші на екран здійснюється за допомоги функції 3 переривання 30h:

```
mov ax,300h
mov cl,1bh
mov dl,scan
int 30h;
```

е) Очищення частини рядку для виведення нових скан-кодів здійснюється за допомоги підфункції 1 функції 1 переривання 30h:

```
mov ax,101h
mov bx,2410h
int 30h;
```

ж) Аналіз скан-кодів та виведення їх значень на екран буде продовжуватися поки не буде натиснута клавіша “Esc” (скан-код дорівнює 1). Якщо ця клавіша натиснута, то робота програми **P_INT** у захищеному режимі завершується.

2.14 Повертання процесора до реального режиму

Для переходу до реального режиму в програмі **P_INT** виконуються такі дії:

1) забороняються зовнішні масковані переривання:

cli.

2) відновлюються атрибути таблиці IDT (таблиці векторів) для роботи в реальному режимі:

```
db 0fh,1,1eh                    { LIDT idtr_r }
dw idtr_r.
```

3) завантажуються сегментні регістри параметрами реального режиму:

```
mov ss,real_sel
mov ds,real_sel
mov es,real_sel.
```

4) скидається біт PE регістра CR0 (дивиться підрозділ 1.13);

5) здійснюється перехід на мітку **real**, де починається робота МП в реальному режимі, шляхом міжсегментного переходу для завантаження сегментного регістра CS значенням реального режиму (**real_cs**):

```
push real_cs
push offset real
retf
```

6) після повернення в реальний режим відновлюється вміст сегментних регістрів DS, SS, ES та покажчику стека SP значеннями, які вони мали до переходу до захищеного режиму (дивиться підрозділ 1.13);

7) оскільки на час роботи процесора у захищеному режиму всі апаратні переривання (масковані та немасковані) були заборонені, після повернення в реальний режим вони одержують дозвіл (дивиться підрозділ 1.13);

8) програмується контролер переривань для роботи в реальному режимі:

pic(0).

9) здійснюється скидання стану клавіш-перемикачів:

mem[$40:$17]:=0.

2.15 Зберігання зображення екрану у захищеному режимі

При роботі у захищеному режимі програми **P_MODE**, **P_INT** та ін. здійснюють виведення на екран як повідомлень, так і чисельних даних. Щоб можна було би включити ці зображення в матеріали навчального посібника, або в інші документи, що готуються в середовищі ОС Windows, необхідно зберегти ці зображення на магнітному диску в вигляді файлу одного з графічних форматів.

Якщо програма **P_INT** працює в режимі віртуалізації в середовищі сучасної ОС, то проблеми збереження зображення не існує. Але якщо програма **P_INT** самостійно перейшла до захищеного режиму, то ніякі програмні засоби BIOS, DOS, не можна використати для збереження зображень.

Для вирішення цієї задачі запропоновані такі дії:

1. У захищеному режимі зберегти зображення на екрані в масиві даних оперативної пам'яті.

Пропонується це зробити так же, як і в середовищі Windows – по натисненню клавіші "Print Screen". При цьому (див. підрозділ 5.13) генерується вісім скан-кодів: E0 2A E0 37 E0 B7 E0 AA.

Зберігати зображення екрану по першому скан-коду (E0) не можна, так як декілька клавіш, такі, як "правий Alt", "правий Ctrl", "Insert", "Home", "Delete" та ін. теж спочатку генерують E0.

Робити зберігання екрану по послідовності перших двох скан-кодів (E0 2A), яка є унікальна для клавіш, не досить коректно для програми **P_INT**, так як в цій програмі на екран виводиться кожний скан-код натиснутої та відтиснутої клавіші. Тому при кожному натисненні клавіші "Print Screen" в ті дані екрану, що будуть збережені в масиві, попаде зображення першого скан-коду, тобто значення "E0".

Тому було прийнято рішення виконати такі дії:

а) якщо два перших скан-кода клавіші є кодами E0h і 2Ah, тобто натиснута клавіша "Print Screen", то встановити ознаку збереження екрана (**save_on = 1**) й змінити зміщення масиву **screen** для збереження другого екрану (цей масив має розмір 8000 байтів);

б) за допомогою функції 7 переривання 30h зберегти поточне зображення

екрана в масиві **screen**:

```
mov ax,700h          { зберегти поточне значення області екрану }
mov bx,0                                  { з координатами:  (0,0) та }
mov dx,4F18h                        { (79,24), тобто всього екрану }
int 30h;                                    {  в масиві screen }
```

в) якщо скан-код натиснутої клавіші дорівнює E0h і скинута ознака збереження екрана, – за допомогою функції 7 INT 30h зберегти поточний вміст області екрана.

Дії по збереженню зображення на екрані реалізовані в вигляді процедури **save_ekran** (модуль **PROT**), яку викликає оброблювач переривання від клавіатури – процедура **keyb** (модуль **PROT**) при натисненні на клавішу "Print Screen".

Треба додати, що виклик процедури **save_ekran** здійснюється тільки в режимі чорно-білого зображення екрану (**modeVA** = 1).

2. Переписати дані з масиву оперативної пам’яті до магнітного диску.

Після переходу до реального режиму викликається процедура **save_scr** (модуль **PROT**), яка виконує такі дії:

– перевіряє ознаку **save_on,** і якщо вона встановлена, то на екран виводиться повідомлення: “**Введите имя массива для сохранения экрана**”;

– приймає від користувача програми ім’я файлу, створює файл з таким ім’ям в каталозі EKRAN диску C:\ та переписує дані з масиву **screen** до цього файлу.

3. У середовищі Windows дані з файлу на диску вивести на екран.

За допомоги програми **ekranw.pas,** що працює під Windows, збережені дані зображення екрану з файлу каталогу EKRAN диску C:\ виводяться на екран вже в графічному вигляді.

4. По натисненню клавіші “Print Screen” зображення на екрані зберегти в буфері обміну.

5. У графічному редакторі, наприклад Paint, зображення екрану взяти з буферу та зберегти на магнітному диску в графічному форматі у формату 16-кольорового BMP.

6. Після цього зображення екрану захищеного режиму можна вставити в будь-який текстовий матеріал.

Усі зображення екрану, які наведені в главах 4, 5 та ін. зроблені за допомоги дій, що наведені в пунктах 1-6.

2.16 Налагоджування програм у захищеному режимі

Розробка програм для роботи у захищеному режимі сама по собі є нелегкою справою. Становище усугубляється ще й тим, що в цьому режимі відсутні системні засоби налагоджування програм.

Тому для полегшення задачі розробки й налагоджування програм у захищеному режимі можна порекомендувати такі програмні засоби модуля **PROT**, які дозволяють здійснити:

– оброблення всіх без винятку виключень захищеного режиму з виведенням на екран номера виключення, повної адреси (селектор та зміщення) команди, що викликала виключення (для відмов) чи наступної (для пасток), та коректне завершення програми;

– виведення на екран текстових повідомлень й числових значень 8-, 16- та 32-розрядних регістрів та змінних в двійковій, 10-річній та 16-річній формі (функції 3, 4, 5, та 6 оброблювача INT 30h);

– виведення на екран найменувань та вмісту регістрів загального призначення: EAX, ECX, EDX, EBX, ESP, EBP, ESI та EDI (функція 2 оброблювача INT 32h);

– виведення на екран вмісту лічильника команд EIP та сегментних регістрів CS, SS, ES, DS, FS та GS (функція 3 оброблювача INT 32h);

– виведення на екран найменувань та значень бітів регістру CR0 (функція 4 оброблювача INT 32h);

– виведення на екран найменувань та значень бітів регістру CR4 (функція 5 оброблювача INT 32h);

– виведення на екран найменувань та значень бітів регістру прапорів EFLAGS (функція 6 оброблювача INT 32h).

Як приклад, приведемо налагоджувальну послідовність команд, яку можна поставити в будь-якому місці програми й яка дозволяє визначити поточне значення регістра або змінної шляхом виведення його в лівий верхній кут екрану. Це значення зберігається на екрані до натиснення будь-якої клавіші.

При використанні налагоджувальної послідовності команд треба:

а) шукане значення регістра чи змінної занести до стека (якщо це 32-

розрядне значення – застосувати перед командою **push** префікс розрядності (**db 66h**);

б) в залежності від розрядності значення використати відповідну під функцію функції 3 переривання 30h:

– 0 – для 8-розрядних;

– 1 – для 16-розрядних;

– 2 – для 32-розрядних;

в) якщо потрібно виводити значення не в 16-ріічній формі (як в прикладі), а в двійковій чи десятковій, треба замість функції 3 переривання 30h застосувати функції 5 або 6 цього переривання;

г) для 32-розрядного значення застосувати перед командою **pop** префікс **db 66h**.

Щоб налагоджувальні команди не змінили значення регістрів та не зіпсували програму, передбачено збереження 8 регістрів загального призначення в стеку з відновленням їх значень.

Виведене на екрані значення зберігається до натиснення будь-якої клавіші. При відтисненні клавіші затримка продовжується, щоб налагоджувальну послідовність можна було одночасно застосувати в декількох точках програми.

Щоб забезпечити на час затримки роботу тільки переривання від клавіатури, здійснюються такі дії:

– поточне значення регістру FLAGS заноситься до стека

– маскуються всі зовнішні переривання окрім переривання від клавіатури;

– встановлюється прапор IF регістра EFLAGS.

Після виконання затримки дозволяються всі зовнішні переривання та відновлюється вихідний стан регістру FLAGS.

```
{---------------------- налагоджувальна послідовність команд----------------------}
        pusha
        mov al,scan          { Значення,регістра чи змінної, яке треба вивести }
        push ax                                  { на екран, заноситься до стека }
        mov ax,100h
        mov bx,0
        int 30h                      { Встановлення маркеру у початок екрану }
        mov ax,300h    { Завдання розрядності для значення, що виводиться: }
```

```
        mov cl,1dh                  { 8 – 300h, 16 – 301h, 32 – 302h }
        pop dx
        int 30h                     { Виведення значення на екран }
        mov al,0dh                  { Заборона усіх зовнішніх переривань }
        out 21h,al                  { крім клавіатури }
        mov al,0ffh
        out 0a1h,al
        pushf
        sti
        mov scan,0
@ww:cmp scan,0                      { Затримка до натиснення будь-якої клавіші }
        jz @ww
        test scan,80h
        jnz @ww
        popf
        mov al,0                    { Дозволити всі зовнішні переривання }
        out 21h,al
        out 0a1h,al
        popa.
```

2.17 Годинник реального часу

Комп’ютери AT оснащені годинником реального часу. Цей пристрій живиться від акумулятора, тому його значення зберігаються при вимкненні комп’ютера.

Доступ до годинника реального часу можливий або через комірки КМОП-пам’яті, або через спеціальні функції **BIOS**. Використання комірок КМОП-пам’яті годинником реального часу наведене в табл. 2.5. На рис. 2.14 наведений формат регістра стану А годинника реального часу.

Таблиця 2.5 – Призначення комірок CMOS - пам’яті

Комірка	Призначення
0	Лічильник секунд
1	Регістр секунд пробуджувала
2	Лічильник хвилин
3	Регістр хвилин пробуджувала
4	Лічильник годин
5	Регістр годин пробуджувала
6	Лічильник днів тижня (1 – неділя)
7	Регістр днів місяця
8	Лічильник місяців
9	Лічильник років (останні дві цифри поточного року)

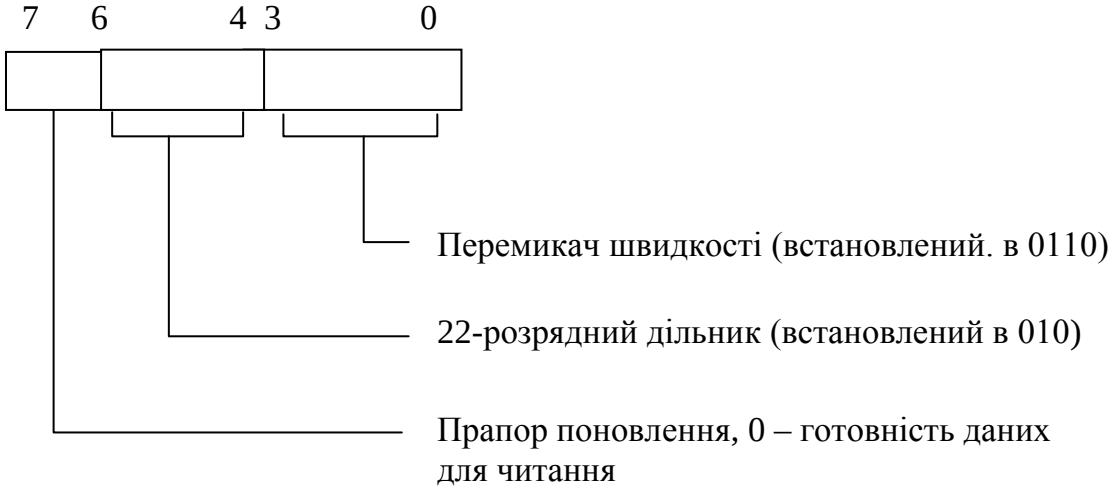

Рисунок 2.14 – Регістр стану А

Годинник реального часу виробляє апаратне переривання IRQ8, якому в реальному режимі відповідає переривання з номером 70h.

Це переривання може вироблятися в трьох випадках (на рис. 2.15 приведений формат регістра стану В годинника реального часу):

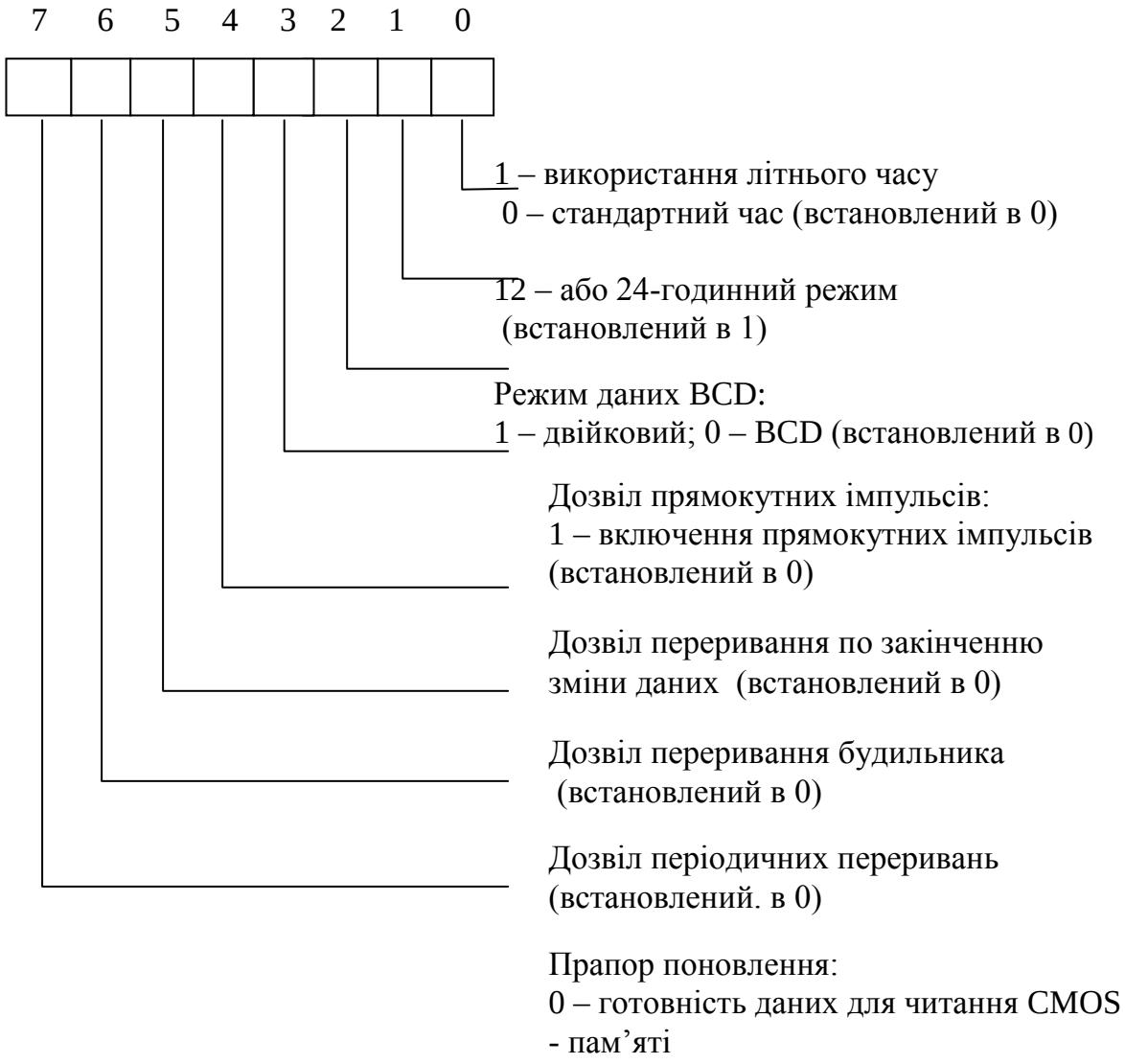

Рисунок 2.15 – Регістр стану B

1) переривання по закінченню зміни даних (виробляється при встановленому в “1” 4-го біту регістра стану B після кожного поновлення регістрів годин);

2) переривання будильника (виробляється при збігу регістрів годин і регістрів будильника й при встановленому в “1” 5-го біту регістра станів B);

3) періодичне переривання (виробляється з періодом приблизно 1 мілісекунду при встановленому в “1” 6-го біту регістра стану B).

Регістр стану C годинника реального часу містить біти стану переривання, його можна тільки читати.

Регістр стану D годинника реального часу: якщо біт 7 дорівнює 0, це означає, що розрядився акумулятор, що живить CMOS-пам’ять.

Для роботи з годинником реального часу можна звертатися безпосередньо до указаних вище комірок CMOS - пам’яті, використовуючи для цього пор-

ти 70h та 71h (для цього в порт 70h необхідно занести адресу комірки, а через порт 71h здійснити запис чи читання даних).

2.18 Текст програми P_INT

```
{==========Обработка прерываний в защищенном режиме==========}
program p_int;
{--------------Модуль PROT предназначен для поддержки програм,--------------}
{-----------------------работающих в защищенном режиме:----------------------}
{------------а) содержит необходимые константы, типы переменных,--------}
{------------переменные, процедуры и функции;-----------------------------------}
{------------б) создает базовые таблицы GDT и IDT-------------------------------}

uses prot;
label
        end_prot,       { Метки окончания работы в защищенном режиме }
        real;                                   { и возврата в реальный режим }
const
{------------Строковые константы для вывода сообщений на экран------------}
        s_real:string=' Работа в реальном режиме:';
        s_prot:string=' Работа в защищенном режиме:';
        s_scan:string=' Скан-коды нажатия/отжания клавиши: ';
        s_err:string=' Исключение: 11 / код ошибки: ';
        s_delay:string='Выполняется трехсекундная задержка';
{--------------Разрешение или запрет (1/0) выработки исключений:--------------}
        ena_exc00:byte=0;                                       { исключения 0 }
        ena_exc05:byte=0;                                       { исключения 5 }
        ena_exc11:byte=0;                                       { исключения 11}
var
        bit32:byte;                     { Признак разрядности обработчика }
        pause,                  { Значение паузы при обработке скан-кодов }
        old_scan:byte;                                  { Предыдущий скан-
    код }
        lim1,lim2:word;         { Нижняя и верхняя границы диапазона }
        time:longint;                                   { Задержка в мс }
{=============Разработка обработчиков исключений=============}

{------------------Обработчики исключений могут использовать-----------------}
{------------------один из трех вариантов обработки отказа:-----------------}
{--------------1: досрочно завершить выполнение программы;-------------------}
{--------------2: обойти команду, вызвавшую отказ;-----------------------------}
{--------------3: устранить причину отказа.-------------------------------------}
{-------Вариант 1 используют обработчики-заглушки всех исключений-------}
{--------------(модуль PROT), в том числе и исключения 0 (exc_00).--------------}
```

```
{-------Вариант 2 использует обработчик исключения 5 (my_exc_05).---------}
{--------Вариант 3 использует обработчик исключения 11 (my_exc_11).------}

{-----------------------------Обработчик исключения 5-------------------------------}
{------------------Изменяет в стеке значение счетчика команд--------------------}
{-----------------на величину длины команды BOUND (4 байта)-----------------}
{--------и выводит на экран с помощью функций 1 и 2 прерывания 32h------}
{----------------------сообщение об ошибке, номер исключения----------------------}
{-------------------и содержимое регистров общего назначения-------------------}

procedure my_exc_05;assembler;
        asm
                pusha
                push ax
                push bx
                mov ax,201h                     { Функция 2 прерывания 32h: }
                mov bx,113h                        { вывод на экран (1,19) }
                int 32h                                  { значений РОН }
                pop bx
                pop ax
                mov bp,sp                             { Коррекция адреса }
                add word ptr [bp+16],4              { следующей команды }
                mov ah,1                          { Функция 1 прерывания 32h: }
                mov dx,5h       { вывод сообщения "Runtime error: исключение" }
                int 32h                                { и номера исключения }
                mov scan,0
         @w:cmp scan,0              { Задержка до нажатия любой клавиши }
                jz @w
                test scan,80h
                jnz @w
                popa
                db 66h
                iret
        end;{my_exc_05}
{----------------------Обработчик исключения 11 вызывается----------------------}
{--------------при отсутствии сегмента в памяти и выполняет------------------}
{-----------коррекцию дескриптора сегмента, восстанавливая бит P-----------}

        procedure my_exc_11;assembler;
            asm
                pusha
                mov bit32,0                 { Обработчик исключения 16-разрядный
        }
                mov bp,sp
                test byte ptr [offset idt+11*8+5],8             { В поле Type T=1? }
```

```
            jz @1                       { Если нет - обработчик 16-
      разрядный; }
            inc bit32                                { если да - 32-разрядный }
       @1:mov bx,[bp+16]     { Чтение из стека 16 разрядов кода ошибки }
            test bx,7              { Это ошибка доступа к сегменту в GDT? }
            jnz @2
{----------Если нет - переход на вывод сообщения об ошибке,----------}
{----------если да - установка бита P=1 в байте доступа сегмента:--------}
            or byte ptr [offset gdt+bx+5],80h
       @2: mov ax,101h                { Функция 1: установка маркера: }
            mov bx,116h                                { столбец 1, строка 17 }
            int 30h                                    { с очисткой строки }
            mov ah,4                  { Функция 4: вывод на экран строки }
            mov cl,1bh
            mov si,offset s_err          { "Исключение: 11 | код ошибки: " }
            int 30h
            mov ax,501h                        { Функция 5/подфункция 1: }
            mov cl,1eh                                 { вывод на экран }
            mov dx,[bp+16]                    {16-разрядного кода ошибки }
            int 30h                                    { в двоичной форме }
            mov scan,0
    @w:cmp scan,0                  { Задержка до нажатия любой клавиши }
            jz @w
            test scan,80h
            jnz @w
            popa
            add sp,2        { Очистка стека от кода ошибки (разряды 15-0) }
            cmp bit32,0
            jz @3
            add sp,2       { Очистка стека от кода ошибки (разряды 31-16) }
            db 66h
       @3:iret
         end; {my_exc_11}
{----------Обработчик прерываний от часов реального времени (RTC)---------}
{----------формирует с помощью переменной time задержку в мс----------}
{----------и временной интервал (pause) для обработки скан-кодов----------}

procedure RTC;assembler;
      asm
            push ax
{----------Сброс часов реального времени----------}

            mov al,0ch
            out 70h,al
            in al,71h
```

```
{-------------------------Сброс контроллеров прерываний:-------------------------}

            mov al,62h
            out 20h,al          { ведущего }
            mov al,20h
            out 0a0h,al         { и ведомого }
            pop ax
{-------------------------Значение переменной time равно 0?----------------------}
            db 66h,81h,3eh                  { cmp (double) word ptr time,0 }
            dw time
            dd 0
            jz @p
{----------------------------Если нет – уменьшить на 1-----------------------------}
            db 66h,0ffh,0eh                  { dec (double) word ptr time }
            dw time
      @p:
            cmp pause,0
            jz @end
            dec pause
      @end:
            db 66h
            iret
          end;{RTC}
{==================ОСНОВНАЯ ПРОГРАММА================}

    begin
{==================Работа в реальном режиме==================}

{----------------По умолчанию в модуле PROT задан режим экрана---------------}
{------------------с выводом цветного изображения (modeVA=0).-----------------}
{----------------Для задания режима с черно-белым изображением---------------}
{----------------нужно раскомментировать следующий оператор----------------}
{modeVA:=1;}
{---------------Задание векторов прерываний 30h и 32h для работы-------------}
{-----------------------------------в реальном режиме-----------------------------------}

      meml[0:4*$30]:=longint(@int_30h);
      meml[0:4*$32]:=longint(@int_32h);
{----------------------Задание базового адреса видеопамяти----------------------}
{---------------------------для работы в реальном режиме-------------------------}

      asm
            mov ax,0b800h
            mov es,ax
{-----------------------------------Очистка экрана-----------------------------------}
```

```
            mov ax,102h
            mov cl,0h
            mov bx,0
            int 30h
{----------Вывод на экран (5,1) строки "Работа в реальном режиме"----------}

            cmp modeVA,0          { Задан режим вывода на экран цветной? }
            jz @1                                          { Если нет - }
            mov ax,1                { с помощью функции 0, подфункции 1 }
            int 30h                              { установка 16-битного фона }
      @1:
            mov ax,100h                                  { Функция 1, подфункция 0:  }
            mov bx,0501h          { установка маркера (столбец 5, строка 1) }
            int 30h
            mov ah,4                          { Функция 4: вывод строки на экран }
            mov cl,1eh                                        { видеоатрибут }
            mov si,offset s_real                { В SI - адрес строки s_real }
            int 30h                                  { Вывод строки на экран }
{------------------------Вывод на экран (1,3) значений регистров----------------------}
{---------------------------------IP, CS, SS, ES, DS, FS и GS---------------------------------}

            mov ax,300h
            mov bx,103h
            int 32h
{--------------------Вывод на экран (5,6) наименований и значений-------------------}
{-------------------------------------битов регистра CR0---------------------------------}

            mov ax,401h
            mov bx,506h
            int 32h
      end;
{------------------------Добавление к базовой таблице GDT-------------------------}
{----------------------дескриптора сегмента кода программы----------------------}

      init_gdt(code_sel,$ffff,longint(Cseg) shl 4,acc_code,0);
{-------------Формирование данных регистра GDTR и его загрузка-------------}
{--------(параметром является селектор последнего из используемых---------}
{--------------------------в программе дескрипторов GDT)-------------------------}

      init_gdtr(text_sel);
{=========Формирование дескрипторов шлюзов таблицы IDT========}
{=========тех обработчиков прерываний, что разработаны=========}
{====================в программе P_INT====================}
```

```
{------------Дескрипторы шлюзов обработчиков исключений 5 и 11:-----------}

        init_idt(5,ofs(my_exc_05),code_sel,acc_trap);
        init_idt(11,ofs(my_exc_11),code_sel,acc_trap_16);
{-------------Дескриптор шлюза обработчика прерывания от RTC------------}

        init_idt($28,ofs(RTC),code_sel,acc_int);
{----------------------Запрет внешних аппаратных прерываний,----------------------}
{---------сохранение содержимого регистра IDTR реального режима,------------}
{--------------формирование данных регистра IDTR и его загрузка -------------}
{-------------------------для работы в защищенном режиме-------------------------}

        init_idtr;
{-------------------Программирование контроллеров прерываний------------------}
{-------------------------для работы в защищенном режиме-------------------------}

        pic(1);
{-------------------------Определение смещений и селекторов----------------------}
{------------------косвенных межсегментных переходов на метки:---------------}
{-----------end_prot (для окончания работы в защищенном режиме);-----------}
{-----------ret_real (для возврата в реальный режим)-------------------------------}

        asm
                mov ofs_end_prot,offset end_prot
                mov sel_end_prot,code_sel
                mov ofs_ret_real,offset ret_real
                mov sel_ret_real,cs
        end;
{------------Сохранение содержимого сегментных регистров и SP:-------------}

        real_cs:=Cseg;                                          { CS, }
        memw[0:4*$60]:=Dseg;                                    { DS, }
        real_ss:=Sseg;                                          { SS, }
        asm mov real_es,es end;                                 { ES }
        real_sp:=SPtr;                                          { и SP }
{-----------------------------Переход в защищенный режим----------------------------}
{------------------------для МП 80386 и МП последующих моделей-------------------}
{----------------путем установки бита PE в регистре управления CR0-----------}

        asm
                db 0fh,20h,0c0h                         { MOV EAX,CR0 }
                or al,1
                db 0fh,22h,0c0h                         { MOV CR0,EAX }
{------------------------Межсегментный переход на метку @prot-------------------}
{-------------------для загрузки регистра CS и сброса очереди команд-------------}
```

```
                db 0eah
                dw offset @prot
                dw code_sel
{=================Работа в защищенном режиме================}

{---------------------Загрузка сегментных регистров DS, SS и ES-------------------}
{---------------------------соответствующими селекторами-------------------------}

    @prot:
                mov ds,data_sel
                mov ss,stack_sel
                mov es, text _sel
{----------------Разрешение генерации часами реального времени----------------}
{----------------------прерываний IRQ8 (28h) с периодом 1 мс---------------------}

                mov al,0bh
                out 70h,al
                mov al,4ah
                out 71h,al
{----------------------------Сброс часов реального времени-------------------------}

                mov al,0ch
                out 70h,al
                in al,71h

{----------------------------Ожидание нажатия клавиши-----------------------------}

          @w:cmp scan,0
                jz @w
{--------Вывод на экран (5,10) строки "Работа в защищенном режиме:"--------}

                mov ax,103h
                mov bx, 50Ah                    { Установка маркера (5,10) }
                int 30h                          { с очисткой части экрана }
                mov ah,4
                mov cl, 9eh                     { Задан режим мерцания }
                mov si,offset s_prot
                int 30h                       { Вывод строки s_prot на экран }
{------------------Вывод на экран (1,12) значений регистра CR0------------------}

                mov bx,20Ch
                mov ax,400h
                int 32h
{--------------------Вывод на экран (1,14) значений регистров--------------------}
```

```
{------------------------------EIP, CS, SS, ES, DS, FS и GS----------------------------}

            mov bx,10Eh
            mov ax,300h
            int 32h
{-----------------------Загрузка в TR селектора задачи main-----------------------}
{---------------------------(для обработки исключения 12)-------------------------}

            db 0fh,0,1eh          { LTR main_sel:}
            dw v_main_sel
{----------------Разрешение обработки маскируемых прерываний----------------}

            sti
{---------------------------Вывод на экран (1,17) сообщения-----------------------}
{----------------------"Выполняется трехсекундная задержка"--------------------}

            mov ax,100h
            mov bx,111h
            int 30h
            mov ah,4
            mov cl,4fh
            mov si,offset s_delay
            int 30h
{--------------Занесение в переменную time величины задержки в мс-------------}

            db 66h,0c7h,6                 { mov (double) word ptr time,3000 }
            dw time
            dd 3000
{-----------------------------------Выполнение задержки-------------------------------}

      @d1: db 66h,81h,3eh                 { cmp (double) word ptr time,0 }
            dw time
            dd 0
            jnz @d1
{---------------------Удаление с экрана сообщения об задержке--------------------}

            mov ax,101h
            mov cl,0
            mov bx,111h
            int 30h
{------------Задание 16-разрядного фона символов (вместо мерцания)------------}

            mov ax,01h                          { Функция 0, подфункция 1: }
            int 30h                           { Установка 16-битного фона }
{==========Выработка требуемых по заданию исключений==========}
```

```
{---------------------------------Выработка исключения 0-----------------------------}
{----------------------путем выполнения команды деления на нуль-------------------}

            cmp ena_exc00,0          { Разрешена выработка исключения 0? }
            jz @5
            mov ax,1
            mov bl,0
            div bl
{-------------------------------Выработка исключения 5-----------------------------}
{----------путем задания в команде BOUND диапазона и проверяемого---------}
{--------------------значения, выходящего за пределы диапазона------------------}

      @5:
            cmp ena_exc05,0          { Разрешена выработка исключения 5? }
            jz @11
            mov lim1,10                        { Нижняя граница диапазона }
            mov lim2,20                        { Верхняя граница диапазона }
            mov ax,5                             { Проверяемое значение }
            db 62h,6                           { Команда BOUND AX,lim1 }
            dw lim1
{-----------------------------Выработка исключения 11----------------------------}
{--------путем сброса в дескрипторе сегмента данных в таблице GDT ------}
{--------------------признака наличия сегмента в памяти (P=0)--------------------}
{----------------и загрузки селектора этого сегмента в регистр DS-------------}

    @11:
            cmp ena_exc11,0          { Разрешена выработка исключения 11? }
            jz @ scan
            mov bx,data_sel    { Смещением дескриптора сегмента в GDT }
                                 { при TI=0 и RPL=0 является его селектор }
            and byte ptr [offset gdt+bx+5],07fh
            mov ds,data_sel
{------------------Ожидание нажатия или отжатия клавиши,--------------------}
{----------вывод ее скан-кодов на экран и, если нажата клавиша Esc,----------}
{------------------------завершение выполнения программы-----------------------}

@scan:
            mov ax,102h                { Установка маркера с очисткой экрана }
            mov bx,111h                              { в начало 17-й строки (1,17) }
            int 30h
            mov ah,4
            mov si, offset s_scan
            mov cl,1eh                                 { Вывод на экран сообщения: }
            int 30h              { Скан-коды нажатия/отжатия клавиши: }
@wait:
```

```
                mov al,scan
                cmp al,old_scan          { Ожидание изменения скан-код клавиши }
                jz @wait
                mov old_scan,al
                cmp pause,0              { Пауза между скан-кодами одной клавиши }
                jnz @w1                                              { исчерпана? }
                mov pause,200                      { Если да - очистка скан-кодов }
                mov ax,101h                                  { предыдущей клавиши }
                mov bx,2411h
                int 30h
@w1:
                mov ax,300h
                mov cl,1bh
                mov dl,scan                              { Вывод на экран скан-кода }
                int 30h                                 { нажатой/отжатой клавиши }
                cmp scan,1                             { Если не нажата клавиша Esc - }
                jnz @wait{ вернуться к ожиданию нажатия/отжатия клавиш, }
end_prot:                                 { иначе - завершить выполнение программы }

{----------------------------Ожидание нажатия клавиши Esc----------------------------}

@wait:
                cmp scan,1
                jnz @wait
end_prot:
{=============Подготовка к возврату в реальный режим===========}

{--------------------Восстановление режима мерцания символов--------------------}

                mov ax,0h
                int 30h
{-------------------Установка параметров сегментов DS, SS и ES-----------------}
{---------для работы в реальном режиме (Limit=0FFFFh, ED=0, W=1)--------}

                mov ds,real_sel
                mov ss,real_sel
                mov es,real_sel
{-------------------------Запрет маскируемых прерываний--------------------------}
                cli
{----------------------Восстановление атрибутов таблицы IDT--------------------}
{--------------------------для работы в реальном режиме--------------------------}

                db 0fh,1,1eh                                          { LIDT idtr_r }
                dw idtr_r
{-------------------Возврат в реальный режим по команде MOV------------------}
```

```
{----------путем сброса бита PE в регистре управления CR0:----------}

            db 0fh,20h,0c0h                          { MOV EAX,CR0 }
            and al,not 1
            db 0fh,22h,0c0h                          { MOV CR0,EAX }
{----------------------Межсегментный переход на метку real----------------------}

            push real_cs
            push offset real
            retf
{============Работа после возврата в реальный режим============}

{------------------------------Восстановление регистров-----------------------------}

        real:
            xor ax,ax
            mov ds,ax
            mov ds,[4*60h]                                   { DS, }
            mov ss,real_ss                                   { SS, }
            mov es,real_es                                   { ES }
            mov sp,real_sp                                  { и SP }
          end;
{----------------Перепрограммирование контроллеров прерываний---------------}
{--------------------------для работы в реальном режиме--------------------------}

        pic(0);

{------------------Разрешение внешних аппаратных прерываний-----------------}

        asm sti end;                    { маскируемых }
        port[$70]:=$d;              { и немаскируемых }
{--------------------Сброс состояния клавиши-переключателей--------------------}

        mem[$40:$17]:=0;
{------------Сохранение, если было нажатие клавиши "Print Screen",-----------}
{-------------содержимого экрана на диске в подкаталоге EKRAN-------------}

        save_scr;
end.
```

2.19 Індивідуальні завдання

Виконати наступні індивідуальні завдання, пов’язані з обробленням переривань у захищеному режиму роботи мікропроцесора:

1. Розробити оброблювач 0-го виключення, що визначає команду, яка викликала це переривання.

2. Встановити режим покрокового виконання програми та забезпечити виведення значень регістрів мікропроцесора після виконання кожної команди.

3. Промоделювати ситуацію виникнення помилки контролю по паритету в ОЗУ і виконати можливі при цьому дії.

4. Обробити ситуацію виникнення знакового переповнення.

5. Перевірити виникнення виключення при виконанні команди BOUND.

6. Створити і обробити ситуацію, що викликає виключення по недозволеному коду операції.

7. Перевірити виникнення виключення, коли тип операнда не відповідає коду операції.

8. Перевірити виникнення виключення при невірному використанні префікса LOCK.

9. Промоделювати ситуацію, коли співпроцесор недосяжний, і обробити виникаюче при цьому виключення.

10. Перевірити виникнення у мікропроцесора стану виключення (shutdown) при невдалій спробі обробити подвійне переривання.

11. Написати оброблювач виключення 10 – "недозволений TSS", з'ясувавши за допомогою коду помилки причину виникнення даного переривання.

12. Написати оброблювач виключення 12, що усуне причину виникнення цього переривання.

13. Перевірити правила звернення до сегментів даних.

14. Промоделювати і обробити ситуацію виникнення помилки FPU.

15. Перевірити виникнення виключення 17 при не вирівняних даних.

16. Перевірити режим трасировки задач.

17. Написати оброблювач переривань від клавіатури, який визначає ASCII-коди російських і англійських літер.

18. Написати оброблювач переривань від системного таймера, що виводить на екран кожні 5 секунд повідомлення "System timer".

19. Написати оброблювач переривань від годинника реального часу, що реалізує функцію часової затримки.

20. Написати оброблювач переривань від годинника реального часу, що реалізує функцію будильника.

3. ОРГАНІЗАЦІЯ МУЛЬТІЗАДАЧНОЇ РОБОТИ МІКРОПРОЦЕСОРА

Задача – це одиниця роботи процесора по здійсненні ним операцій диспетчеризації програмного коду, його виконанню та завершенню. Це поняття може бути застосоване до програм, підпрограм, процесів, оброблювачів переривань та виключень.

Архітектура та апаратні засоби процесорів сімейства x86 забезпечують зберігання стану задачі, її виконанню та переключенню до іншої задачі. Кожна програма, що застосовує механізми захисту пам'яті, навіть проста система, має мати принаймні одну задачу. Більш складні системи можуть застосовувати можливості процесора по управлюнню задачами для підтримки багатозадачного режиму.

При мультізадачній роботі МП в пам'яті зберігаються водночас декілька програм і даних для виконання декількох процесів оброблення інформації. При цьому забезпечується взаємний захист програм і даних. Механізм мультізадачної роботи був впроваджений в процесорах, починаючи з МП 80286, для оптимізації використання процесорного часу – якщо одна із задач очикує дані, наприклад, нажаття клавіши клавіатури при редагуванні тексту документу, то інші задачі можуть в цей час виконуватися, зменшуючи простої процесора.

3.1. Апаратні засоби підтримки мультізадачного режиму

Мультізадачний режим може бути реалізований не тільки у захищеному режимі. Робота мікропроцесора, при якій кожній задачі виділяється квант часу і здійснюється переключення задач, може буде організована також в реальному режимі. Але тільки у захищеному режимі розроблювач програми, яка застосовує задачі, має змогу використати апаратні засоби підтримки мультізадачної роботи, які має процесор, а саме: сегмент стану задачі та регістр задач. Крім того засоби захисту пам'яті (див. розділ 7) дозволяють створювати надійні мультізадачні програми, прикладами яких є сучасні операційні системи.

3.1.1. Сегмент стану задачі

Кожна задача повинна мати свій сегмент стану – TSS (Task State Segment), структура якого зображена на рисунку 3.1.

	Селектор повернення		0h
ESP0			4h
0	SS0		8h
ESP1			Ch
0	SS1		10h
ESP2			14h
0	SS2		18h
CR3			1Ch
EIP			20h
EFLAGS			24h
EAX			28h
ECX			2Ch
EDX			30h
EBX			34h
ESP			38h
EBP			3Ch
ESI			40h
EDI			44h
0	ES		48h
0	CS		4Ch
0	SS		50h
0	DS		54h
0	FS		58h
0	GS		5Ch
0	LDT		60h
Адреса БКВВ	0	T	64h
Інформація операційної системи (програми)			
Бітова карта введення-виведення (БКВВ)			
		11111111	

Рисунок 3.1 – Формат сегмента стану задачі

Вона має дві частин:

а) обов’язкову (104 байта), яка містить всю інформацію, необхідну мікропроцесору для рішення даної задачі;

б) додаткову, що містить:

– дані, які можуть бути використані операційною системою (програмою) для допоміжних цілей, наприклад, для завдання імені задачі, коментаря до неї та ін.;

– бітову карту введення-виведення (БКВВ), що застосовується для дозволу чи заборони доступу до окремих портів з метою захисту даних (див. розділ 7) і не має прямого відношення до роботи з задачами.

Оскільки розмір області TSS, що використається ОС, може бути довільний, для визначення початку області БКВВ використовується поле адреса БКВВ, в якому вказується адреса бітової карти введення-виведення відносно початку TSS. Якщо адреса БКВВ дорівнює нулю, це означає, що бітова карта введення-виведення не використовується.

Кожний біт бітової карти введення-виведення відповідає окремому однобайтному порту введення/виведення (від порту з адресою 0 до порту з адресою 65535).

За останнім байтом БКВВ повинен бути заключний байт з одиницями в кожному біті. Адреса цього байта повинна відповідати границі сегмента, що визначається дескриптором TSS.

У перших двох байтах основної частини TSS зберігається селектор попередньої задачі, яка викликала дану задачу командою CALL. Це дозволяє потім повернутися до старої задачі.

Далі йдуть три подвійних поля, в яких зберігаються адреси стеків (SS:ESP), які використовуються при викликах процедур різного рівня привілеїв (див. розділ 4) і прямого відношення до мультізадачної роботи не мають. У інших полях TSS міститься вміст регістрів МП, який ще називають контекстом задачі.

При переключенні задач вміст регістрів мікропроцесора старої задачі переписується в її TSS а вміст полів TSS нової задачі завантажується в відповідні регістри МП. Вміст ряду полів (CR3, адреси стеків) при цьому не змінюється.

Якщо біт T сегменту TSS дорівнює 1, то при переключенні на дану задачу після виконання її першої команди генерується переривання 1, яке може бути використане при налагоджуванні задач.

Границя сегмента TSS, що указана в дескрипторі, повинна бути не менше 103 (67h) байтів, інакше виникає переривання 10 (виключення невірного TSS).

Дескриптор TSS має такий же формат, що і дескриптор сегмента пам'яті (див. підрозділ 1.2.1) за винятком старшої частини байта 6 і байта доступу. Чотири біти старшої частини байта 6 дескриптора TSS мають такі значення: ***G, 1, 0, 0***, де **G** – біт дрібності.

Формат байта доступу дескриптора TSS наведений на рис. 3.2:

7	6	5	4	3	2	1	0
P	DPL		S=0	T	0	B	1

Рисунок 3.2 – Формат байта доступу TSS

Біти P, DPL та ***S*** мають традиційне значення (див. підрозділ 1.2.1);

Біт T – визначає тип МП: T=0 - 80286, T=1 – 32-розрядний МП;

Біт B (Busy) – біт зайнятості, встановлюється в 1 при переключенні на дану задачу.

3.1.2. Регістр задач

Регістр задач – TR (Task Register) містить 16-розрядний селектор сегмента TSS для поточної задачі, який вказує на її дескриптор TSS, що знаходиться в таблиці GDT. Крім того, у його тіньовому (дескрипторному) регістрі зберігається 32-розрядна базова адреса сегмента, 16-розрядна границя сегмента та атрибути дескриптора.

Регістр TR можна завантажити командою LTR Після цього вміст TR змінюється при переключенні задач командами JMP, CALL та IRET.

3.2. Переключення задач

Для переключення задач МП використовує звичайні команди міжсегментного переходу: JMP, CALL та IRET. Якщо селектор команди JMP або CALL вказує в таблиці GDT на дескриптор не сегмента коду, а на дескриптор, що має S=0, тобто системний, який в полі типу має значення 1001, то виконується не виклик процедури, а переключення задачі. При цьому поле відносної адреси (зміщення) команди ігнорується, і мікропроцесор виконує такі дії:

– вміст регістрів МП переписується в TSS поточної задачі (селектор дескриптора цього TSS береться з регістра TR);

– в регістр TR заноситься селектор дескриптора TSS нової задачі, що береться з команди, а в зв’язаний з ним програмно недосяжний (тіньовий) регістр завантажується дескриптор TSS нової задачі;

– з TSS нової задачі в МП завантажується вміст регістрів (оскільки при цьому завантажуються регістри CS та EIP, то з цього моменту починає виконуватися нова задача);

– встановлюється біт ***TS*** регістра CR0;

– встановлюється біт ***B*** байта доступу нової задачі, при цьому якщо переключення задач здійснювалось командою JMP, то біт **B** байта доступу старої задачі скидається;

– встановлюється біт ***NT*** регістра EFLAGS у випадку переключення задач командою CALL.

Переключення задачі здійснюється лише в тому випадку, якщо біт зайнятості ***B*** нової задачі має значення 0 (інакше виникає переривання 13). Це зроблено для того, щоб в мультіпроцесорній системі декілька процесорів не могли одночасно виконувати одну й ту ж задачу. У зв’язку з цим забороняється рекурсивний виклик задач.

Як вже повідомлялося, при переключенні задач вміст регістрів процесора апаратно зберігається в TSS, але це не стосується вмісту регістрів пристрою FPU – при переключенні задач він може бути втрачений. Тому, коли процесор зустрічає команду з плаваючою крапкою або MMX-команду при встановленому біті TS в регістрі CR0, тобто, коли починає працювати нова задача, виникає виключення 7 “Недоступний пристрій FPU”. Оброблювач цього виключення може зберегти контекст FPU в пам’яті до виконання команд з плаваючою крапкою або MMX-команд.

Значення прапора NT регістра CR0 використовується МП при виконанні команди IRET. Якщо NT = 0, то команда виконується звичайно (із стека вибираються значення EIP, CS та EFLAGS і завантажуються в ці регістри). При NT = 1, тобто у випадку, коли має місце повернення з задачі, МП використовує поле TSS "Селектор повернення" як селектор дескриптора TSS нової задачі. Після цього йде переключення задач, як це описано вище.

3.3. Опис програми P_TASK

Для організації мультізадачної роботи процесора, по-перше, необхідно виконати всі дії, які розглянуті в розділі 5,”” по переводу МП у захищений режим, здійсненню обробки переривань у захищеному режимі та повертанню МП в реальний режим.

Крім того, необхідно:

– розробити задачі у вигляді процедур, що виконують певні завдання;

– для кожної задачі сформувати сегмент стану задачі;

– сформувати дескриптори цих сегментів та записати їх в таблицю GDT;

– здійснити переключення задач.

Як приклад організації мультізадачної роботи процесора розглядається програма **P_TASK**.

3.3.1. Розробка задач

Першим кроком організації мультізадачної роботи МП є розробка задач. У програмі **P_TASK** використовуються такі задачі:

1. Задача MAIN – це задача, якій операційна система першою передає управління. У реальному режимі вона виконує всі дії програми **P_TASK** з організації багатозадачної роботи МП (дивись розділ 2 «РОБОТА З ПЕРЕРИВАННЯМИ У ЗАХИЩЕНОМУ РЕЖИМУ” та підрозділи 3.3.2).

Дії задачі MAIN у захищеному режимі наведені в підрозділі 3.3.3.

2. Задача KEYB_TASK реалізована в вигляді процедури k**eyb_task**, яка є оброблювачем переривань від клавіатури й при натисненні та відтисненні клавіші клавіатури виконує такі дії:

– здійснює скидання біта обслуговування переривання від клавіатури:

```
mov al,20h
out 20h,al ;
```

– читає скан-код клавіші з порту 60h;

– викликає процедуру **save_ekran**, яка заносить в змінну **scan** скан-код натиснутої або відтиснутої клавіші та аналізує скан-коди з метою зберегти зображення екрану по натисненню клавіші “Print Screen” (дивись опис процедури **save_ekran** в підрозділі 2.15 “Зберігання зображення екрану у захи-

щеному режимі”);

– якщо натискається клавіша “лівий Shift” (скан-код – 2Ah), то в змінну **left_shift** заноситься 1, якщо відпускається клавіша “лівий Shift” (скан-код – 0AAh), то в змінну **left_shift** заноситься 0, це дозволяє розпізнавати комбінації клавіш “лівий Shift-D/d” в задачі DATE_TASK, “лівий Shift-M/m” та лівий Shift-I/i” в процедурі **dispatcher**;

– по натисненню клавіші «D/d» викликає командою CALL задачу DATE_TASK:

db 9ah_task
dw 0
dw date_sel.

Реалізація оброблювача переривання від клавіатури в вигляді задачі зроблена з навчальною метою – показати, що оброблювачі переривань також можуть бути задачами, хоча якихось переваг перед традиційними підходами формування оброблювачів в вигляді процедур це не дає. Тому інші оброблювачі зовнішніх переривань – від системного таймеру та годинника реального часу, розроблені як процедури.

3. Задача PASSW_TASK реалізована в вигляді процедури **pasw_task**, яка здійснює введення паролю та по результату його аналізу дозволяє, якщо пароль вірний, чи не дозволяє в протилежному випадку, користувачеві мати дозвіл до меню задач для зміни стану та статусу задач

4. Задача DATE_TASK реалізована в вигляді процедури **date_task**, що викликається задачею KEYB_TASK при натисненні клавіші “D/d”, виконує такі дії:

– визначає за допомогою годинника реального часу (RTC) день місяця, номер місяця і дві молодші цифри року та виводить поточну дату на екран в форматі ДД-ММ-20РР, наприклад, **07-12-2007**;

– аналізує змінну **left_shift** і, якщо вона дорівнює 1, тобто натиснута клавіша “лівий Shift”, витирає з екрану виведену раніше дату.

Решта задач є задачами, які працюють в режимі багатозадачної роботи (в режимі переключення задач). Це такі з адачі: TIME_TASK, COLOR_TASK, SOUND_TASK та DURATION_TASK.

5. Задача TIME_TASK, яка реалізована в вигляді процедури **time_task,**

викликає 32-розрядною командою CALL процедуру **get_time** модуля **PROT**, що визначає за допомогою годинника реального часу поточний час та виводить його на екран в форматі ГГ:ММ:СС, наприклад,**19:07:15**.

6. Задача COLOR_TASK, яка реалізована в вигляді процедури **color_task**, переміщує по екрану прямокутник, демонструючи роботу з відеопам'яттю у захищеному режимі.

7. Задача SOUND_TASK, яка реалізована в вигляді процедури **sound_task**, програє мелодію «Новорічна» за допомоги каналу 2 системного таймеру комп'ютера. При цьому масив **mf** містить частоти звуків (нот) мелодії, масив **md** – тривалості звуків, а в масиві **k** формуються на базі масиву **mf** коефіцієнти ділення каналу 2. системного таймеру.

Часові затримки звуків формуються за допомоги змінної **time1**, значення, якої зменшується на 1 до нуля по кожному перериванню від годинника реального часу, тобто кожну милісекунду.

8. Задача DURATION_TASK, яка реалізована в вигляді процедури **duration_task**, виконує такі дії:

– за допомоги типізованої константи

```
s_dur:array[0..2] of string=
        (' С момента пуска прошло   ',
         '              мс      ',
         '              тактов МП');
```

виводить на екран форму для виведення даних;

– за допомоги змінної **time3**, значення якої збільшується на 1 по кожному перериванню від RTC, виводить на екран час в мілісекундах, що пройшов після запуску програми **P_TASK**;

– за допомоги лічильника TSC (Time Stamp Counter визначає кількість тактів роботи МП за цей же час.

На рис. 3.3 приведене зображення екрану, яке показує, що з початку виконання програми **P_TASK** у захищеному режимі при роботі одної задачі DURATION_TASK до її закінчення пройшло 52 мілісекунди або було виконано 5243164h тактів МП.

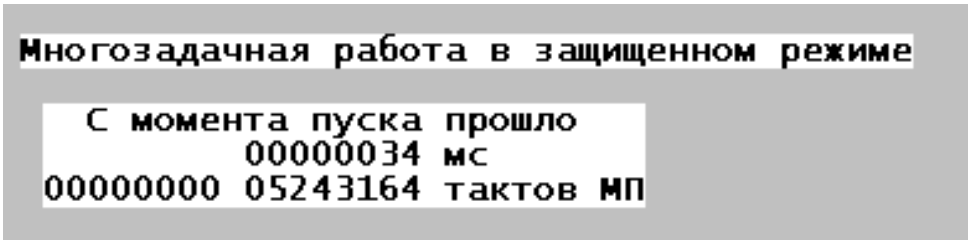

Рисунок 3. 3 – Зображення екрану після виконання задачі DURATION_TASK

Кожна з задач, які виконуються в режимі переключення задач, має свій номер, стан та статус.

Цим задачам присвоєні такі номери:

– TIME_TASK 1;

– COLOR_TASK 2;

– SOUND_TASK 3;

– DURATION_TASK 4.

Задача MAIN, яка не працює в режимі переключення задач, але починає його та завершує, має номер 0. Оскільки з задачі MAIN починається переключення задач, селектор **main_sel** саме цієї задачі повинен бути заздалегідь завантажений в регістр задач TR (дивись підрозділ 3.3.3).

Стан кожної задачі, що переключається, має два значення:

– 0 – задача не виконується (стан задачі пасивний);

– 1 – задача виконується (стан задачі активний).

Статус цих задач також має два значення:

– 0 – задача виконується безперервно;

– 1 – задача виконується після її запуску тільки один раз.

Стан та статус всіх задач, що переключаються, зберігаються в двомірному масиві **state_arr.**

Типізована константа **max_n** задає число задач, що підлягають переключенню (значення співпадає з максимальним номером задачі) і в програмі **P_TASK** дорівнює 4.

3.3.2. Формування сегментів TSS, їх дескрипторів та стеків задач

У відповідності до формату TSS (див. підрозділ 3.1.1) був розроблений тип **t_tss** змінних, які зберігають значення TSS всіх задач, що застосовує програма. Опис цього типу змінних міститься не в програмі **P_TASK,** а в модулі

PROT. Це зроблено, щоб інші програми, які застосовують задачі, також могли його використати.

Структура TSS задається таким типом запису:

```
t_tss=record                                        { Структура TSS: }
      link,                                         { селектор повернення, }
      esp0,ss0,                                     { покажчик стека кільця 0, }
      esp1,ss1,                                     { покажчик стека кільця 1, }
      esp2,ss2,                                     { покажчик стека кільця 2, }
      cr3,eip,eflags,eax,ecx,                       { регістри }
      edx,ebx,esp,ebp,esi,edi,
      es,cs,ss,ds,fs,gs,ldtr:longint;               { процесора, }
      bit_t:byte;                                   { біт пастки задачі }
      adr_BKVV:word;        { зміщення БКВВ відносно початку TSS }
      sys_inf:string;                 { поле для системної інформації }
      BKVV,                       { бітова карта введення-виведення }
      byte_end:byte                                 { байт завершення TSS }
end;
```

Для кожної задачі в програмі **P_TASK** об’явлені змінні типу **t_tss**: **keyb_tss, passw_tss, date_tss, time_tss,color_tss, sound_tss, duration_tss.**

Також в модулі **PROT** об’явлена змінна **main_tss** цього типу, яка застосовується во всіх программах (за винятком **P_MODE**),що працюють у захищеному режимі.

Крім того, кожна задача повинна мати свій стек, інакше вміст стеку при переключенні задач буде втрачений. В програмі **P_TASK** стеки задач реалізовані в вигляді масивів розміром 256 байтів, які розташовані в сегменті даних:

keyb_stack, passw_stack, date_stack, time_stack, color_stack, sound_stack, duration_stack: array[0..255] of byte;

Формат TSS містить 29 полів, але далеко не всі ці поля використовуються в роботі тієї чи іншої задачі. Тому процедура **init_tss** (модуль **PROT**), яка розроблена для формування TSS всіх задач, має тільки шість параметрів, п’ять з яких задають вміст найбільш важливих полів TSS:

init_tss(var tss:t_tss; cs,ds,es,ip,sp:word),

де

tss – ім’я змінної типу **t_tss**, в якій формується сегмент стану даної задачі;

cs, ds, es – селектори регістрів CS, DS та ES;

ip – зміщення задачі (точка входу задачі);

sp – значення вказівника стека задачі.

Крім того, поля **ss, eflags, bit_t** та **byte_end** заповнюються автоматично:

tss.ss:=ds;

tss.eflags:=$200;

tss.bit_t:=0;

tss.byte_end:=$ff.

Наприклад, формування TSS задачі TIME_TASK в програмі **P_TASK** виконується так:

init_tss(time_tss,code_sel,data_sel,video_sel,
ofs(time_task),ofs(time_stack)+sizeof(time_stack);

Аналогічно здійснюється формування TSS всіх інших задач, крім задачі MAIN**,** яка на момент переключення задач вже працює, і тому для неї значення полів TSS не формуються, а переписуються з поточних значень відповідних регістрів МП при першому переключенні задач.

Усі інші поля TSS для кожної задачі, якщо це потрібно, задаються окремо без використання процедури **init_tss**. Наприклад, задачі, що працюють в режимі переключення, мають дані в полі TSS «Інформація для операційної системи», які процедура **dispatch** виводить на екран по натисненню клавіші I/i. Інші задачі це поле не використовують.

Формування вмісту поля «Інформація операційної системи (програми)» для задачі TIME_TASK робиться таким чином:

time_tss.sys_inf:=’TIME_TASK выводит на экран текущее время’;

Окрім формування самого TSS для кожної задачі необхідно сформувати дескриптор цього TSS та записати його в GDT (в доповнення до дескрипторів сегментів пам’яті). Наприклад, для задачі TIME_TASK це зроблено так:

init_gdt(time_sel,sizeof(t_tss)-1,longint(Dseg) shl 4+ofs(time_tss),acc_TSS,

0);,

де **acc_TSS** (модуль **PROT**) **–** байт доступу дескриптора TSS, сформований відповідно до його формату, що зображений на рис. 3.2.

3.3.3 Робота програми P_TASK у захищеному режимі

1. Після переходу мікропроцесора до захищеного режиму продовжує працювати задача MAIN, яка виконує такі дії:

– задає для каналу 2 системного таймеру режим генерації імпульсів типу меандр (режим 3);

– здійснює ініціалізацію годинника реального часу RTC (Real Time Clock), яка передбачає установлення дозволу на генерацію цим пристроєм переривань з частотою 1 Мгц та його скидання (дивись підрозділ 5.13 розділу 5 “Робота з перериваннями у захищеному режимі”);

– за допомоги функцій 1 та 4 переривання 30h очищає екран та виводить на екран (1,1) повідомлення: “Многозадачная работа в защищенном режиме” (дивись рис. 6.) ;

– завантажує в регістр. задач TR селектор дескриптора TSS задачі MAIN;

– викликає за допомоги команди CALL задачу passw_task для прийому та аналізу паролю:

```
db 9ah
dw 0
dw passw_sel;
```

– постійно аналізує поточне значення змінної scan – якщо її значення стане дорівнювати 1 (тобто, була натиснута клавіша “Esc”), то завершує роботу програми у захищеному режимі й переходить до дій по поверненню процесора до реального режиму.

2. Після виклику задача passw_task виконує такі дії:

– завантажує в регістр GS селектор сегмента даних LDT задачі PASSW_TASK:

```
db 8eh,2eh                          { MOV GS,dataLDT_sel }
dw dataLDT_sel;
```

– заносить в сегмент даних LDT скан-коди трьох клавіш (“I”,”S” та “Z”), які складають пароль;

– за допомоги функцій 1 та 4 переривання 30h виводить на екран (1,3) повідомлення: **“Введите пароль или нажмите Enter” (**дивись рис. 3.4);

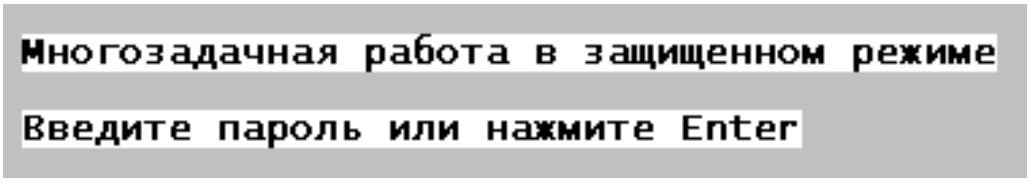

Рисунок 3.4 – Зображення екрану після пуска програми

– очікує натиснення клавіші на клавіатурі;

– здійснює аналіз скан-кодів натиснутих клавіш з метою вияснити правильність введеного паролю;

– якщо скан-коди введених клавіш складають пароль, то встановлює признак “пароль вірний” шляхом занесення “1” в біт 5 (X) байту 6 дескриптора сегмента коду;

– якщо до введення паролю (або при введенні паролю) була натиснута клавіша “Enter”, то аналіз паролю закінчується достроково без встановлення признака “пароль вірний”;

– затирає на екрані рядок **“Введите пароль или нажмите Enter”;**

– дозволяє обробку переривань від системного таймеру та годиника реального часу:

```
mov al,0
out 21h,al
out 0a1h,al
```

3. Після цього по перериванням від системного таймеру або від годинника реального часу (це визначається значенням типізованої константи **t_disp**, яка за умовчанням має значення 0 та задає переривання від системного таймеру) :викликається процедура диспетчеризації задач **dispatch.** Ця процедура у відповідності до прийнятої дисципліни обслуговування задач, що передбачає почережну безпріоритетну роботу задач, та згідно з поточним значенням стану та

статусу кожної задачі, які задаються масивом state_arr, здійснює переключення задач, виконуючи такі дії:

– якщо нажата клавіша “M/m” та встановлений признак правильності введеного паролю, викликає процедуру **menu** для виведення меню задач на екран;

– якщо нажата клавіша “I/i”, виводить на екран інформацію про призначення задачі з поля **sys_inf** сегмента TSS поточної задачі;

– якщо нажата комбінація клавіш “лівий Shift-I/i”, стирає на екрані виведену раніше інформацію ОС.

– якщо нажата клавіша “Escape”, черговою задачою буде задача MAIN, на яку буде здійснює переключення для завершення програми;

– визначає номер чергової задачі, яка має активний стан;

– по номеру задачі за допомоги масиву

sel_arr:array[1..max_n] of byte=($90,$98,$A0,$A8)

визначає селектор її сегменту TSS та заносить його в змінну **sel_new_task;**

– якщо продовжує виконуватися та ж сама задача, переключення задач не відбувається (бо задача не може переключитися сама на себе);

– якщо активних задач нема, буде здійснене переключення на задачу MAIN для продовження виконання програми;

– злійснює переключення на задачу, селектор TSS якої міститься в змінній **sel_new_task**:

```
db 0ffh,2eh
dw ofs_new_task,
```

де **ofs_new_task** та **new_task_sel** – дві поспіль розташовані комірки пам’яті, які містять адресу переходу: відповідно зміщення та селектор.

Відмітимо, що для задач значення зміщення в командах JMP та CALL не має значення. Отже, й значення змінної **ofs_new_task** теж може бути будь-яким, хоча згідно з форматом команди вона повинна бути указана.

Начальние значення стану та статусу задач, що задається за допомоги типізованої константи (масиву) **state_arr:**

```
state_arr:array[0..1,1..max_n] of byte=
((1,1,0,1),                    { Стан задач: 1/0 (виконується/ні), }
(0,0,0,0));          { статус задач: 1/0 (один раз/постійно) },
```

означає, що будуть виконуватись задачі з номерами 1, 2 та 4, тобто TIME_TASK, COLOR_TASK та DURATION_TASK, причому всі задачі будуть виконуватись постійно (доти працює програма **P_TASK).**

На рисунках 3.5 та 3.6 наведені зображення екрану, які показують в динаміці результати роботи трьох задач з інтервалом в дві секунди.

З рисунків видно, що:

– задача TIME_TASK виводить поточний час в форматі ГГ:ММ:СС (14:50:46 та 14:50:48);

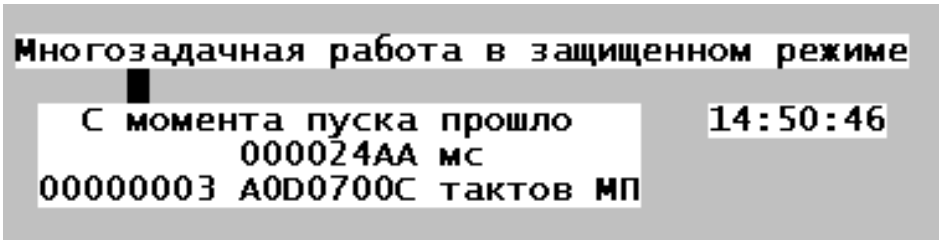

Рисунок 3.5 – Зображення екрану при виконанні задач TIME_TASK, COLOR_TASK та DURATION_TASK

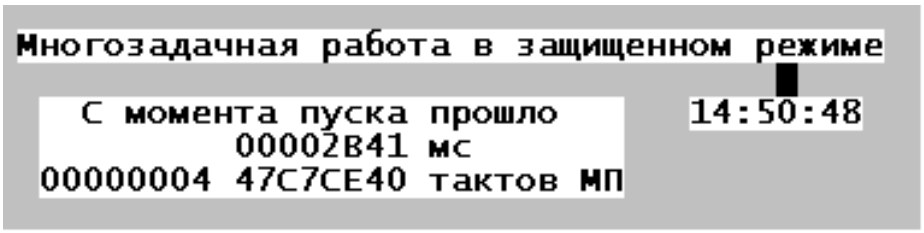

Рисунок 3.6 – Зображення екрану через дві секунди

– задача COLOR_TASK перемішує по екрану чорний прямокутник;

– задача DURATION_TASK показує, скільки мілісекунд пройшло (24AAh, 2841h) та скільки тактів роботи мікропроцесора було виконано (3A0D0700Ch, 447C7CE40h) з моменту запуску багагозадачного режиму до моменту натиснення клавіші “Print Screen” для зберігання поточного вмісту екрану.

Кожна з цих задач при своєму завершенні аналізує свій біт статусу в масиві state_arr. Якщо він дорівнює 0, то робота цієї задачі продовжується з початку, якщо ж статус задачі є одноразовим, то задача скидає свій біт стану в масиві **state_arr,** змінюючи його з активного на пасивний, та викликає процедуру

dispatcher для переключення на чергову задачу (щоб марно не тратити залишок свого кванта часу).

За допомоги процедури **menu,** яка викликається по натисненню комбінації клавіш лівий Shift-M", кожний користувач, який правильно указав пароль, може змінити стан та статус кожної задачі. При цьому на на екран виводиться меню, яке містить стан та статус всіх задач.

За допомоги клавіш "стрілка вверх", "стрілка вниз" "стрілка вправо" та "стрілка вліво" можна вибрати пункт меню, за допомоги клавіші "Space" – змінити значення пункту на протилежне, а за допомоги клавіш і "Enter" – зберегти нове значення статусу та стану задач та продовжити переключення задач.

На рис. 3.7 показано зображення екрану з виведеним меню зразу після натиснення клавіші "M/m", з якого видно, що працюють три задачі – TIME_TASK, COLOR_TASK та DURATION_TASK.

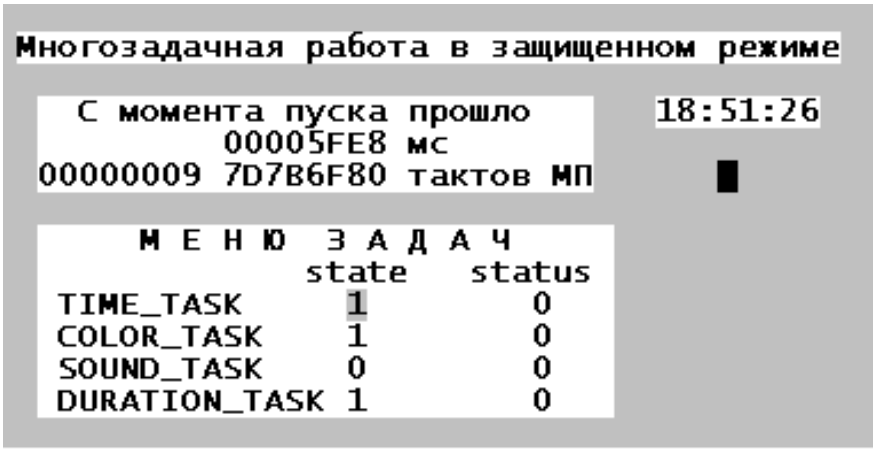

Рисунок 3.7 – Зображення екрану при виведенні меню

На рис. 3.8 показано зображення екрану меню із зміненим станом та станусом задач:

– задача COLOR_TASK одержала стан 0 й перестала виконуватися (положення прямокутника на рис. 6.8 та 6.9 одне і теж);

– задача SOUND_TASK стала активною й отримала статус 1: пісдя завершення роботи з меню буде виконана один раз (програє новорічну мелодію) ;

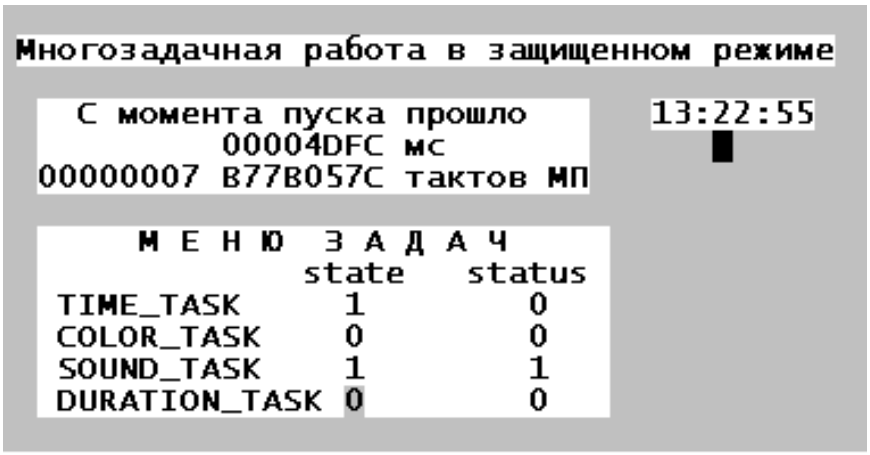

Рисунок 3.8 – Зображення екрану із зміненим меню

– задача DURATION_TASK одержала стан 0 й перестала виконуватися (значення числа мілісекунд та тактів МП на рис. 3.8 та 3.9 одне і теж) ;

На рис. 3.9 показано зображення екрану при виконанні задач після зміни стану та статусу деяких з них (дивись стан та статус задач на рис. 3.8).

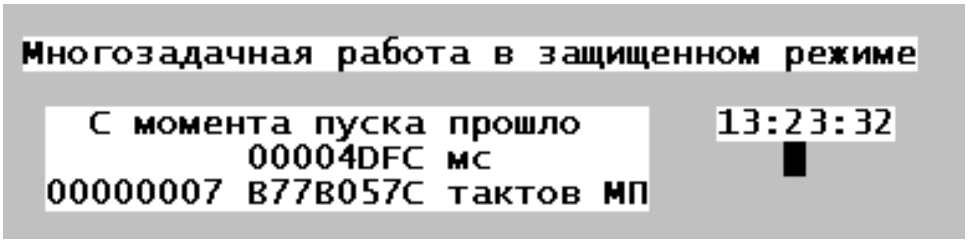

Рисунок 3.9 – Зображення екрану при виконанні задач TIME_TASK та SOUND_TASK

У програмі **P_TASK** для виявлення того, чи впливає і як величина кванта часу на якість виконання задач, використовуються, як вже повідомлялося вище, два джерела переключення задач:

– переривання від системного таймеру з періодом 55 мс;

– переривання від годинника реального часу з періодом 1 мс.

Значення типізованої константи **t_disp** вказує, яке з цих джерел буде використане:

0 – системний таймер (задається за умовчанням);

1 – годинник реального часу.

Експериментальне виконання програми **P_TASK** з переключення задач

від системного таймеру та від годинника реального часу не виявило помітної різниці в виконанні вказаних задач.

Переключення на чергову задачу може здійснити також кожна з задач багато задачного режиму в випадку, коли вона завершила свою роботу і має статус одноразового виконання. Це зроблено для того, щоб інші задачі не гаяли марно часу чекаючи, поки закінчиться квант часу задачі, яка вже завершила роботу.

4. По натисненню кдавіщі “D/d”викликається задача DATE_TASK, яка виводить на екран поверх повідомлення “**Многозадачная работа в защищенном режиме**” значення поточної дати (дивись рис. 3.10), а по натисненню комбінації кдавіщ “лвий Shift-D/d” – відновлює повідомлення (дивись рис. 3.11).

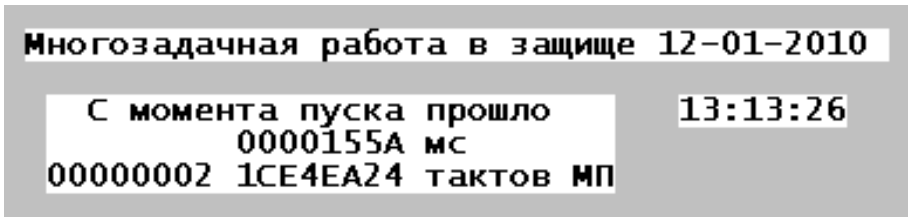

Рисунок 3.10 – Зображення екрану з виведеною датою

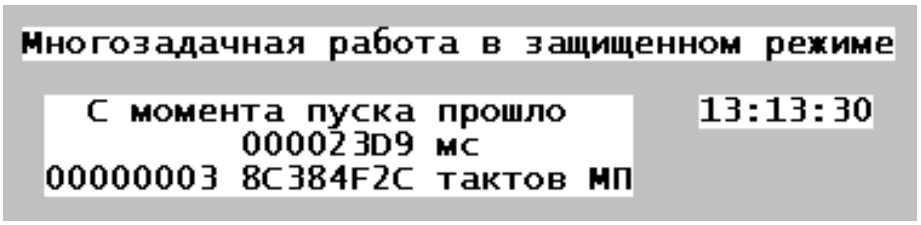

Рисунок 3.11 – Зображення екрану з відновленим повідомленням

6. По натисненню кдавіші “I/i” на екран виводиться інформація щодо задачі, яка виконується в даний момент (дивись рис. 3.12).

У програмі **P_TASK** крім переключення задач за допомогою міжсегментної команди JMP, **з**дійснений також виклик задачі **Task_Date** за допомоги міхсегментної команди CALL:

```
db 9ah      { Виклик задачі Task_Date }
dw 0        { командою  CALL }
dw date_sel
```

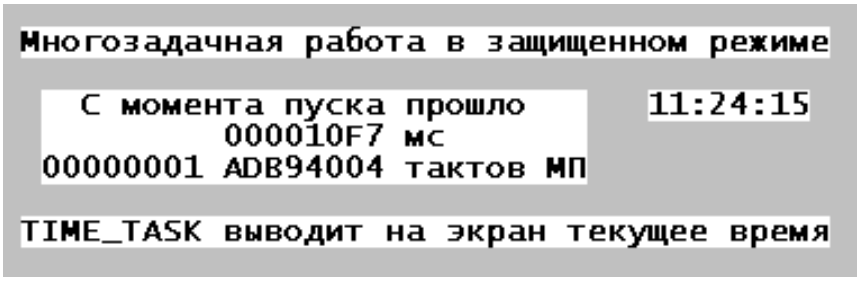

Рисунок 3.12 – Зображення екрану з інформація щодо задачі

3.4. Системний таймер

3.4.1. Структура таймера

Таймер 8254 складається з трьох незалежних каналів, або лічильників. Кожний канал містить такі регістри:

– стану каналу RS (8 розрядів);

– керуючого слова RSW (8 розрядів);

– буферний регістр OL (16 розрядів);

– регістр лічильника CE (16 розрядів);

– регістр констант перелічення CR (16 розрядів).

Канали таймера підключаються до зовнішніх пристроїв за допомогою трьох ліній:

– *GATE* – керуючий вхід;

– *CLOCK* – вхід тактової частоти;

– *OUT* – вихід каналу таймера.

Регістр лічильника CE працює в режимі віднімання поточного змісту, зменшується по задньому фронту сигналу CLOCK при умові, що на вході GATE встановлена логічна 1. У залежності від режиму роботи таймера при досягненні лічильником CE нуля тим або іншим шляхом змінюється вихідний сигнал OUT.

Буферний регістр OL призначений для запам’ятовування поточного змісту регістра лічильника CE без зупинки процесу лічби. Після запам’ятовування буферний регістр доступний програмі для читання.

Регістр констант перелічення CR може завантажуватися в регістр лічильника, якщо це вимагається в поточному режимі роботи таймера. Регістри стану каналу і керуючого слова призначені відповідно для визначення поточного стану каналу і для завдання режиму роботи таймера.

Спрощена схема взаємодії регістрів каналу наведена на рис. 3.13. При роботі з перезапуском не вимагається повторного програмування таймера для виконання тієї ж функції. По фронту сигналу GATE значення константи з

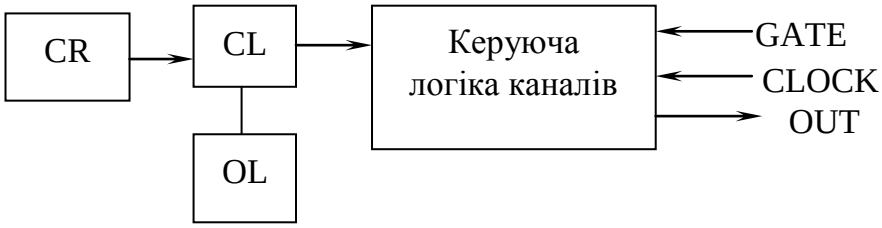

Рис. 3.13 Структура таймера

регістра CR знов переписується в регістр CE, навіть якщо поточна операція не була завершена. Формат керуючого регістра наведений на рис. 3.14.

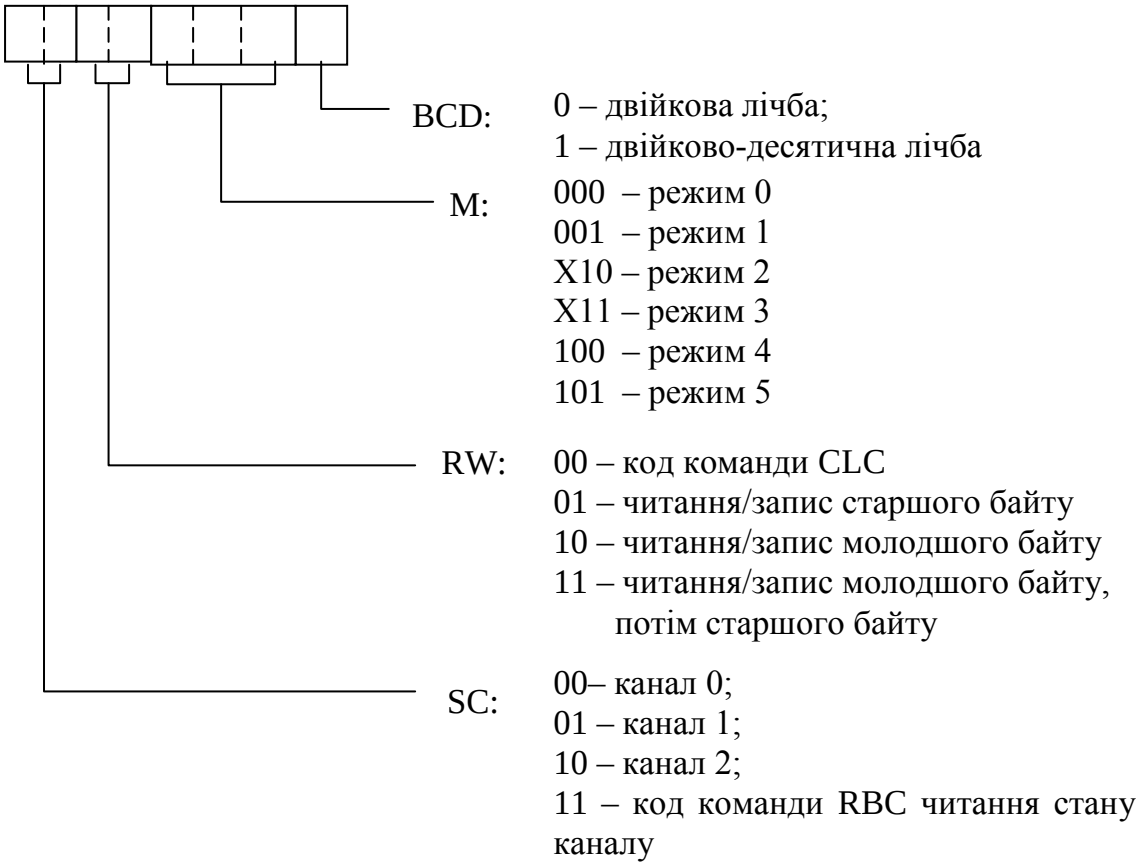

Рис. 3.14. Формат керуючого слова

У двійково-десятковому режимі константа задається в діапазоні 1-9999. Поле M визначає такі режими роботи мікросхеми 8254:

0 – затримка;

1 – мультивібратор, що чекає;

2 – генератор тактових імпульсів;

3 – генератор меандру;

4 – строб, що керується програмно;

5 – строб, що керується апаратно.

Ці шість режимів роботи таймера поділяються на три типи: режими 0,4 – однократне виконання функцій; режими 1,5 – робота з перезапуском; режими 2,3 – робота з автозавантаженням.

У режимі автозавантаження регістр CR автоматично переписується в регістр CE після завершення лічби. Сигнал на виході OUT і тільки при наявності на вході GATE рівня логічної 1. Цей режим використовується для створення імпульсних генераторів, що програмуються і генераторів прямокутних імпульсів (меандрів).

3.4.2. Використання таймера в ПЕОМ IBM PC/AT

У комп'ютері IBM PC/XT/AT/PS2 задіяні всі 3 канали таймера (рис.3.15):

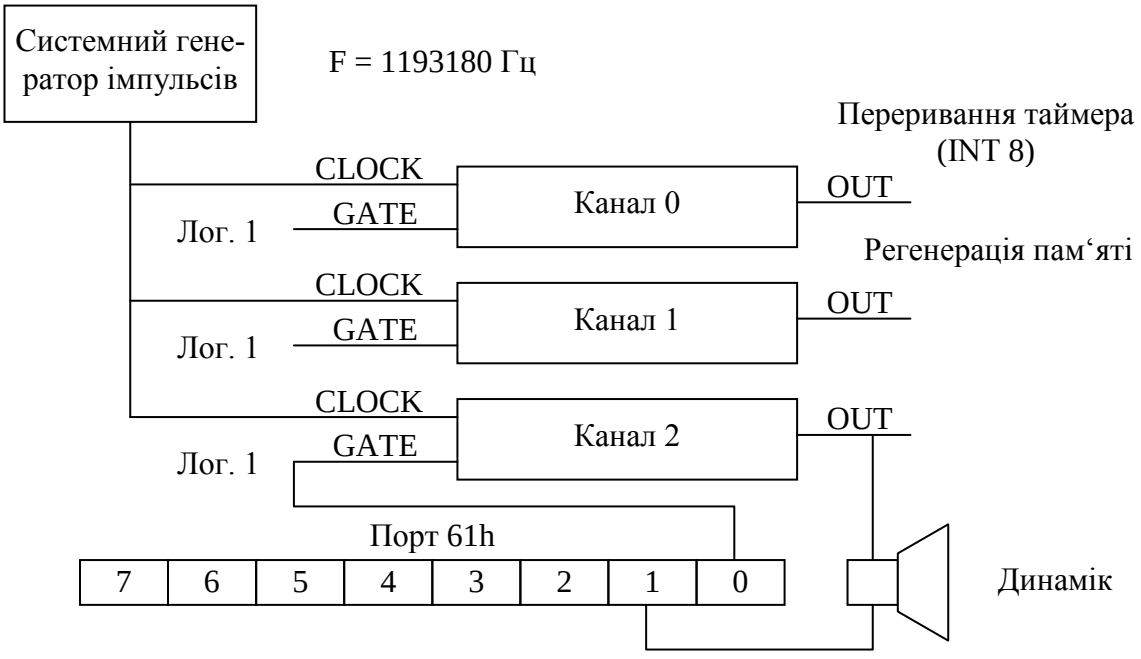

Рисунок. 3.15. Використання каналів таймера в ПЕОМ

Канал 0 використовується в системному годиннику часу доби (не слід плутати з годинником реального часу, що реалізувався на іншій мікросхемі). Цей канал працює в режимі 3 і використовується як генератор імпульсів з час-

тотою приблизно 18.2 Гц (1193180/65536). Саме ці імпульси викликають апаратне переривання INT 8.

Канал 1 використовується для регенерації змісту динамічної пам'яті комп'ютера. Вихід каналу OUT використовується для запиту до каналу прямого доступу DMA, що і виконує поновлення змісту пам'яті.

Канал 2, підключений до динаміка комп'ютера, може бути використаний для генерації різноманітних звуків чи музики, або як генератор випадкових чисел, або як генератор часових затримок.

За програмування таймера відповідають чотири порти введення-виведення за наступними адресами:

40h - канал 0;

41h - канал 1;

42h - канал 2;

43h - керуючий регістр.

3.5 Оброблення зовнішніх переривань

У програмі **P_TASK,** на відміну від програмі **P_INT,** крім переривань від клавіатури та годинника реального часу, обробляються також переривання від системного таймера.

Оброблювач переривання від клавіатури в програмі **P_TASK** відрізняється від відповідного оброблювача в програмі **P_INT**, по-перше, тим, що реалізований в вигляді не процедури, а задачі **keyb_task**, по-друге, тим, що замість виведення не екран значень скан-кодів аналізує стан клавіші "лівий Shift" і формує значення змінної **left_shift**: якщо ця клавіша натиснута, то **left_shift**=1, інакше **left_shift**=0. Крім того по натисненню клавіши D/d викликає задачу **Task_Date.**

Оброблювач переривання від системного таймера, розроблений в вигляді процедури **timer**, виконує такі дії:

скидає біт обслуговування переривання PIC:

```
mov al,20h
out 20h,al
```

аналізує змінну **t_disp** і, якщо вона дорівнює 0, викликає процедуру **dispatch** для переключення задач.

На відміну від системного таймера, який на момент пуску програми **P_TASK** був запрограмований програмой **BIOS** на генерації переривань з періодом 55 мс (див. опис системного таймера в підрозділах 6.4.1 та 6.4.2), годинник реального часу RTC (див підрозділ 5.6) зпочатку не генерує переривання. Для програмування RTC на роботу з генерацією переривань потрібно в його регістр стану за адресою Bh записати код 4ah, а перед початком роботи скинути RTC шляхом читання його регістра стану Ch (див. підрозділ 5.5.1.3).

Оброблювач переривання від годинника реального часу реалізований в вигляді процедури **clock**, виконує такі дії:

скидає біти обслуговування переривання в ведучому та відомому контролерах PIC:

```
mov al,20h
out 20h,al
out 0a0h,al
```

скидає RTC, шляхом читання з його регістра стану з адресою Ch:

```
mov al,0ch
out 70h,al
in al,71h
```

по кожному перериванню зменшує на одиницю (до нуля) значення змінних **time1** та **time2** і додає одиницю до значення змінної **time3**.

Таким чином змінні **time1** та **time2** можут використовуватися в програмі для формування часових затримок з заданою в мілісекундах величиною.

3.6 Текст програми P_TASK

```
{==============Организация многозадачной работы===============}
program p_task;
{--------------Модуль PROT предназначен для поддержки програм,---------------}
{-----------------------работающих в защищенном режиме:-----------------------}
{------------а) содержит необходимые константы, типы переменных,---------}
{---------------переменные, процедуры и функции;-----------------------------------}
{------------б) создает базовые таблицы GDT и IDT-------------------------------}
```

```
uses prot;
label
      end_prot,      { Метки окончания работы в защищенном режиме }
      real;                                  { и возврата в реальный режим }
type  tm=array[0..28] of word;
const
{--------------------Переменные, в которых хранится время в мс,--------------------}
{-------------изменяемое по прерываниям от часов реального времени-------------}
      time1: word=0;                                  { Задача SOUND_TASK }
      time2: word=0;                                  { Задача COLOR_TASK }
      time3: longint=0;                             { Задача DURATION_TASK }
{-------------------------Массив состояния и статуса задач-------------------------}
      state_arr:array[0..1,1..max_n] of byte=
      ((1,1,0,0),                { Состояние задач: 1/0 (выполняется/нет), }
      (0,0,1,0));                 { статус задач:  1/0 (один раз/постоянно) }
      t_disp:byte=0;                                        { Переключение
задач: }
                                  - { 0 - от системного таймера через 55 мс }
                                 { 1 - от часов реального времени через 1 мс }
{---------------------Массив селекторов переключаемых задач----------------------}
      sel_arr:array[1..max_n] of byte=($90,$98,$A0,$A8);
{--------------------------------Массив адресов полей меню----------------------------}
      di_arr:array[1..max_n*2-1] of integer=
      (160,160,160,-462,160,160,160);
{-------------------Массив частот звуков мелодии "Новогодняя"--------------------}
      mf: array[0..28] of word
        =(587,988,988,880,988,785,587,587,587,988,988,880,1174,988,
        1267,988,659,659,1046,1046,988,880,785,587,988,988,880,988,785);
{-------------------Массив длительностей звуков мелодии (в мс)--------------------}
      md: tm
        =(200,200,200,200,200,200,200,200,200,200,200,200,200,400,200,
        200,200,200,200,200,200,200,200,200,200,200,200,200,400);
    s _task:string='Многозадачная работа в защищенном режиме';
    s_pass:string='Введите пароль или нажмите Enter';
    s_date:string='  -  -20  ';
```

```
      s_menu:array[0..5] of string=
            ('       М Е Н Ю  З А Д А Ч        ',
             '                        state  status  ',
             ' TIME_TASK                      ',
            ' COLOR_TASK                     ',
            ' SOUND_TASK                     ',
            ' DURATION_TASK                  ');
      s_dur:array[0..2] of string=
            ('  С момента пуска прошло     ',
            '                    мс          ',
                                 тактов МП');
   var
{----------Массив коэффициентов деления для канала 2 таймера----------}
      k:tm;
{----------Имена переменных, хранящих TSS задач----------}
      keyb_tss,passw_tss,date_tss,time_tss,color_tss,sound_tss,
      duration_tss:t_tss;
{----------Стеки задач----------}
      keyb_stack,passw_stack,date_stack,time_stack,color_stack,
      sound_stack,duration_stack:array[0..255] of byte;
      ldt:t_dt;                                  { LDT задачи PASSW_TASK }
      ofs_new_task,                                          { Смещение }
      sel_new_task:word;                        { и селектор новой задачи }
      di_m:integer;                               { Смещение поля меню }
      left_shift,                          { Состояние клавиши "левый Shift" }
      sw,                                      { Признак переключения задач }
      i,
      n_state,          { Номер колонки меню: 0/1 - состояние/статус задач }
      n_task:byte;                    { Номер задачи (номер строки меню) }
      j:word;
{----------Процедура тепи осуществляет----------}
{----------изменение режима переключения задач:----------}
{----------TIME_TASK(1), COLOR_TASK(2),----------}
{----------SOUND_TASK(3) и DURATION_TASK(4)----------}
   procedure menu;assembler;
```

```
      asm
{------------------------------Вывод на экран изображения меню----------------------}
{---------------------------------(массив строк s_menu)-----------------------------}
            mov ax,100h
            mov bx, 0207h
            push bx              { Сохранить координаты меню для его удаления }
            int 30h
            lea bx,s_menu
            mov cx,6
      @b: push cx
            push bx
            mov ah,4
            mov cl,1bh
            mov si,bx
            int 30h
            add di,104
            pop bx
            add bx,256
            pop cx
            loop @b
            mov n_state,0
            mov n_task,1
            sub di,609
            cmp modeVA,0
            jnz @b01
            mov cl,3eh
            jmp @b02
    @b01:
            mov cl,70h
    @b02:
            mov byte ptr es:[di],cl                { Выделить1-е поле меню }
            mov di_m,di
{---------------Вывод на экран текущего состояния и статуса задач---------------}
            dec di
            lea bx, state_arr
```

```
            lea si,di_arr
             mov cx,8
      @b1:
            mov al,[bx]
            add al,30h
            inc bx
            mov byte ptr es:[di],al
            add di,[si]
            add si,2
            loop @b1
            mov di,di_m
            lea bx,state_arr
            mov cx,max_n
{----------------------На время работы меню задач запретить:----------------------}
{------------------------------работу канала 2 и динамика,-----------------------------}
            in al,61h
            and al,not 3
            out 61h,al
{--------прерывания от системного таймера и часов реального времени-------}
            mov al,1
            out 21h,al
            out 0a1h,al
{----------и разрешить прерывания от клавиатуры для анализа клавиш---------}
        sti
{-----------Организация меню задач, позволяющего с помощью клавиш----------}
{--------------"Стрелка вниз", "Стрелка вверх", "Стрелка влево",----------------}
{-------------"Стрелка вправо", "Space" и "Enter" изменить текущие -----------}
{------------значения состояния и статуса задач в массиве state_arr-----------}
  @menu:
            mov scan,0
{------------------------------Задание цветов для полей меню-------------------------}
            cmp modeVA,0
            jnz @m1
                                                    { в цветном режиме: }
            mov ch,1bh                                  { невыделенного, }
```

```
            mov cl,3eh                          { выделенного; }
            jmp @wait
        @m1:                                    { в черно-белом режиме: }
            mov ch,0f0h                         { невыделенного, }
            mov cl,70h                          { выделенного }
     @wait:                                     { Ожидание нажатия клавиши }
            cmp scan,0
            jz @wait
            cmp scan,50h                        { "Стрелка вниз"? }
            jz @down
            cmp scan,48h                        { "Стрелка вверх"? }
            jz @up
            cmp scan,4dh                        { "Стрелка вправо "? }
            jz @right
            cmp scan,4bh                        { "Стрелка влево "? }
            jz @left
            cmp scan,39h                        { "Space "? }
            jz @space
            cmp scan,1ch                        {
   "Enter "? }
            jz @enter
            cmp scan,1                          { "Escape "? }
            jz @enter
            jmp @wait
  @down:                                        { Перейти к соседнему полю вниз }
            cmp n_task,4
            mov byte ptr es:[di],ch             { Убрать выделение поля }
            jz @d1
            inc n_task
            add di,160
            jmp @d2
        @d1:mov n_task,1
            sub di,480
        @d2: mov byte ptr es:[di],cl            { Выделить новое поле }
            jmp @menu
```

```
 @up:                                  { Перейти к соседнему полю вверх }
          cmp n_task,1
          mov byte ptr es:[di],ch
          jz @u1
          dec n_task
          sub di,160
          jmp @u2
    @u1: mov n_task,4
          add di,480
    @u2: mov byte ptr es:[di],cl
          jmp @menu
@right:                                 { Перейти к соседнему полю вправо }
@left:                                                  { или влево }
          cmp n_state,0
          mov byte ptr es:[di],ch
          jz @r1
          mov n_state,0
          sub di,18
          jmp @r2
    @r1: mov n_state,3
          add di,18
    @r2: mov byte ptr es:[di],cl
          jmp @menu
@space:                          { Изменить значение текущего поля меню }
          dec di
          mov al,byte ptr es:[di]
          cmp al,30h
          jz @s1
          dec al
          jmp @s2
    @s1: inc al
    @s2: mov byte ptr es:[di],al
          inc di
          jmp @menu
@enter:
```

```
{------------Занести измененные значения состояний и статуса задач-----------}
{---------------------------в массив state_arr и выйти из меню-------------------------}
            mov di,di_m
            dec di
            lea bx, state_arr
            lea si,di_arr
            mov cx,8
    @b3:
            mov al,byte ptr es:[di]
            sub al,30h
            mov [bx],al
            inc bx
            add di,[si]
            add si,2
            loop @b3
            mov ax,102h                                  { Очистка меню }
            pop bx
            int 30h
 @end:
            mov al,0
            out 21h,al
            out 0a1h,al
        end;{menu}
{--------------Процедура dispatch осуществляет переключение задач ----------}
{------------------ в соответствии с их состоянием и статусом,----------------------}
{---------------------------заданными в массиве state_arr-------------------------------}

  procedure dispatch;assembler;
      asm
            pusha
            cmp scan,32h                          { Если нажата клавиша M(m) }
            jnz @i
            mov bx,code_sel                                { и пароль введен }
            test byte ptr [offset gdt+bx+6],10h                { правильно - }
            jz @i                          { вызов процедуры menu для изменения }
```

```
            call menu                          { состояния и статуса задач }
   @i:cmp scan,17h                 { Если нажата комбинация клавиш I(i) и }
            jnz @beg
            cmp left_shift,1                                  { левый Shift - }
            jz @d                          { переход на очистку строки экрана }
{------------------Если нажата только клавиша I(i) - вывод на экран----------------}
{-----------------------информации ОС из TSS текущей задачи-----------------------}
            pusha
            db 0fh,0,0ceh                                                  {
STR,SI }
            db 66h
            mov si, word ptr [offset gdt+si+2]
            db 66h,81h,0e6h                          { AND ESI,0FFFFFFh }
            dd 0ffffffh                        { В SI - абслютный адрес TSS }
            db 66h
            mov ax, word ptr [offset gdt+20h+2]
            db 66h,25h                           { AND EAX,0FFFFFFh }
            dd 0ffffffh                        { В AX - абслютный адрес DS }
            db 66h
            sub si,ax                          { В SI - смещение TSS в DS }
            add si,68h                         { В SI - смещение поля sys_inf }
            push si
            mov ax,101h
             mov bx,107h
            int 30h              { Установка маркера (1,7) с очисткой строки }
            mov ah,4               { Вывод на экран содержимого поля sys_inf }
            mov cl,19h
            pop si
            int 30h
            popa
            mov scan,0
            jmp @beg
{--------------------Очистка строки экрана с информацией ОС----------------------}
      @d: mov ax,101h
            mov bx,18h
```

```
            int 30h
            mov scan,0
    @beg:
            mov sw,1                    { Переключение задач разрешено }
            cmp scan,1                  { Нажата клавиша Escape? }
            jz @main                    { Если да - переход на задачу main, }
{-----------------------------иначе - анализ состояния задач-----------------------------}
{----------------------------для выполнения очередной задачи----------------------------}

            lea bx,state_arr            { В BX - адрес массива состояний задач }
            mov cx,max_n                { В CX - число задач }
    @l:
            cmp byte ptr [bx],0         { Имеются активные за-
дачи? }
            jnz @act                    { Если да - переход на поиск очередной задачи }
            inc bx
            loop @l
  @main:                                { Если нет - переход на выполнение задачи main }
            cmp cur_task,0              { Текущая задача - main? }
            jnz @2
            mov sw,0                    { Если да, то переключения задач }
            jmp @pop                    { не производится }
    @2:
            mov cur_task,0
            mov ax,main_sel             { Если нет, то выполняется }
            jmp @new                    { переключение на задачу main }
    @act:
{----------------------------Поиск очередной активной задачи----------------------------}

            lea bx,state_arr
            mov al, cur_task            { В AL - номер текущей задачи }
   @next:                               { Определение номера очередной задачи }
            cmp al,max_n
            jnz @3
            mov al,1
```

```
            jmp @4
        @3: inc al
        @4:                          { Проверка активности очередной задачи }
            mov ah,al
            dec al
            xlat
            cmp al,1
            mov al,ah                                  { Задача активная? }
            jnz @next                { Нет -  проверяется следующая задача }
            cmp al,cur_task                  { Активная задача - новая? }
            jnz @swi                 { Да - переход на переключение задач }
            mov sw,0                  { иначе - продолжает выполняться }
            jmp @pop                  { текущая задача без переключения }
    @swi:
            mov al,ah
            mov cur_task,al             { Определение селектора задачи, }
            lea bx,sel_arr               { которая будет выполняться, }
            dec al
            xlat
            mov ah,0                                   { и занесение его }
    @new: mov sel_new_task,ax                { в переменную new_task_sel }
    @pop:
            popa
            cmp sw,0               { Если переключения задач не требуются - }
            jz @end                      { завершить работу обработчика }
            db 0ffh,2eh             { Переключение на задачу с селектором, }
            dw ofs_new_task           { заданным в sel_new_task, с помощью }
     @end:                          { косвенного межсегментного перехода }
        end;{ dispatch }
{-----------------------Обработчик прерываний от таймера-----------------------}

    procedure timer;assembler;
        asm
            push ax
            mov al,20h                { Сброс бита обслуживания прерывания }
```

```
            out 20h,al                          { от системного таймера }
            pop ax
            cmp t_disp,0
            jnz @end
            call dispatch                       { Вызов диспетчера задач }
        end: db 66h
            iret
        end;{timer}
{----------Задача KEYB_TASK: обработчик прерывания от клавиатуры-----------}

    procedure keyb_task;assembler;
        asm
       @k0:
            mov al,20h                          { Сброс бита обслуживания }
            out 20h,al                          { прерывания от клавиатуры }
            in al,60h                           { Чтение скан-кода клавиши }
            cmp modeVA,1
            jz @save
            mov scan,al
            jmp @scan
{------------------С помощью процедуры save_ekran определяется,------------------}
{-------------------------нажата ли клавиша "Print screen":--------------------------}
{-----------если да - сохранить изображение экрана в переменной screen---------}

            @save:
            db 9ah                              { 16-разрядный вызов }
            dw offset save_ekran                { процедуры save_ekran }
            dw PROT_sel
 {-----------------Анализ скан-кодов нажатых и отжатых клавиш-----------------}

     @scan:
            cmp al,2Ah                  { Нажата клавиша "Левый Shift"? }
            jnz @k1                                         { Если да - }
            mov left_shift,1    { установить признак нажатия этой клавиши }
            jmp @end
```

```
      @k1:
            cmp al,0AAh                    { Отжата клавиша " Левый
Shift"? }
            jnz @k2                                    { Если да - }
            mov left_shift,0          { сбросить признак нажатия клавиши }
            jmp @end
      @k2:
            cmp al,20h                        { Нажата клавиша "D/d"? }
            jnz @end
            db 9ah                          { Если да - вызов задачи task_date }
            dw 0                                       { командой CALL }
            dw date_sel
      @3: mov scan,al                { В scan - скан-код нажатой клавиши }
   @end:
            db 66h
            iret
            jmp @k0-
      end;{keyb_task}
{------------Обработчик прерываний от часов реального времени (RTC)----------}
{----------формирует с помощью переменных time1, time2 задержки в мс;--------}
{----------с помощью переменной time3 определяет прошедшее время в мс;-------}
{-----------------при t_disp=1 используется для переключения задач-----------------}

   procedure clock;assembler;
      asm
            push ax
            mov al,0ch
{---------------------------Сброс часов реального времени ----------------------------}
            mov al,0ch
            out 70h,al
            in al,71h
{--------------------------Сброс контроллеров прерываний:---------------------------}
            mov al,62h
            out 20h,al                                  {  1-го PIC }
            mov al,20h
```

```
            out 0a0h,al                                  { и 2-го PIC }
            pop ax
            cmp time1,0
            jz @1                    { Если значение переменной time не равно 0 }
            dec time1                                    { - уменьшить на 1 }
        @1:cmp time2,0
            jz @2                    { Если значение переменной time2 не равно 0 }
            dec time2                                    { - уменьшить на 1 }
        @2:db 66h
            inc word ptr time3
            cmp t_disp,0
            jz @end
            call dispatch                          { Вызов диспетчера задач }
     @end:
            db 66h
            iret
        end;{clock}
{-------------------------------------Задача PASSW_TASK---------------------------------}
{---------------------Выполняет ввод и анализ пароля пользователя:-----------------}
{--------------если пароль правильный - разрешает работу с меню задач----------}

     procedure passw_task;assembler;
        asm
{--------------Загрузка регистра GS селектором сегмента данных LDT------------}

            db 8eh,2eh                                 { MOV GS,dataLDT_sel }
            dw dataLDT_sel
    @1:
{-------------------------Занесение пароля в сегмент данных LDT---------------------}

            mov cx,3
            mov si,0
            db 65h                                    { Префикс регистра GS }
            mov byte ptr [si],17h                      { Скан-код клавиши "I"}
            inc si
```

```
            db 65h
            mov byte ptr [si],1Fh                    { Скан-код клавиши "S"}
            inc si
            db 65h
            mov byte ptr [si],2Ch                    { Скан-код клавиши "Z"}
{----------------------------------Вывод на экран строки-------------------------------}
{----------------------------"Введите пароль или нажмите Enter"---------------------}

            mov ax,100h                              { Установка маркера: }
            mov bx,0103h                                                  { (1,3) }
            int 30h
            mov ah,4                                 { Вывод строки s_pass на экран: }
            mov cl,1bh
            mov si,offset s_pass
            int $30
{--------------------------Ввод скан-кодов нажатых клавиш--------------------------}
{--------------------------и проверка правильности пароля--------------------------}

            mov cx,3
            mov si,0
   @pass:
            mov scan,0
     @ww:cmp scan,0                  { Задержка до нажатия любой клавиши }
            jz @ww
            test scan,80h
            jnz @ww
            cmp scan,1Ch                             { Нажата клавиша Enter? }
            jz @out
            db 65h
            mov al,[si]
            cmp scan,al                              { Если нет - анализ пароля }
            jnz @l
            db 65h
            inc byte ptr [3]
          @l:
```

```
            db 65h
            inc si
            loop @pass
{---------------------------------Если пароль введен верно -----------------------------}
{----------------------------установка признака "пароль верен"----------------------}

            db 65h
            cmp byte ptr [si],3
            jnz @out
            mov bx,code_sel
            or byte ptr [offset gdt+bx+6],10h
{---------------------------------Стереть строку s_pass---------------------------------}

      @out:
            mov ax,101h
            mov bx,0103h
            int 30h
{------------------Разрешить прерывания от системного таймера-----------------}
{-----------------------------и часов реального времени-----------------------------------}

            mov al,0
            out 21h,al
            out 0a1h,al
            iret
            jmp @1
        end;{passw_task}
{-------------------------------------Задача DATE_TASK-----------------------------------}
{---------------определяет с помощью часов реального времени (RTC)-------------}
{----------и выводит на экран (1,1) текущую дату в формате дд-мм-20гг---------}
{-------------------с сохранением занятой под дату части экрана,-----------------}
{-----------при нажатой клавие "левый Shift" - восстанавливает экран-----------}

   procedure date_task;assembler;
       asm
      @1:
```

```
            cmp left_shift,1          { Если нажата клавиша " левый Shift" - }
            jz @2
{--------------восстановить часть экрана с координатами (29,1),(41,1)------------}
            mov offs_ekr,0
            mov bx,1D01h
            mov dx,2901h
            mov ax,701h
            int 30h
            jmp @4
@2:                                                          { иначе - }
                {------сохранить часть экрана с координатами (29,1),(41,1)-----}
            mov offs_ekr,0
            mov bx,1D01h
            mov dx,2901h
            mov ax,700h
            int 30h
            add offs_ekr,160
{-------------вывести на экран (30,1) текущую дату в формате дд-мм-гг----------}
            mov ax,100h
            mov bx,1E01h
            int 30h                                  { Установка маркера: }
            mov ah,4
            mov cl,1dh
            mov si,offset s_date              { Вывод шаблона даты на экран }
            int $30
            sub di,22
            mov al,7
            out 70h,al
            in al,71h                      { Чтение из RTC в AL - дня месяца }
            mov cl,1dh
            mov dl,al
            mov ax,300h
            int 30h                            { Вывод на экран дня месяца }
            mov ah,2
            mov al,8
```

```
            out 70h,al
            in al,71h                    { Чтение из RTC в AL - номер месяца }
            mov cl,1dh
            mov dl,al
            mov ax,300h
            int 30h                      { Вывод на экран номера месяца }
            add di,4
            mov ah,2
            mov al,9
            out 70h,al                   { Чтение из RTC в AL }
            in al,71h                    { двух последних цифр года }
            mov cl,1dh
            mov dl,al
            mov ax,300h
            int 30h                  Вывод на экран двух последних цифр года }
      @4:
            iret
            jmp @1
      end;{date_task}
{-----------------------------------Задача TIME_TASK-----------------------------------}
{------------осуществляет 32-разрядный вызов процедуры GET_TIME,-----------}
{------------которая определяет с помощью часов реального времени------------}
{------------и выводит на экран текущее время в формате ЧЧ:ММ:СС-----------}

  procedure time_task;assembler;
      asm
      @1:
            db 66h,9ah                   { 32-разрядный вызов }
            dw offset get_time           { процедуры get_time }
            dw 0
            dw PROT_sel
            dw 0
            nop
{-------------------Анализ статуса задачи: если она одноразовая -------------------}
{-----------------------запретить ее дальнейшее выполнение,-----------------------}
```

```
{----------------------изменив состояние с активного на пассивное--------------------}

            lea bx,state_arr
            add bx,max_n
            cmp byte ptr [bx],0
            jz @1
            sub bx,max_n
            mov byte ptr [bx],0
            call dispatch
            jmp @1
        end;{time_task}
{---------------------------------Задача COLOR_TASK---------------------------------}
{---------------перемещает по экранукрасный (черный) прямоугольник-------------}

    procedure color_task;assembler;
        asm
        @1:
            mov dx,es:[di]
            cmp modeVA,0
            jz @11
            mov ax,0f0h                    { прямоугольник красного цвета }
            jmp @12
       @11:
            mov ax,0cdbh                   { прямоугольник черного цвета }
       @12:
            mov es:[di],ax
            mov time2,30
            @t: cmp time2,0
            jnz @t
            mov es:[di],dx
            add di,2
            cmp di,4000
            jl @2
            mov di,0
{--------------------Анализ статуса задачи: если она одноразовая ------------------}
```

```
{------------------------запретить ее дальнейшее выполнение,-----------------------}
{-------------------изменив состояние с активного на пассивное---------------------}
      @2:lea bx,state_arr
           add bx,max_n
           inc bx
           cmp byte ptr [bx],0
           jz @1
           sub bx,max_n
           mov byte ptr [bx],0
           call dispatch
           jmp @1
      end;{color_task}
{----------------------------------Задача SOUND_TASK---------------------------------}
{--------------------------проигрывает заданную мелодию-----------------------------}

   procedure sound_task;assembler;
      asm
      @1:
           mov cx,29                    { В CX - число нот (звуков) мелодии }
           mov si,offset k              { В  SI - адрес массива коэффициентов k }
           mov bx,offset md             { В BX - адрес массива длительностей md }
      @2: lodsw
           out 42h,al                   { Задание коэффициента деления частоты }
           mov al,ah                    { канала 2 системного таймера }
           out 42h,al
           in al,61h
           or al,3
           out 61h,al                   { Разрешение работы канала 2 и динамика }
           mov ax,[bx]                  { В AX - длительность ноты }
           add bx,2
           mov time1,ax                 { В time1 - время задержки звучания
в мс }
      @3:
           cmp time1,0                  { Выполнение задержки звучания }
           jnz @3
```

```
            in al,61h
            and al,not 3
            out 61h,al                  { Запрет работы канала 2 и динамика }
            mov time1,30                { В time1 - время паузы между звуками
в мс }
        @4: cmp time1,0                 { Выполнение задержки-паузы }
            jnz @4
            loop @2                 { Вернуться для проигрывания следующей ноты }
            mov time1,1000                      { В time1 - время паузы }
        @5: cmp time1,0                 { между проигрываниями мелодии }
            jnz @5
{------------------Анализ статуса задачи: если она одноразовая ------------------}
{-------------------------запретить ее дальнейшее выполнение,-------------------------}
{--------------------изменив состояние с активного на пассивное--------------------}
            lea bx,state_arr
            add bx,max_n
            add bx,2
            cmp byte ptr [bx],0
            jz @1
            sub bx,max_n
            mov byte ptr [bx],0
            call dispatch
            jmp @1
      @end:
        end;{sound_task}
{---------------------------------Задача DURATION_TASK-------------------------------}
{---------------определяет и выводит на экран время, прошедшее с----------------}
{-------------момента пуска программы в мс и в тактах роботы МП-------------}

    procedure duration_task;assembler;
        asm
{-------------------------Вывод на экран массива строк s_dur-------------------------}
        @1:
            mov ax,100h
            mov bx,0203h
```

```
        int 30h
        lea bx,s_dur
        mov cx,3
@b: push cx
        push bx
        mov ah,4
        mov cl,09h
        mov si,bx
        int 30h
        add di,106
        pop bx
        add bx,256
        pop cx
        loop @b
        sub di,302
        mov ax,302h          { Вывод на экран значения переменной time3, }
        mov cl,0ah           { содержащей число миллисекунд, прошедших }
        db 66h                                  { с момента пуска программы }
        mov dx, word ptr time3
        int 30h
        add di,124
        db 0fh,31h           { Чтение в EDX:EAX 64 разрядов содержимого }
        db 66h                          { счетчика TSC (Time Stamp Counter) }
        push ax
        db 66h
        push dx
        mov ax,302h
        mov cl,0ah
        db 66h
        pop dx
        int 30h                 { Вывод на экран старших 32 разрядов TSC }
        mov ax,302h
        mov cl,0ah
        db 66h
        pop dx
```

```
            int 30h                    { Вывод на экран младших 32 разрядов TSC }
{------------------Анализ статуса задачи: если она одноразовая, то----------------}
{-------------------------запретить ее дальнейшее выполнение,----------------------}
{--------------------изменив состояние с активного на пассивное--------------------}
            lea bx,state_arr
            add bx,max_n
            add bx,3
            cmp byte ptr [bx],0
            jz @2
            sub bx,max_n
            mov byte ptr [bx],0
        @2:call dispatch
            jmp @1
        end;{duration_task}
{==================ОСНОВНАЯ ПРОГРАММА==================}
   begin
{-----------Задать цветное (0) или черно-белое (1) изображение экрана----------}
{-------------Определение значения сегмента кода cs_PROT процедур,-------------}
        left_shift:=0;                 { Клавиша "левый Shift" не нажата }
{------------------Формирование массива k коэффициентов деления----------------}
{----------------------------канала 2 системного таймера----------------------------}
{----------------------для генерации частотзвуков из массива mf----------------------}

        for i:=0 to 28 do k[i]:=round(1193180/mf[i]);
{================Добавление к базовой таблице GDT==============}

{---------------------------------дескрипторов сегментов-------------------------------}
                                                        { кода программы: }
        init_gdt(code_sel,$ffff,longint(Cseg) shl 4,acc_code,0);
                                                        { таблицы LDT }
        init_gdt(LDT_sel,7,longint(Dseg) shl 4+ofs(ldt),acc_LDT,0);
{-------------------------------------дескрипторов TSS-----------------------------------}
                                                        { Задачи KEYB_TASK: }
        init_gdt(keyb_sel,sizeof(t_tss)-1, longint(Dseg) shl 4+
        ofs(keyb_tss),acc_TSS,0);
```

```
                                                    { Задачи PASSW_TASK: }
    init_gdt(passw_sel,sizeof(t_tss)-1, longint(Dseg) shl 4+
    ofs(passw_tss),acc_TSS,0);
                                                    { Задачи DATE_TASK: }
    init_gdt(date_sel,sizeof(t_tss)-1, longint(Dseg) shl 4+
    ofs(date_tss),acc_TSS,0);
                                                    { Задачи TIME_TASK: }
    init_gdt(time_sel,sizeof(t_tss)-1, longint(Dseg) shl 4+
    ofs(time_tss),acc_TSS,0);
                                                    Задачи COLOR_TASK: }
    init_gdt(color_sel,sizeof(t_tss)-1,longint(Dseg) shl 4+
    ofs(color_tss),acc_TSS,0);
                                                    { Задачи SOUND_TASK: }
    init_gdt(sound_sel,sizeof(t_tss)-1, longint(Dseg) shl 4+
    ofs(sound_tss),acc_TSS,0);
                                                    { Задачи DURATION_TASK: }
    init_tss(duration_tss,code_sel,data_sel,text_sel,
    ofs(duration_task),ofs(duration_stack)+ sizeof(duration_stack));
{==================Формирование сегментов TSS================}

                                                    { Задачи KEYB_TASK: }
    init_tss(keyb_tss,code_sel,data_sel,text_sel,
    ofs(keyb_task),ofs(keyb_stack)+sizeof(keyb_stack));
    keyb_tss.eflags:=0;
                                                    { Задачи PASSW_TASK: }
    init_tss(passw_tss,code_sel,data_sel,text_sel,
    ofs(passw_task), ofs(passw_stack)+sizeof(passw_stack));
    passw_tss.ldt:=LDT_sel;
                                                    { Задачи DATE_TASK: }
    init_tss(date_tss,code_sel,data_sel,text_sel,
    ofs(date_task), ofs(date_stack)+sizeof(date_stack));
                                                    { Задачи TIME_TASK: }
    init_tss(time_tss,code_sel,data_sel,text_sel,
    ofs(time_task),ofs(time_stack)+sizeof(time_stack));
    time_tss.sys_inf:='TIME_TASK выводит на экран текущее время';
```

```
                                        { Задачи COLOR_TASK: }
    init_tss(color_tss,code_sel,data_sel,text_sel,
    ofs(color_task),ofs(color_stack)+sizeof(color_stack));
    color_tss.sys_inf:=
     'COLOR_TASK перемещает по экрану прямоугольник';
                                        { Задачи SOUND_TASK: }
    init_tss(sound_tss,code_sel,data_sel,text_sel,
    ofs(sound_task),ofs(sound_stack)+sizeof(sound_stack));
    sound_tss.sys_inf:='SOUND_TASK проигрывает мелодию';
                                        { Задачи DURATION_TASK: }
    init_tss(duration_tss,code_sel,data_sel,text_sel,
    ofs(duration_task),ofs(duration_stack)+  sizeof(duration_stack));
    duration_tss.sys_inf:=
     'DURATION_TASK выводит время выполнения программы';
{-----------------Формирование дескриптора данных таблицы LDT----------------}
{-----------------------------------для задачи PASSW_TASK--------------------------------}
    asm
        mov bx,offset ldt
        mov word ptr [bx],3                 { Размер сегмента -  4 байта }
        mov word ptr [bx+2],61h*4
        mov byte ptr [bx+4],0                   { Базовый адрес - $61*4 }
        mov byte ptr [bx+5],92h          { Байт доступа сегмента данных }
        mov word ptr [bx+6],0                                { Байт 6 равен 0 }
    end;
{==========Формирование дескрипторов шлюзов таблицы IDT========}
{==========тех обработчиков прерываний, что разработаны==========}
{======================в программе P_TASK=====================}
                                    {-обработчика прерывания от таймера-}
    init_idt($20,ofs(timer),code_sel,acc_int);
                                   {-обработчика прерывания от клавиатуры-}
    init_idt($21,0,keyb_sel,acc_task);
                                        {-обработчика прерывания от RTC-}
    init_idt($28,ofs(clock),code_sel,acc_int);
{------------------Сохранение и формирование данных регистра IDTR----------------}
{-----------------с его загрузкой для работы в защищенном режиме----------------}
```

```
      init_idtr;
{--------------------Программирование контроллеров прерываний--------------------}
{---------------------------для работы в защищенном режиме-----------------------}

      pic(1);
{-------------------------Определение смещений и селекторов----------------------}
{-----------------косвенных межсегментных переходов на метки:------------------}
{------------end_prot (для окончания работы в защищенном режиме);-------------}
{------------ret_real (для возврата в реальный режим)-------------------------------}

      asm
            mov ofs_end_prot,offset end_prot
            mov sel_end_prot,code_sel
            mov ofs_ret_real,offset ret_real
            mov sel_ret_real,cs
      end;
{---------------Сохранение содержимого сегментных регистров и SP:-------------}

      real_cs:=Cseg;                                          { CS, }
      memw[0:4*$60]:=Dseg;                                    { DS, }
      real_ss:=Sseg;                                          { SS, }
      asm mov real_es,es end;                                       {
ES }
      real_sp:=SPtr;                                          { и SP }
{------------------------------Переход в защищенный режим---------------------------}
{-----------------------для МП 80386 и МП последующих моделей--------------------}
{----------------путем установки бита PE в регистре управления CR0------------}
      asm
            db 0fh,20h,0c0h                                   { MOV EAX,CR0 }
            or al,1
            db 0fh,22h,0c0h                                   { MOV CR0,EAX }
{-----------------------Межсегментный переход на метку @prot--------------------}
{-----------------для загрузки регистра CS и сброса очереди команд----------------}
            db 0eah
```

```
            dw offset @prot
            dw code_sel
{==================Работа в защищенном режиме===============}
{---------------------Загрузка сегментных регистров DS, SS и ES--------------------}
{--------------------------соответствующими селекторами-------------------------}

   @prot:
            mov ds,data_sel
            mov ss,stack_sel
            mov es, text_sel
{------------Установка режима работы канала 2 системного таймера-----------}

            mov al,0b6h
            out 43h,al
{-------------------------------Сброс часов реального времени-------------------------}

            mov al,0ch
            out 70h,al
            in al,71h
{--------------------Разрешение генерации часами реального времени----------------}
{-------------------------прерываний IRQ8 (28h) с периодом 1 мс---------------------}

            mov al,0bh
            out 70h,al
            mov al,4ah
            out 71h,al
{----------------------------------Обнуление счетчика TSC-----------------------------}

            db 66h
            xor ax,ax
            db 66h
            xor cx,cx
            mov cl,10h                                  { Номер TSC }
            db 0fh,30h                                  { WRMSR: EAX ->TSC }
{-------------------------------------Очистка экрана-----------------------------------}
```

```
        mov ax,102h
        mov cl,0h
        mov bx,0
        int 30h
{-------------------------Задание 16-разрядного фона символов----------------------}

        mov ax,01h                              { Функция 0, подфункция 1:  }
        int 30h
{----------------------------------Вывод на экран строки-------------------------------}
{--------------------"Многозадачная работа в защищенном режиме"---------------}

        mov ax,100h                             { Установка маркера: }
        mov bx, 0101h                                                { (1,1) }
        int 30h
        mov ah,4                            { Вывод строки s_task на экран: }
        mov cl,1eh
        mov si,offset s_task
        int $30
{--------------------------Загрузка в TR селектора задачи main-----------------------}

        db 0fh,0,1eh                                    { LTR main_sel:  }
        dw v_main_sel
{--------------------Разрешить прерывания только от клавиатуры-----------------}

        mov al,1
        out 21h,al
        mov al,0
        out 0a1h,al
        sti
{-----------------------Вызов задачи passw_task командой CALL----------------------}

        db 9ah
        dw 0
        dw passw_sel
```

```
{--------------------Разрешить прерывания от системного таймера----------------}
{--------------------------------и часов реального времени---------------------------}

            mov al,0
            out 21h,al
            out 0a1h,al
{-----------------------------Ожидание нажатия клавиши Esc-----------------------}
   @wait:
            cmp scan,1
            jnz @wait
{---------------------------------Запрет генерации звука-------------------------------}
            in al,61h
            and al,not 3
            out 61h,al
end_prot:
{=============Подготовка к возврату в реальный режим============}

{--------------------Установка параметров сегментов DS, SS и ES------------------}
{-----------для работы в реальном режиме (Limit=0FFFFh, ED=0, W=1)---------}

            mov ds,real_sel
            mov ss,real_sel
            mov es,real_sel
{----------------------------Запрет маскируемых прерываний-------------------------}

            cli
{----------------------Восстановление атрибутов таблицы IDT----------------------}
{-----------------------------для работы в реальном режиме---------------------------}
            db 0fh,1,1eh                                    { LIDT idtr_r }
            dw idtr_r
{---------------------Возврат в реальный режим по команде MOV-------------------}
{-----------------путем сброса бита PE в регистре управления CR0:----------------}

            db 0fh,20h,0c0h                                 { MOV EAX,CR0 }
            and al,not 1
```

```
            db 0fh,22h,0c0h                                  { MOV CR0,EAX }
{-----------------------Межсегментный переход на метку ret_real------------------}

            db 0ffh,2eh
            dw ofs_ret_real
{==============Работа после возврата в реальный режим=============}

{---------------------------------Восстановление регистров----------------------------}

 ret_real:
            xor ax,ax
            mov ds,ax
            mov ds,[4*60h]                                    { DS, }
            mov ss,real_ss                                    { SS, }
            mov es,real_es                                    { ES }
            mov sp,real_sp                                   { и SP }
        end;

{------------------Перепрограммирование контроллеров прерываний----------------}
{-----------------------------для работы в реальном режиме-------------------------}

        pic(0);
{-------------------------Разрешение аппаратных прерываний-----------------------}

        asm sti end;                                          { маскиру-
емых }
        port[$70]:=$d;                                       { и немаскируемых }
{-----------------------Сброс состояния клавиш-переключателей--------------------}

        mem[$40:$17]:=0;
{--------------Сохранение, если была нажата клавиша "Print Screen",-------------}
{-----------------содержимого экрана на диске в подкаталоге EKRAN-------------}

        save_scr;
    end.
```

3.7. Індивідуальні завдання

Організувати багатозадачну роботу мікропроцесора з наступними параметрами:

1. Число задач – 4; переключення задач – по перериванню від системного таймера; дисципліна обслуговування – послідовне виконання задач з рівними проміжками часу.

2. Число задач – 2 (фонова задача і задача переднього плану; переключення задач – по перериванню від годинника реального часу (CRT); фоновій задачі надається 10 % процесорного часу.

3. Число задач – 3; переключення задач - по перериванню від системного таймера; використовується три рівня пріоритетів: вищий – 60 % процесорного часу, середній – 30 % і нижчий – 10 %; організувати обслуговування задач з циклічним зрушенням пріоритетів.

4. Число задач – 2; переключення задач – по перериванню від годинника реального часу (CRT); за допомогою клавіатури задавати наступні режими обслуговування: пріоритет має перша задача, пріоритет має друга задача, обидві задачі мають рівні часові проміжки.

5. Число задач – 3; переключення задач – по перериванню від системного таймера; здійснити обмін даними між задачами за допомогою поштової скриньки, яка організована в пам’яті.

6. Число задач – 3; переключення задач – по перериванню від годинника реального часу (CRT); переключення задач виконати з використовуванням шлюзів задач.

7. Число задач – 5; переключення задач – по перериванню від системного таймера; дисципліна обслуговування – послідовне виконання задач; за допомогою клавіатури забезпечити можливість вимкнення з роботи будь-якої однієї задачі.

8. Число задач – 6; переключення задач – по перериванню від системного таймера; дисципліна обслуговування – послідовне виконання задач; виконати трасування задач за допомогою переривання INT 1, при цьому виводити на екран зміст регістра TR.

9. Перевірити захист пам’яті при переключенні задач.

10. Перевірити можливість переключення на задачу з більш високим сту-

пенем захисту, ніж рівень привілей поточної програми.

11. Перевірити можливість дозволу звернення до порту введення/виведення за допомогою карти БКВВ в TSS.

12. Перевірити захист пам’яті шляхом звернення до сегмента даних з заданим DPL з програм з різним рівнем привілеїв.

13. Перевірити захист пам’яті у разі звертання до сегмента стеку.

14. Перевірити виклик однієї програми іншою через шлюз виклику (програми мають різний рівень привілеїв).

15. Перевірити можливість переключення з однієї програми на іншу з більш високим рівнем привілеїв, але яка має ознаку підпорядкованості.

4. ЗАХИСТ ПАМ'ЯТІ

Мікропроцесори сімейства x86, починаючи з МП 80286, мають спеціальні апаратні засоби, що забезпечують у захищеному режимі механізм захисту пам'яті. При діючому механізмі захисту при кожному звертанні до пам'яті здійснюється перевірка його коректності ще до того, як почнеться виконуватися цикл обміну даними з пам'яттю. Це не потребує зайвих тактів роботи процесору, так як робиться паралельно формуванню адреси пам'яті. Усі виявлені при перевірці порушення захисту пам'яті реалізуються в вигляді відповідних виключень.

Механізм захисту пам'яті передбачає такі види перевірок:

– перевірка границь сегментів;

– перевірка типів сегментів;

– перевірка рівнів привілеїв сегментів.

4.1. Перевірка границь сегментів

При виконанні перевірки границі сегмента процесор застосовує поле ***Границя сегмента*** та біти ***G*** і ***ED*** дескриптора сегмента (структура дескриптора сегмента розглянута в підрозділі 4.2.1). Коли G=0, тобто встановлена грануляція в байтах, значення ефективної границі задається в 20-розрядному полі "Границя сегмента" і має діапазон від 0 до 0FFFFFh (розмір 1 МБ). Коли G=1 (грануляція в 4 КБ одиницях), значення ефективної границі має діапазон від 0FFFh (розмір 4 КБ) до 0FFFFFFFFh (розмір 4 ГБ).

Зауважимо, що коли встановлене масштабування границі G=1, молодші 12 розрядів зміщення в сегменті не перевіряються. Наприклад, якщо границя дорівнює 0, значення зміщення в сегменті від 0 до FFFh будуть вірними.

Для всіх типів сегментів, окрім сегментів даних, що поширюються донизу, ефективною границею є остання доступна адреса в сегменті, яка на 1 байт менше розміру сегмента.

Процесор кожний раз викликає виключення загального захисту (виключення 13) при спробі доступу:

– до байту, якщо зміщення більш ніж ефективна границя;

– до слову, якщо зміщення більш ніж ефективна границя -1;

– до подвійного слова, якщо зміщення більш ніж ефективна границя -3;

– до четверного слова, якщо зміщення більш ніж ефективна границя -7.

Для сегментів, що поширюються донизу, границя виконує ту ж саму функцію, але інтерпретується по-іншому: вона показує останню адресу в сегменті, до якої не можна звертатися, і має діапазон від ефективної границі +1 до 0FFFFh, коли біт B скинутий, або до 0FFFFFFFFh, коли біт B встановлений. Сегмент з ED = 1 має максимальний розмір, коли ефективна границя дорівнює 0.

Перевірка границь сегментів дозволяє вчасно виявити і відносно просто ідентифікувати такі помилки програмування, як спроби перепису програмного коду або даних в інших сегментах.

Крім границь сегментів пам'яті також перевіряються границі дескрипторних таблиць GDT та IDT на базі значень регістрів GDTR та IDTR (див. підрозділ 4.3).

4.2. Перевірка типів сегментів

Дескриптор сегменту містить інформацію о типах сегментів в двох місцях їх дескрипторів (див. підрозділ 4.2.1):

– в біті ***S***;

– в полі ***TYPE.***

Процесор використовує цю інформацію для виявлення помилок програмування внаслідок спроб некоректного доступу до сегментів чи шлюзів.

Біт S указує, чи є сегмент системним, або він є сегментом коду чи даних. Поле TYPE дає додаткові 4 біта для визначення різних типів системних сегментів, сегментів коду та даних.

Далі приводиться список операцій (він не є вичерпаним), при виконанні яких здійснюється перевірка типів сегментів:

1) При завантаженні селекторів сегментів в сегментні регістри:

– регістр CS може бути завантажений тільки селектором для сегмента коду;

– в сегментні регістри даних DS, ES, FS GS не можуть бути завантажені селектор сегменту коду, що захищений від читання, та селектор системного сегменту;

– тільки селектор сегменту, в який можна писати дані, може бути завантажений в регістр SS.

2) При завантаженні сегментних селекторів в регістри LDTR та TR:

– регістр LDTR може бути завантажений тільки селектором для LDT;

– регістр TR може бути завантажений тільки селектором для дескриптора TSS.

3) При доступі до сегментів, чиї дескриптори вже завантажені в сегментні регістри:

– жодна команда не може здійснити запис до сегмента коду;

– жодна команда не може здійснити запис до сегмента даних, якщо його дескриптор має скинутий біт дозволу запису (W=0);

– жодна команда не може здійснити читання з сегмента коду, якщо його дескриптор має скинутий біт дозволу читання (R=0);

4) При виконанні команди, операнд якої містить селектор сегмента:

– міжсегментні (дальні) команди CALL та JMP (far CALL та far JMP) можуть здійснити доступ тільки до підлеглого сегмента коду, непідлеглого сегмента коду, шлюзу виклику, шлюзу задачі та TSS;

– команда LLDT повинна мати звернення до дескриптора сегмента LDT;

– команда LTR повинна мати звернення до дескриптора сегмента TSS;

– команда LAR повинна мати звернення до дескриптора сегмента LDT, сегмента TSS, шлюзу виклику, шлюзу задачі, сегмента коду або сегмента даних;

– команда LSL повинна мати звернення до дескриптора сегментів LDT, TSS, коду або даних;

– таблиця IDT повинна містити дескриптори переривань, пасток або шлюзів задач.

4.2.1. Перевірка селектора нуль-дескриптора

При спробі завантажити селектор нуль-дескриптора (дескриптора, який містить нулі у всіх полях) в сегментні регістри CS та SS генерується виключення 13. При завантаженні селектора нуль-дескриптора в сегментні регістри DS, ES, FS та GS виключення 13 не виникає. Але воно генерується при спробі доступу до відповідного сегменту.

Завантаження селектора нуль-дескриптора в сегментні регістри даних, які не використовуються в програмі, є корисний засіб запобігання доступу до цих сегментів даних, а також їх захисту.

4.3. Рівні привілеїв

Для захисту інформації, що зберігається в сегментах пам’яті, від несанкціонованого доступу, використовується система привілеїв, що регулює доступ

до того або іншого сегмента в залежності від рівня його захищеності і від ступеня важливості (привілею) запиту.

Мікропроцесори x86 застосовують чотири рівня привілеїв, що задаються числами від 0 до 3. Менший номер відповідає більшому рівню привілею. Ступінь захищеності сегментів також має чотири рівня, що схематично представляються у вигляді вкладених кіл захисту:

– нульове коло захисту складає ядро операційної системи (ОС);

– перше – інші програми ОС;

– друге – системні програми (наприклад, транслятори, редактори та ін.);

– третє – програми користувача.

Процесор використовує рівні привілеїв для запобігання програм або задач, що працюють на менш привілейованих рівнях, від доступу до сегментів з більшими привілеями, за винятком контрольованих ситуацій.

Коли процесор знаходить порушення рівня привілеїв, він генерує виключення загального захисту 13 (#GD).

При виконанні перевірки рівня привілеїв між сегментами коду і сегментами даних процесор використовує такі три типа рівнів привілеїв:

Поточний рівень привілею CPL (Current Privilege Level) задається бітами 0 та 1 селектора в сегментних регістрах CS та SS. Він визначає рівень привілею програми або задачі, що щойно виконується. Звичайно, CPL дорівнює рівню привілею сегмента коду, з якого команди вибираються для виконання.

Процесор змінює CPL, коли здійснює передачу програмного контролю до сегмента коду з іншим рівнем привілею, крім тих випадків, коли контроль передається до підлеглих сегментів коду.

Програми ядра операційної системи мають найбільш високий рівень привілею (CPL = 0), програми користувача - найменший (CPL = 3).

Відзначимо, що всі програми в реальному режимі мають рівень привілею 0.

Рівень привілею дескриптора DPL (Descriptor Privilege Level) задається полем DPL (біти 5 та 6) в байті доступу дескриптора сегмента. Він визначає рівень привілею сегмента чи шлюзу. Найбільш захищений сегмент чи шлюз має значення DPL=0, найменш захищений – DPL = 3.

Рівень привілею запиту RPL (Requested Privilege Level) задається бітами 0 та 1 селектора сегментного регістра при звертанні до сегменту пам’яті. Значення RPL визначає рівень привілею запиту до сегмента. Ініціатором запиту є програма або пристрої, що за допомогою даного селектора звертаються до

пам’яті мікропроцесорної системи. Значення RPL, як правило, встановлюється таке, що дорівнює CPL.

4.4. Доступ до сегмента даних

Щоб отримати доступ до операнда в сегментах даних селектор дескриптора для сегмента даних повинен бути завантажений в сегментні регістри даних DS, ES, FS та GS або в сегментний регістр стеку SS. Сегментні регістри можуть бути завантажені за допомогою команд MOV, POP, LDS, LES, LFS, LGS та LSS. До того, як ці регістри будуть завантажені селекторами, процесор здійснює перевірку рівня привілеїв порівнюючи рівні привілеїв програми чи задачі, що виконується (CPL), RPL селектору сегмента та DPL дескриптора сегмента.

Процесор завантажує селектор до сегментного регістру, якщо виконується правила:

CPL, <= DPL,
RPL <= DPL

тобто програма чи задача повинна мати такий же, або більш високий рівень привілею, ніж рівень привілею сегмента даних. Порушення правила викликає переривання 13 "Порушення загального захисту".

4.5. Доступ до сегмента стека

Звернення до сегмента стека виконується шляхом завантаження селектора в регістр SS. Доступ до сегмента стека буде дозволений за умови

CPL=RPL=DPL

При цьому повинен бути також дозволений запис в сегмент стека:

W=1

У інших випадках викликається переривання 13.

4.6. Доступ до сегмента коду

Щоб передати програмне управління від одного сегмента коду до іншого необхідно завантажити сегментний регістр коду CS. Як частина цього процесу завантаження процесор перевіряє дескриптор сегмента коду призначення. Якщо перевірки виявилися успішними, регістр CS завантажується, програмне управ-

ління передається до нового сегмента коду і програма почне виконуватися з команди, адреса якої знаходиться в регістрі EIP.

Передача програмного управління здійснюється командами JMP, CALL, RET, INT n та IRET, а також механізмами виникнення виключень та переривань.

Виключення, переривання та робота команди IRET були розглянуті в розділі 2 «РОБОТА З ПЕРЕРИВАНННЯМИ У ЗАХИЩЕНОМУ РЕЖИМУ». Тому в цьому розділі буде приділена увага до виконання між сегментних команд JMP, CALL та RET.

Команди JMP та CALL здійснюють звернення до іншого сегмента коду за допомоги одного з чотирьох засобів:

– операнд призначення містить селектор сегмента коду призначення;

– операнд призначення вказує на дескриптор шлюзу, що містить селектор сегмента коду призначення;

– операнд призначення вказує на TSS, який містить селектор сегмента коду призначення;

– операнд призначення вказує на шлюз задачі, який, в свою чергу, вказує на TSS, який містить селектор сегмента коду призначення.

Далі будуть розглянуті тільки два перші засоби. Інформацію про передачу програмного контролю за допомоги шлюзу задач та/або TSS дивиться в розділі 6 «Організація багатозадачної роботи мікропроцесора».

4.6.1. Прямі виклики та переходи до сегментів коду

Короткі (near) форми команд JMP, CALL та RET передають програмне управління в межах поточного сегмента коду, і тому перевірка рівня привілею не робиться. Дальні (far) або між сегментні форми цих команд передають програмне управління до інших сегментів коду, і тому мають бути піддані перевірці. Коли здійснюється передача програмного управління без застосування шлюзу виклику, процесор перевіряє такі чотири типи показників щодо рівнів привілею та типу:

– *CPL* – рівень привілею сегмента коду, що містить процедуру, яка здійснює виклик чи перехід (calling procedure);

– *DPL* дескриптора сегмента коду призначення, що містить процедуру, яка викликається (called procedure);

– *RPL* селектора сегмента коду призначення;

– біт підлеглості **C** в дескрипторі сегмента коду призначення.

Правила, що застосовує процесор для перевірки CPL, DPL та RPL залежать від значення біта C.

4.6.1.1. Доступ до непідлеглих сегментів коду

При доступі до непідлеглих сегментів коду (C=0) CPL процедури, що викликає, повинен дорівнювати DPL процедури, що викликається:

CPL = DPL

У інших випадках генерується виключення 13. (#GP). При цьому повинно також виконуватися

RPL <= CPL

Коли селектор непідлеглого сегмента коду завантажується в регістр CS, рівень привілеїв не змінюється навіть тоді, коли RPL не дорівнює CPL.

4.6.1.2. Доступ до підлеглих сегментів коду

Коли здійснюється доступ до підлеглих сегментів коду (C=1) з рівнем привілею DPL, CPL процедури, що викликає, повинен бути

CPL >= DPL

Процесор генерує виключення 13 тільки тоді, коли CPL <DPL. Зауважимо, що RPL при доступі до підлеглих сегментів коду не перевіряється.

Коли програмне управління передається до підлеглих сегментів коду, CPL не змінюється навіть тоді, коли рівень привілеїв процедури призначення (DPL) менший (чисельно більший), ніж CPL. Це є єдиною ситуацією в роботі процесора, коли CPL відрізняється від DPL сегмента коду, що виконується.

Оскільки CPL не змінюється, не здійснюється і переключення стеку.

Підлеглі сегменти коду застосовуються в програмних модулях, таких, як математичні бібліотеки або оброблювачі виключень. які здійснюють підтримку прикладних програм (програм користувача), але не вимагають доступу до захищених системних ресурсів. Ці модулі є частиною операційної системи чи управляючої програми, але вони можуть бути виконані на чисельно більшому рівні привілею (менш привілейованому рівні).

Утримання CPL при переключенні до підлеглого сегмента коду на рівні CPL того сегмента коду, що зробив виклик, запобігає прикладну програму від доступу до непідлеглих сегментів коду завдяки високому рівню привілею

(DPL) підлеглого сегмента коду, і таким чином запобігає від доступу до більш привілейованих даних.

Більшість сегментів коду є непідлеглими. Для цих сегментів програмне управління може бути передано тільки сегментам коду того ж самого рівня привілеїв, якщо тільки це не зроблено через шлюзи виклику.

4.6.2. Дескриптори шлюзів

Щоб забезпечити доступ до сегментів коду з різним рівнем привілеїв, процесор застосовує набір дескрипторів, які мають назву дескриптори шлюзів. Є чотири типа дескрипторів шлюзів:

– шлюзи виклику (Call Gates);

– шлюзи пасток (Trap Gates);

– шлюзи переривань (Interrupt Gates);

– шлюзи задач (Task Gates).

Шлюзи пасток і переривань є спеціальними типами шлюзу виклику, які використовуються для виклику оброблювачів виключень та переривань (див. розділ 2).

Шлюзи задач застосовуються при переключенні задач і були розглянуті в розділі 6.

4.6.3. Шлюзи виклику

Шлюзи виклику забезпечують контрольовану передачу управління між сегментів коду з різним рівнем привілеїв. Типовим є їх використання в операційній системі чи управляючій програмі, які застосовують механізм захисту за допомоги рівнів привілеїв.

Шлюзи виклику також корисні при передачі програмного управління між 16- та 32-розрядними сегментами коду.

На рис.4.1 приведений формат дескриптора шлюзу виклику:

– *Поле Зміщення в сегменті* визначає точку входу (entry point) в сегменті, куди передається програмне управління (звичайно це перша команда процедури призначення);

7 6	5	4			3 2	1 0
Зміщення в сегменті (31-16)	Байт доступу	0	0	0	Кількість параметрів	Селектор сегмента	Зміщення в сегменті (15-0)

Рисунок. 4.1 – Формат дескриптора шлюзу виклику

– Поле Селектор сегмента вказує сегмент коду, до якого здійснюється доступ;

– Поле Кількість параметрів при переключенні стека вказує кількість параметрів, що копіюються з стеку процедури, що викликає до стеку викликаної процедури;

Формат байта доступу дескриптора шлюзу виклику приведений на рис. 4.2:

Біт P визначає, чи є дійсним шлюз виклику: P = 1 - дійсний; P = 0 – недійсний. Зазначимо, що біт P в дескрипторі шлюзу звичайно встановлюються в 1. Коли він дорівнює 0, то при спробі доступу до шлюзу генерується виключення 11 #NP.

Операційна система може використати цей біт для спеціальних рішень. Наприклад, для підрахунку, скільки разів було звернення для той чи іншої процедури. Для чого спочатку встановлюється P=0, що генерує при доступі до шлюзу виключення 11. Потім оброблювач цього виключення підраховує виклики і встановлює P=1 для подальшого правильного доступу через шлюз;

7	6 5	4	3	2	1	0
P	DPL	S=0	T	1	0	0

Рисунок 4.2. Формат байта доступу шлюзу

Поле DPL задає рівень привілею шлюзу виклику;

Біт T визначає розрядність виклику: 0 – 16-розрядний; 1 - 32-розрядний. Це впливає на розрядність даних, які заносяться в стек при виклику (CS та EIP), та на розрядність параметрів, що передаються при виклику через стек.

Дескриптори шлюзу виклику можуть міститися як в таблиці GDT, так і в таблиці LDT, але не в таблиці IDT.

4.6.4. Доступ до сегментів коду через шлюз виклику

Для доступу до шлюзу виклику застосовується дальній вказівник, як операнд в між сегментних командах CALL та JMP (far CALL та far JMP). Селектор сегмента з цього вказівника визначає шлюз виклику. Зміщення в команді по її формату потрібно, але не використовується при роботі зі шлюзами, не перевіряється процесором і тому може бути завжди встановлено в 0.

Коли процесор дістає доступ до шлюзу виклику він використовує селектор сегмента з шлюзу виклику для знаходження дескриптора сегмента коду призначення (цей дескриптор може бути в GDT або в LDT). Потім процесор бере базову адресу з дескриптора сегмента коду та зміщення з шлюзу виклику для формування лінійної адреси точки входу процедури призначення в сегменті коду.

Чотири рівня привілеїв застосовуються для перевірки правильності передачі програмного управління через шлюз виклику:

– ***CP***L;

– ***RPL*** селектора шлюзу виклику ;

– ***DPL*** дескриптора шлюзу виклику;

– ***DPL*** дескриптора сегмента коду призначення.

Біт C в дескрипторі сегмента коду призначення також перевіряється.

Перевірка правил привілеїв різниться для викликів передачі програмного управління за допомоги команди CALL і команди JMP (таблиця 4.1).

Таблиця 4.1 – Перевірка правил для шлюзу виклику

Команда	Правила перевірка привілеїв
CALL	**CPL <= DPL**$_{\text{шлюзу виклику;}}$ **RPL <= DPL**$_{\text{шлюзу виклику;}}$ **DPL <= CPL**
JMP	**CPL <= DPL**$_{\text{шлюзу виклику;}}$ **RPL <= DPL**$_{\text{шлюзу виклику;}}$ Якщо **C** = 1: **DPL <= CPL** Якщо **C** = 0: **DPL = CPL**

Як це видно з таблиці 4.1, правила доступу до шлюзу виклику ті ж самі, що і до сегмента даних.

Якщо правила доступу до шлюзу не порушені, процесор починає порівнювати DPL сегмента коду призначення з CPL, тобто DPL процедури, що викликає. При цьому перевірка різниться для команди CALL та команди JMP. Тільки команда CALL може передати управління до непідлеглого сегмента коду, що має більш високий рівень привілею ніж процедура, яка викликає.

Якщо виклик зроблений до більш привілейованого непідлеглого сегмента коду, значення CPL зніжується до рівня сегмента коду призначення та здійснюється переключення стеків.

Якщо виклик чи перехід зроблений до підлеглого сегмента коду CPL не змінюється та переключення стеків не робиться.

Шлюзи виклику дозволяють одному сегменту коду бути доступним на різних рівнях привілею. Наприклад, операційна система може мати деякі сервісні програми, які можуть бути застосовані як для ОС, так і для прикладного програмного забезпечення (наприклад, драйвери пристроїв введення-виведення). Шлюзи виклику для цих процедур можуть бути встановлені для доступу програм усіх рівнів привілеїв (від 0 до 3).

Більш привілейовані шлюзи виклику (с DPL 0-1) можуть бути встановлені для інших сервісних програм операційної системи, яки призначені для використання тільки ОС, наприклад, процедури ініціалізації драйверів пристроїв введення-виведення.

4.6.5. Переключення стеку

Коли шлюз виклику застосовується для передачі управління до більш привілейованого непідлеглого сегмента коду, процесор автоматично переключається до стеку сегмента коду призначення.

Це переключення стеку здійснюється для запобігання більш привілейованої процедури від неможливості продовжити роботу через переповнення стеку. Це також запобігає менш привілейованим процедур від втручання (навмисного чи випадкового) до більш привілейованих процедур через загальний стек.

Кожна задача повинна визначити до чотирьох стеків: один для прикладних процедур (DPL=3), та по одному для кожного рівня привілею з 0 до 2, що використовується (якщо застосовуються тільки рівні привілею 0 та 3, то потрібно два стека).

Кожний з цих стеків повинен бути розміщений в окремому сегменті, та

ідентифікований селектором та зміщенням в сегменті (вказівником стеку).

Селектор сегмента та вказівник стеку рівня 3 містяться відповідно в регістрах CS та EIP підчас виконання процедури з DPL=3 і автоматично запам'ятовуються в стеку процедури, що викликається, коли стеки переключаються.

Вказівники стеків рівнів 0-2 містяться в TSS задачі, що виконується. Кожний з цих вказівників містить селектор сегмента та вказівник стеку. Значення цих начальних вказівників можна тільки читати. Процесор не може змінити їх значення підчас роботи задачі. Вони використовуються тільки для створення нових стеків, коли здійснюється виклик до більш привілейованої процедури.

Як тільки процедура викликана, в наступний момент часу створюється новий стек, із застосуванням цих начальних значень.

У TSS не має поля для вказівника рівня 3, тому що процесор не дозволяє процедурам, які працюють на рівні 0, 1 чи 2 передавати програмне управління до процедур рівня привілею 3 за винятком команди повернення (RET).

Операційна система несе відповідальність за створення сегментів стеків і їх дескрипторів та за завантаження начальних значень вказівників цих стеків в TSS. Кожний стек повинен бути дозволений для запису та читання і мати достатній розмір для зберігання таких даних:

– змісту регістрів SS, ESP,CS та EIP процедури, що викликає;

– параметрів та тимчасових змінних, які потрібні процедури, що викликається;

– змісту регістра EFLAGS та коду помилки, якщо робиться неявний виклик оброблювача виключення чи переривання.

Стек повинен мати достатній розмір для зберігання багатьох наборів цих значень, тому що процедури часто викликають інші процедури, а операційна система повинна підтримувати вкладене генерування виключень.

Якщо ОС не застосовує багатозадачну роботу, вона повинна створити по крайній мірі один TSS тільки з метою забезпечення переключення стеків.

Коли виклик процедури через шлюз виклику змінює рівні привілеїв, процесор виконує такі кроки для переключення стеку і початку виконання викликаної процедури на новому рівні привілею:

1. Застосовує DPL сегмента коду призначення (новий CPL) для вибору з TSS вказівника на новий стек (селектора сегмента та вказівника стеку).

2. Читає селектор та вказівник стеку, на який буде зроблено переключення, з поточного TSS. Будь-яке порушення границь, виявлене впродовж читання

селектора сегмента коду, вказівника стеку або дескриптора сегмента коду, призводить до генерації виключення 10 (#TS)

3. Перевіряє дескриптор сегмента стеку на відповідний рівень привілею і тип та генерує виключення 10, якщо виявленні порушень.

4. Тимчасово зберігає поточне значення регістрів SS та ESP.

5. Завантажує селектор сегмента стеку та вказівник стеку для нового стеку в регістри SS та ESP.

6. Заносить в стек тимчасово збережені значення регістрів SS та ESP (для процедури, що здійснила виклик).

7. Копіює параметрі з стеку процедури, що викликала, до нового стеку. Кількість параметрів, що копіюються вказана в полі “Кількість параметрів” шлюзу виклику. Якщо значення цього поля дорівнює 0, ніякого копіювання не відбувається.

8. Заносить вказівник команди (поточне значення регістрів CS та EIP) в новий стек.

9. Завантажує селектор нового сегменту коду та нового вказівника команди до регістрів CS та EIP відповідно і починає виконання викликаної процедури.

Максимальна кількість параметрів, яка може бути передана від процедури, що викликає до процедури, що викликається, згідно до формату шлюзу виклику становить 31. Якщо потрібно передати більш ніж 31 параметр, один з параметрів може бути вказівником до структури даних, або збереженим значенням регістрів SS та ESP, яке може бути застосовано для доступу до параметрів в старому стеку.

На рис. 4.3 показано застосування стеку при виклику процедури з однаковим рівнем привілею (рис. 4.3 а)) та при виклику процедури більш високого рівня привілею через шлюз виклику (рис. 4.3 б)).

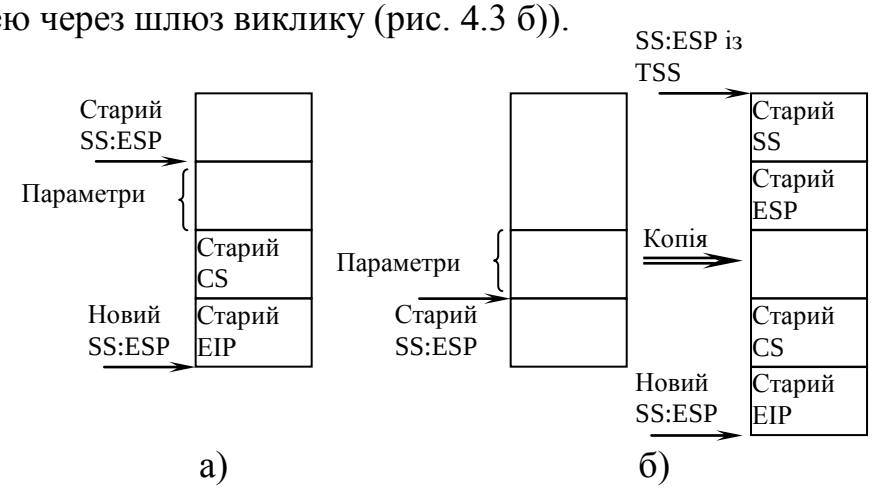

Рисунок 4.3 – Застосування стеку: а) при виклику процедур одного кола привілеїв
б) при виклику процедур різних кіл привілеїв

4.6.5. Повернення з викликаної процедури

Команда RET може бути застосована для виконання близького (в межах одного сегмента коду) повернення, команда far RET для повернення на інший сегмент коду з тим ж самим рівнем привілеїв та команда far RET для повернення на сегмент коду з іншим рівнем привілеїв.

Команда RET призначена для виконання повернення з процедури, яка була викликана командою CALL.

Коли виконується близьке повернення, процесор виконує тільки часткову перевірку. Коли процесор бере з стеку вказівник команди куди передається програмне управління при поверненні, він перевіряє тільки те, чи не перевершена границя сегмента коду при завантаженні EIP.

При далекому поверненні на той же самий рівень привілеїв процесор бере з стеку селектор сегмента коду, до якого буде здійснене повернення, та вказівник команди.

При нормальних умовах ці значення є вірними, тому що були поміщені туди командою CALL. Але процесор виконує перевірку привілеїв для виявлення ситуацій, коли поточна процедура може змінити значення вказівника стеку, але зробити помилку при роботі зі стеком.

Далеке повернення, що потрібує зміну рівня привілею, дозволене тільки тоді, коли виконується повернення до менш привілейованого сегмента коду. Процесор застосовує поле RPL збереженого змісту регістра CS процедури, що виконувала виклик, для визначення того, чи це повернення до сегмента коду з меншими привілеями, як це повинно бути. Якщо RPL чисельно більше (менш привілейоване) ніж CPL, то повернення здійснюється.

Процесор виконує такі кроки при поверненні до процедури, яка здійснювала виклик:

1. Перевіряє поле RPL збереженого змісту регістра CS для виявлення того, чи змінюється рівень привілеїв при поверненні.

2. Завантажує регістри CS та EIP значеннями з стеку викликаної процедури (при цьому виконується перевірка типу та привілеїв дескриптора сегмента коду та RPL селектора сегмента коду).

3. (Якщо команда RET містить операнд числа параметрів, та якщо повернення потребує зміни рівня привілеїв). Додає значення числа параметрів (в байтах, отримане з команди RET) до поточного значення регістра EIP (після вий-

мання з стеку значень CS та EIP). Після цього значення регістра ESP вказує на збережене значення SS та ESP стеку процедури, що викликала (відзначимо, що число байтів в команді RET повинно бути вибрано таким, щоб дорівнювати значенню числа параметрів в шлюзі виклику з урахуванням заданого розміру параметрів).

4. (Якщо повернення потрібує зміни рівня привілеїв). Завантажує регістри SS та ESP значеннями, які були збережені, і переключає стек знову до процедури, що здійснювала виклик. Значення SS та ESP викликаної процедури втрачаються. Будь-які порушення, що виявлені впродовж завантаження селектора сегмента стека, або вказівника стеку, приводять до генерації виключення загального захисту 13. Новий дескриптор сегмента стеку теж перевіряється на порушення типу та привілеїв.

5. (Якщо команда RET має операнд числа параметрів). Додає значення числа параметрів (в байтах, отримане з команди RET) до поточного значення регістра ESP. Отримане значення не перевіряється на переповнення стеку. Якщо значення ESP виходить за межі границі сегмента стеку, цей факт не розпізнається до виконання наступної операції зі стеком.

6. (Якщо повернення потрібує зміни рівня привілеїв). Перевіряється зміст сегментних регістрів DS, ES, FS та GS. Якщо будь-який з цих регістрів посилається до сегмента, чий DPL є меншим за новий CPL (за винятком підлеглих сегментів коду), то сегментний регістр завантажується селектором нуль-дескриптора.

4.7. Привілейовані команди

Деякі з системних команд, які ще звуться привілейованими командами, захищені від застосування з боку прикладних програм. Привілейовані команди управляють системними функціями (такими, як завантаження системних регістрів). Вони можуть бути виконані тільки якщо CPL дорівнює 0, тобто сегментом коду з найвищим рівнем привілею. Якщо одна з цих команд виконується, коли CPL не дорівнює 0, генерується виключення 13 загального захисту. Привілейованими є такі команди:

– ***LGDT*** – завантаження регістра GDT (Load GDT register);
– ***LLDT*** – завантаження регістра LDT (Load LDT register);

– ***LTR*** – завантаження регістра TR (Load TR register);
– ***MOV*** – (регістри управління);
– ***LMSW*** – завантаження слова стану машини (Load Machine State Word);
– ***CLTS*** – чищення біта TS в регістрі CR0;
– ***MOV*** – (регістри налагоджування);
– ***INVD*** – очищення кеш-пам'яті без зворотного запису;
– ***WBINVD*** – очищення кеш-пам'яті зі зворотним записом;
– ***INVLPG*** – очищення TLB;
– ***HLT*** – зупинка процесора ;
– ***RDMSR*** – читання регістрів MSR (MSR - Model Specific Register);
– ***RDPMC*** – читання лічильника моніторингу продуктивності;
– ***RDTSC*** – читання лічильника часових відміток (TSC – Time Stamp Counter).

Біти ***PCE*** та ***TSD*** в регістрі CR4 дозволяють виконувати відповідно команди RDPMC та RDTSC на будь-якому рівні привілеїв.

4.8. Визначення дійсності вказівників

Підчас роботи у захищеному режимі процесор визначає дійсність всіх вказівників для забезпечення захисту між сегментами та підтримки різниці в рівнях привілеїв.

Визначення дійсності вказівників передбачає такі перевірки:

1. Перевірку прав доступу для виявлення , чи відповідає тип сегменту його застосуванню.

2.Перевірку прав читання та запису.

3. Перевірку, чи перевершує зміщення вказівника границю сегмента.

4. Перевірку, чи дозволений користувачеві вказівника доступ до сегмента.

5. Перевірку вирівнювання зміщення.

Процесор автоматично здійснює першу, другу та третю перевірки підчас виконання команди. Програмне забезпечення повинно явно здійснити четверту перевірку за допомоги команди ARPL. П'ята перевірка здійснюється автоматично на рівні привілею 3, якщо встановлено, що ця перевірка повинна відбуватися.

4.8.1. Перевірка прав доступу (команда LAR)

Коли процесор виконує доступ до сегмента застосовуючи далекий (між

сегментний) перехід, він здійснює перевірку прав доступу дескриптора того сегмента, на який вказує далекий вказівник. Ця перевірка виконується для визначення, чи тип та рівень привілеїв (DPL) дескриптора сегмента сумісні з командою, що виконується.

Наприклад, коли виконується далекий виклик у захищеному режимі, тип дескриптора повинен відповідати підлеглому або непідлеглому сегментам коду, шлюзу виклику, шлюзу задачі або TSS. Тоді, якщо виклик здійснюється до непідлеглого сегмента коду, DPL сегмента коду повинен дорівнювати CPL, та RPL селектора сегмента повинен бути менше чи дорівнювати DPL.

Якщо тип чи рівень привілеїв буде знайдені несумісними, генерується відповідне виключення. Щоб уникнути генерацію виключення несумісності, програмне забезпечення може перевіряти права дескриптора сегмента, застосовуючи команду LAR (Load Access Rights). У команді LAR вказується селектор сегмента для дескриптора сегмента, чиї права доступу перевіряються, та регістр призначення.

Команда здійснює такі операції:

1. Перевіряє, що селектор сегмента не вказує на нуль-дескриптор.

2. Перевіряє, що селектор сегмента вказує на дескриптор сегмента, який знаходиться в таблиці GDT чи LDT в межах їх границь.

3. Перевіряє, що дескриптор сегмента є типом дескриптора сегмента коду, даних, LDT та TSS.

4. Якщо сегмент коду є непідлеглим, перевіряється, чи є дескриптор сегмента видимим для CPL (це означає, що CPL та RPL селектора сегмента є меншими чи дорівнюють DPL).

5. Якщо перевірка типу та рівня привілею пройшла успішно, в регістр призначення завантажується друге подвійне слово дескриптора сегмента (яке маскується значенням 00FXFF00h, де X вказує, що відповідні чотири біта є невизнаними) та встановлює прапор ZF в регістрі EFLAGS. Якщо дескриптор сегмента не є видимим для CPL, або має недійсний для команди LAR тип, команда не змінює зміст регістра призначення й скидає прапор ZF.

4.8.2. Перевірка прав читання-запису (команди VERR та VERW)

Коли процесор здійснює доступ до будь-якого сегмента коду чи даних, він перевіряє права читання та запису, які призначені сегменту, для виявлення

того, чи дозволені команди читання або запису, яки мають бути виконані. Програмне забезпечення може перевіряти права читання та запису, застосовуючи команди VERR (Verify for Reading) та VERW (Verify for Writing). У обох цих командах вказані селектори дескрипторів сегментів, що повинні перевірятися.

Команди здійснюють такі операції:

1. Перевіряють, що селектор сегмента не вказує на нуль-дескриптор.

2. Перевіряють, що селектор сегмента вказує на дескриптор сегмента, який знаходиться в таблиці GDT чи LDT в межах їх границь.

3. Перевіряють, що дескриптор сегмента є дескриптором сегмента коду чи даних

4. Якщо сегмент коду є непідлеглим, перевіряється, чи є дескриптор сегмента видимим для CPL.

5. Перевіряють, що сегмент можна читати (для команди VERR) або в нього можна писати (для команди VERW).

Команда VERR встановлює прапор ZF в регістрі EFLAGS, якщо сегмент є видимим і його можна читати.

Команда VERW встановлює прапор ZF в регістрі EFLAGS, якщо сегмент є видимим і в його можна писати (в сегмент коду ніколи не можна пискати).

Прапор ZF скидається, якщо будь-яка з цих перевірок не пройшла.

4.8.3. Перевірка зміщення вказівника (команда LSL)

Коли процесор виконує доступ до будь-якого сегменту, він здійснює перевірку, чи не перевершує зміщення границі сегменту. Програмне забезпечення може виконувати цю перевірку, застосовуючи команду LSL (Load Segment Limit). Як і в команді LAR, в команді LSL вказується селектор сегмента для дескриптора того сегмента, границю якого треба перевірити, та регістр призначення.

Команда здійснює такі операції:

1. Перевіряє, що селектор сегмента не вказує на нуль-дескриптор.

2. Перевіряє, що селектор сегмента вказує на дескриптор сегмента, який знаходиться в таблиці GDT чи LDT в межах їх границь.

3. Перевіряє, що дескриптор сегмента є типом дескриптора сегмента коду, даних, LDT, шлюзу виклику, шлюзу задачі та TSS.

4. Якщо сегмент коду є непідлеглим, перевіряється, чи є дескриптор сег-

мента видимим для CPL.

5. Якщо перевірка типу та рівня привілею пройшла, значення границі сегменту в байтах (скореговане в відповідності із значенням біта G в дескрипторі сегмента) завантажується в регістр призначення та встановлюється прапор ZF в регістрі EFLAGS. Якщо дескриптор сегмента не є видимим для CPL, або має недійсний для команди LAR тип, команда не змінює зміст регістра призначення і скидає прапор ZF.

Після завантаження регістра призначення програмне забезпечення може порівнювати границю сегмента із зміщення вказівника.

4.8.4. Перевірка привілею процедури, що здійснює виклик

Поле рівня привілею запиту селектора сегмента RPL призначене для передачі рівня привілею процедури, що здійснює виклик (CPL), до процедури, що викликається. Викликана процедура тоді застосовує RPL для визначення, чи дозволений їй доступ до сегмента. Можна сказати, що RPL зменшує рівень привілею викликаної процедури до значення RPL.

Процедури операційної системи типово застосовують RPL для запобігання того, щоб менш привілейовані прикладні програми мали доступ до даних, які містяться в більш привілейованих сегментах.

Коли процедура ОС (викликана процедура) отримає селектор сегмента від прикладної програми (процедури, що здійснює виклик), вона встановлює RPL селектора сегмента таким, яким є рівень привілею процедури, що викликає.

Тоді, коли операційна система застосовує селектор сегмента для доступу до відповідного сегмента, процесор здійснює перевірку привілеїв, використовуючи рівень привілею процедури, що здійснила виклик (збережений в тіньовому RPL), замість рівня привілею процедури ОС. Таким чином, RPL, забезпечує те, що операційна система не здійснить доступ до сегмента від імені прикладної програми, якщо тільки ця програма сама не має достатнього для цього рівня привілею.

Нехай, наприклад, прикладна програма володіє селектором сегмента, що вказує на привілейовану структуру даних в сегменті даних рівня привілею 0. Прикладна програма не може здійснити доступ до цього сегмента даних, тому що не має достатнього привілею, але операційна система це може зробити. Таким чином, в спробі доступу до сегмента даних прикладна програма виконує

виклик до процедури ОС і передає селектор сегмента до ОС як параметр через стек. До передачі параметра прикладна програма, яка діє коректно, встановлює RPL селектора сегмента такий самий, як поточний рівень привілею (CPL), який в цьому прикладі дорівнює 3. Якщо процедура ОС спробує здійснити доступ до сегмента даних, застосовуючи цей селектор сегмента, то процесор порівняє CPL, який зараз дорівнює 3, з RPL селектора сегмента та DPL сегмента даних (який дорівнює 0). Оскільки RPL більш ніж DPL, доступ до сегмента даних не буде дозволений.

Таким чином, механізм захисту, який застосовує процесор, заборонив доступ до сегмента даних зі сторони операційної системи, тому що рівень привілею прикладної програми (який представляє поле RPL селектора сегмента), більше, ніж DPL сегмента даних.

Припустимо зараз, що замість встановлення RPL селектора сегмента рівним 3, прикладна програма встановила RPL рівним 0. При цьому ОС вже може здійснити доступ до сегмента даних, тому що CPL та RPL сегмента даних обоє дорівнюють DPL сегмента даних. Оскільки прикладна програма спроможна змінити RPL селектора сегмента на будь-яке значення, вона може потенційно використати процедуру, що працює на більш високому рівні привілеїв для доступу до захищеної структури даних. Ця спроможність понизити RPL селектора сегмента робить можливим обійти механізм захисту процесора.

Оскільки викликана процедура не може покладатися на те, що процедура, яка робить виклик, встановить вірне значення RPL, процедури операційної системи, які отримають селектор сегмента від менш привілейованої процедури, повинні аналізувати RPL селектора сегмента щоб визначити, чи має він достатній рівень привілею.

Для цього запроваджена команда ARPL (Adjust Requested Privilege Level), яка корегує RPL одного селектора сегмента так, щоб він став дорівнювати селектору іншого сегмента.

Коли операційна система отримає селектор сегмента від прикладної програми, вона застосовує команду ARPL для порівняння ARPL селектора сегмента з рівнем привілею прикладної програми (представленим селектором сегменту коду в стеку). Якщо RPL менше ніж рівень привілею програми, команда ARPL змінює RPL селектора сегмента так, щоб він дорівнював рівню привілею прикладної програми.

Таке застосування команди запобігає процедури, які виконуються на чи-

сельно більшому рівні привілею від доступу до сегментів з чисельно меншим рівнем привілею (більш привілейованим) шляхом зменшення рівня привілею RPL селектора сегмента.

Зазначимо, що рівень привілею прикладної програми може бути визнаний шляхом читання поля RPL селектора сегмента коду прикладної програми, чий селектор сегмента міститься в стеку, збережений при виклику до ОС. Операційна система може копіювати селектор сегмента коду з стеку в регістр призначення для використання його як операнд команди ARPL.

4.8.5. Перевірка вирівняння

Коли CPL дорівнює 3, вирівняння посилань до пам’яті може бути перевірено шляхом встановлення біту AM в регістрі управління CR0 та прапору AC в регістрі EFLAGS. Не вирівняні посилання в цьому випадку генерують виключення вирівняння 17 (#AC). Процесор не генерує виключення вирівняння, коли програма виконується на рівнях привілею 0, 1 чи 2.

У таблиці 2.2 розділу 2 “РОБОТА З ПЕРЕРИВАННЯМИ У ЗАХИЩЕНОМУ РЕЖИМУ” наведені вимоги до різних типів даних щодо вирівняння, якщо ця перевірка дозволена.

4.9. Захист на рівні сторінок

Захист на рівні сторінок може бути застосований як сам по собі, так і разом з захистом сегментів. Коли захисти на рівні сторінок використовується разом з плоскої моделлю пам’яті, програми та дані рівня супервізора (операційної системи чи управляючої програми) будуть захищені від програм та даних рівня користувача (прикладних програм).

Коли захист сегментів та сторінок об’єднані, захист читання чи запису на рівні сторінок дозволяє забезпечити захист окремих фрагментів межах сегменту.

При захисті на рівні сторінок, так само як і при захисті на рівні сегментів, кожне посилання до пам’яті перевіряється для виявлення, чи задовольняє воно вимогам захисту.

Усі перевірки робляться до початку циклу пам’яті і будь-які порушення запобігають виконання циклу і результатом є генерування виключення помилки сторінки. Так як перевірка виконується паралельно з перетворенням адреси, це не призводить до зниження продуктивності процесора.

Процесор виконує два типа перевірок на рівні сторінок:

– обмеження доступу (режими супервізора та користувача);

– типу сторінок (тільки для читання чи для читання та запису).

Порушення будь-якого з цих типів перевірок викликає генерацію виключення помилки сторінки 14 (#PF) (опис виключень дивиться в підрозділі 2.1).

4.9.1. Біти захисту сторінок

Інформація щодо захисту сторінок міститься в двох бітах каталогу сторінок та таблиці сторінок:

– в біті “читання/запис” (біт 1);

– в біті “супервізор/користувач” (біт 2).

Перевірка захисту здійснюється для першого та другого рівня таблиць сторінок (тобто для каталогу сторінок і таблиць сторінок).

4.9.2. Обмеження доступу до сторінок

Механізм захисту рівня сторінок дозволяє обмежити доступ до сторінок, що базується на двох рівнях привілеїв:

режимі супервізора (біт U/S = 0 – найбільш привілейований) для операційній системи, управляючої програми, іншого системного програмного забезпечення (такого, як драйвери пристроїв) та захищених системних даних (таких, як дескрипторні таблиці);

режимі супервізора (біт U/S = 1 – найменш привілейований) для прикладних програм та даних.

Рівні привілеїв сегментів відображаються на рівні привілеїв сторінок таким чином: якщо процесор працює, маючи CPL, який дорівнює 0, 1 чи 2, то він знаходиться в режимі супервізора, якщо ж процесор працює з CPL=3, він знаходиться в режимі користувача.

Коли процесор працює в режимі супервізора, він має доступ до всіх сторінок (якщо в регістрі управління CR0 біт WP=1, то дозвіл супервізора модифікується, як це показано в підрозділі). Коли процесор працює в режимі користувача він має доступ тільки до сторінок рівня користувача. Зазначимо, що при застосуванні механізму захисту на рівні сторінок, сегменти коду та даних повинні бути встановлені, по крайній мірі, в двох сегментно-орієнтованих рівнях привілею: рівню 0 для програм і сегментів супервізора та рівня 3 для програм і

сегментів користувача (в цій моделі стеки поміщаються в сегменти даних).

Щоб мінімізувати використання сегментів, може бути застосована плоска модель пам'яті. При цьому всі сегменти - програм та даних, користувача та супервізора розташовані, починаючи з адреси 0 в лінійному адресному просторі, перекриваючи один одного. При цій організації пам'яті програми операційної системи (які працюють на рівні супервізора) та прикладні програми (які працюють на рівні користувача) можуть виконуватися так, якби сегментів взагалі не існує. Захист між програмами ОС та користувача забезпечується процесором за допомоги механізму захисту на рівні сторінок.

4.9.3. Тип сторінок

Механізм захисту на рівні сторінок признає два типа сторінок:

– з доступом тільки для читання (біт R/W = 0);

– з доступом для читання і для запису (біт R/W = 1).

Коли процесор працює в режимі супервізора та біт WP в регістрі управління CR0 є скинутим (стан після ініціалізації по скиданню процесора), всі сторінки є доступними як для читання, так і для запису (заборона запису ігнорується).

Коли процесор працює в режимі користувача, він може писати тільки в ті сторінки режиму користувача, які доступні для читання та запису. Усі сторінки режиму користувача є доступними для читання.

Сторінки режиму супервізора є недоступними ні для читання, ні для запису, якщо процесор працює в режимі користувача.

Виключення помилки сторінки генерується при будь-якій спробі порушити ці правила захисту.

Встановлення біта WP в регістрі CR0 в 1 дозволяє процесорам, починаючи з i486, захистити сторінки рівня користувача, які призначені тільки для читання, від запису при доступі до них в режимі супервізора. Ця особливість корисна при впровадженні стратегії "копіювати та записати" ("copy-on-write"), яка використовується деякими операційними системами (такими, як UNIX) для створення задач. Коли створюється нова задача, є можливість копіювати весь адресний простір задачі-предка. Це дає задачі-нащадку повний набір сегментів та сторінок задачі-предка, що дублюються .

Альтернативна стратегія зберігає адресний простір та час шляхом накладення сегментів і сторінок задачі-нащадка на ті ж самі сегменти і сторінки, яки

використовуються задачею-предком. Окреме копіювання сторінок здійснюється тільки тоді, коли одна із задач робить запис до сторінки.

Застосовуючи біт WP та роблячи сторінки, яки використовуються сумісно, як такі, що доступні тільки для читання, супервізор може виявити спроби писати до цих сторінок рівня користувача, в той же час може їх копіювати.

4.9.4. Комбінування захисту двох рівнів таблиць сторінок

Для будь-якої однієї сторінки атрибути захисту в елементі каталогу сторінок (перший рівень таблиць сторінок) можуть різнитися від атрибутів захисту в елементах таблиці сторінок (другий рівень таблиць сторінок).

Процесор перевіряє захист сторінок як в каталогах сторінок, так і в таблицях сторінок.

У таблиці 4.2 показано, як забезпечується захист сторінок при можливих комбінаціях атрибутів захисту при умові, що біт WP скинутий.

Таблиця 4.2 – Комбінований захист каталогу сторінок та таблиці сторінок

Елемент каталогу сторінок		Елемент таблиці сторінок		Комбінований ефект	
Рівень привілею	Тип доступу	Рівень привілею	Тип доступу	Рівень привілею	Тип доступу
К*	Тільки Ч*	К	Тільки Ч	К	Тільки Ч
К	Тільки Ч	К	Ч/З	К	Тільки Ч
К	Ч/З*	К	Тільки Ч	К	Тільки Ч
К	Ч/З	К	Ч/З	К	Ч/З
К	Тільки Ч	С	Тільки Ч	С	Ч/З**
К	Тільки Ч	С	Ч/З	С	Ч/З**
К	Ч/З	С	Тільки Ч	С	Ч/З**
К	Ч/З	С	Ч/З	С	Ч/З
С*	Тільки Ч	К	Тільки Ч	С	Ч/З**
С	Тільки Ч	К	Ч/З	С	Ч/З**
С	Ч/З	К	Тільки Ч	С	Ч/З**
С	Ч/З	К	Ч/З	С	Ч/З
С	Тільки Ч	С	Тільки Ч	С	Ч/З**
С	Тільки Ч	С	Ч/З	С	Ч/З**
С	Ч/З	С	Тільки Ч	С	Ч/З**
С	Ч/З	С	Ч/З	С	Ч/З

Примітка

*: У таблиці зроблені такі скорочення: К – користувач; С – супервізор; Ч – читання; З – запис.

**: Якщо в регістрі CR0 біт WP встановлений, тип доступу визначається бітами R/W елементів каталогу сторінок та таблиці сторінок.

4.9.5. Перекладання захисту на рівень сторінок

До типів доступу до пам’яті, які перевіряються таким чином, ніби доступ здійснюється на рівні привілею 0 незалежно від CPL процесу, що зараз виконується, відносяться:

– доступ до дескрипторів сегментів в таблицях GDT, LDT та IDT;

– доступ до стеку внутрішнього рівня привілею впродовж виклику зі зміною рівня привілею, або виклику в оброблювачі виключення чи переривання, коли трапляється зміна рівня привілею.

4.10. Комбінування захисту сторінок та сегментів

Коли здійснюється сторінкова організація пам’яті, процесор перевіряє спочатку захист сегментів, потім захист сторінок. Якщо процесор виявляє порушення захисту як на рівні сегментів, так і на рівні сторінок, доступ до пам’яті не здійснюється та генерується виключення. Якщо виявлено порушення сегментації, виключення порушення сторінок не генерується.

Захист на рівні сторінок не може відмінити захисту рівня сегментів. Наприклад, сегмент коду по визначенню є таким, в який заборонено робити запис. Якщо цей сегмент коду розбити на сторінки з встановленим значенням біту R/W, то це не зробить сторінку такою, в яку можна робити запис. Спроба писати в сторінки будуть заблоковані захистом рівня сегментів.

Захист рівня сторінок може бути застосований для розширення можливостей захисту рівня сегментів. Наприклад, якщо великий сегмент даних з дозволом читання-запису розбивається на сторінки, то механізм захисту рівня сторінок може бути використаний для захисту від запису окремих сторінок.

4.11. Доступ до зовнішніх пристроїв

Реалізація звернення до зовнішніх пристроїв за допомоги команд IN, OUT, INS та OUTS у захищеному режимі здійснюється з урахуванням рівня привілею програми, тобто CPL, та поля IOPL (Input/Output Privilege Level) регістра EFLAGS. Введення та виведення даних при роботі зі зовнішніми пристроями здійснюється тільки при виконанні умови

CPL <= IOPL.

Порушення цієї умови викликає переривання 13.

Але якщо ця умова не виконується, то є можливість отримати доступ до того чи іншого окремого порту за допомоги відповідних бітів бітової карти введення/виведення в TSS (див. підрозділ 6.1.1). При цьому номер біта в БКВВ визначає номер однобайтного порту. Його нульове значення дозволяє доступ до порту, а одиничне (замасковане) - не дозволяє.

Підкреслимо, що БКВВ перевіряється тільки при порушенні умови CPL <= IOPL. Якщо ця умова виконується, програмі дозволяється доступ до будь-якого порту незалежно від стану бітів БКВВ.

4.12. Опис програми P_USER

Програма **P_USER** розроблена як приклад створювання програм з двома рівнями привілеїв, що є типовим для сучасних операційних систем – процедури та структури даних, що складають їх основу, мають рівень привілеїв 0, процедури користувачів, які виконуються під управлянням ОС, мають рівень привілеїв 3.

У програмі **P_USER** розглядаються питання виклику програми користувача та перевірка можливостей цієї програми відносно доступу до сегментів даних та її взаємодії з процедурами різних рівнів привілею.

Для створення процедур рівня привілею користувача (DPL = 3) та підлеглих процедур найвищого рівня привілею (C=1; DPL = 0) були сформовані сегменти коду у вигляді модулів відповідно **USER та CONF**.

Як повідомлялося вище, єдиним способом виклику програмою з вищим рівнем привілею програми користувача, яка має найнижчий рівень привілею, є представлення цієї програми в вигляді задачі.

Питання розроблення задач та робота з ними у захищеному режимі були розглянути в розділі 3 ”ОРГАНІЗАЦІЯ БАГАТОЗАДАЧНОЇ РОБОТИ МІКРОПРОЦЕСОРА”. Тому зупинимось лише на тих особливостях, які має програма **P_USER** відносно програми **P_TASK**:

– формування дескрипторів сегментів, шлюзу виклику та полів TSS задачі користувача здійснюється з урахуванням рівня привілею 3;

– формується локальна дескрипторна таблиця.

4.12.1. Формування дескрипторів сегментів, шлюзу виклику та полів TSS задачі користувача

Оскільки програма користувача має рівень привілею 3, необхідно сформувати дескриптор сегмента коду модуля **USER** з DPL = 3:

init_gdt(USER_sel,$ffff,longint(cs_USER) shl 4, acc_code+$60,0).

Для того, щоб користувач міг працювати зі своїми даними та відео-пам'яттю для виведення повідомлень слід розробити дескриптори відео-пам'яті -

init_gdt(text_sel,4000-1,$b8000,acc_data+$60,0) ,

та даних користувача -

init_gdt(data3_sel,$ffff,longint(Dseg) shl 4, acc_data+$60,0),

які також мають рівень привілею 3.

Оскільки транслятор з мови Паскаль дані всіх модулів, що підключаються до програми, об'єднує в одному сегменті даних, то доводиться сегмент даних користувача (DPL = 3) накласти на сегмент даних основної програми DPL = 0.

Крім цього, щоб задача користувача могла виконати виклик процедури більш високого рівня привілею - **get_time** модуля **PROT**, здійснюється формування дескриптора шлюзу виклику:

```
mov word ptr [offset gdt+gate_sel],offset get_time    { поле зміщення 15-0}
mov word ptr [offset gdt+gate_sel+2],PROT_sel         { поле селектору }
mov byte ptr [offset gdt+gate_sel+4],0                { поле числа параметрів }
mov byte ptr [offset gdt+gate_sel+5],acc_call+$60     { поле байту доступу }
mov word ptr [offset gdt+gate_sel+6],0                { поле зміщення 31-16 }
```

Програма користувача представлена у вигляді задачі USER_CHECK_TASK, яка має рівень привілею користувача (DPL=3) і тому має такі особливості формування окремих полів своготSS:

```
user_tss.ss:=data3_sel+3;        {поля сегмента стеку з DPL=3, RPL=3 }
user_tss.ds:=data3_sel+3;        { поля сегмента даних з DPL=3, RPL=3 }
user_tss.es:=video_sel+3;        { поля сегмента відео-пам'яті з RPL=3 }
user_tss.cs:=code3_sel+3;        { поля сегмента коду з DPL=3, CPL=3 }
user_tss.ss0:=data_sel;          { поля адреси стеку шлюзу для програм }
user_tss.sp0:=ofs(stack_0)+sizeof(stack_0);      { 0-го рівня привілею }.
```

4.12.2. Формування локальної дескрипторної таблиці

На відміну від програми **P_TASK**, де жодна задача не має сегментів, дескриптори яких задані в локальній дескрипторній таблиці LDT, в програмі **P_USER**, розроблена таблиця LDT, яка містить дескриптор сегмента даних користувача

```
ldt.lim_l  :=0;                 { Размер сегмента пользователя - 1 байт }
ldt.base_l :=base_adr;          {базова адреса сегмента даних користувача }
ldt.base_h :=base_adr shr 16;
ldt.acc :=acc_data+$60;
ldt.lim_h  :=0;
ldt.base_hh:=0;
```

та її дескриптор, який зберігається у GDT:

```
init_gdt(LDT_sel,7,longint(Dseg) shl 4+ofs(ldt),acc_LDT,0);
```

4.13. Опис модуля USER

Модуль **USER** , заданий як сегмент коду рівня привілею користувача (DPL=3), містить задачу USER_CHECK_TASK (процедуру **user_check_task**) та процедуру **out_str3**.

Процедура **user_check_task** виводить на екран повідомлення ”**Працює програма користувача (CPL = 3)**” (рис. 4.3) та виконує перевірки доступу до сегментів пам’яті та до портів введення-виведення.

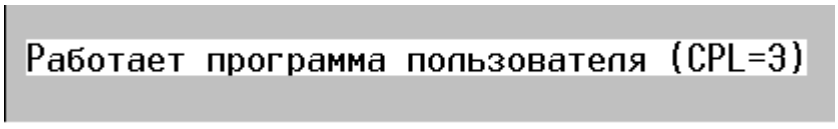

Рисунок 4.3 – Повідомлення процедури **user_check_task**

4.13.1. Доступ до сегментів даних

Якщо дозвіл для проведення перевірки доступу до сегментів даних встановлений –

test_data:byte=1;,

то, по-перше, здійснюється перевірка доступу до сегмента даних користувача, заданого в таблиці LDT. Для цього селектор зі значенням 4, що вказує на дескриптор з індексом 0 в таблиці LDT, завантажується в сегментний регістр DS (поточне значення DS зберігається в стеку):

mov ax,4
push ds
mov ds,ax.

На екран за допомоги функцій 1 та 4 переривання INT 30h виводиться повідомлення "**Сегмент даних користувача (LDT) містить:** " (рис. 4.4). Потім дані читаються з цього сегмента по зміщенню 0

mov dl, [0]
pop ds:

та виводяться на екран за допомоги функції 3 переривання INT 30h (рис. 4.4).

По-друге, здійснюється перевірка доступу до сегмента даних користувача. Для цього дані читаються по зміщенню 2 поточного значення сегментного регістру даних DS:

mov dl,[2].

На екран за допомоги функцій 3 та 4 переривання INT 30h виводиться відповідне повідомлення та прочитані дані (рис. 4.4).)

По-третє, здійснюється перевірка доступу до сегмента даних з DPL – 0. Для цього робиться спроба прочитати дані з сегмента даних програми **P_USER**:

mov ds,data_sel
mov al,[0].

та вивести їх на екран. Це призводить до виникнення переривання 13 (0Dh). Стандартний оброблювач цього переривання виводить на екран про це повідомлення з вказівкою про селектор та зміщення команди, при виконанні якої виникло переривання (рис. 4.4),та закінчює виконання програми.

Селектор зі значенням 63h вказує на процедуру модуля **USER**.

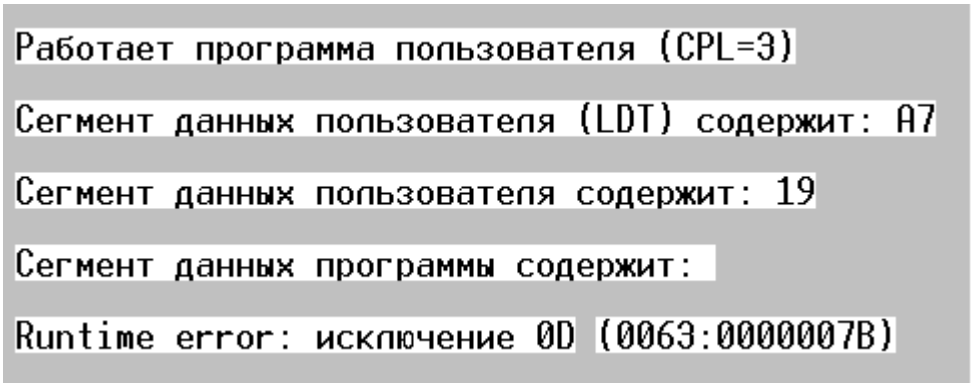

Рисунок 4.4 – Перевірка доступу до сегментів даних

4.13.2. Доступ до сегментів коду

Якщо дозвіл для проведення перевірки доступу до сегментів коду встановлений –

test_code:byte=1;,

то, по-перше, здійснюється перевірка виклику процедури рівня користувача. Для цього викликається процедура **out_str3** (DPL = 3).

```
db 9ah
dw offset out_str3
dw USER_sel,
```

яка виводить на екран повідомлення ”**Виклик процедури з DPL=3**” (рис. 4.5).

По-друге, викликається підлегла процедура **out_str** з модуля CONF:

```
db 0ffh,1eh
dw ofs_call,
```

яка виводить на екран повідомлення ”Викликана підлегла процедура з C =1 та D = 0”.

По-третє, на екран виводиться повідомлення ”**Виклик процедури get_time з DPL-0 через шлюз:**” та здійснюється цей виклик програмою користувача:

```
mov bx,2F09h
db 9ah
dw 0
```

```
dw gate_sel
```

Процедура **get_time** , застосовуючи годинник реального часу, виводить на екран поточне значення часу (рис. 4.5).

По-четверте, здійснюється перевірка за допомоги команди LAR можливості виклику процедури **get_time**, яка має вищий рівень привілею, програмою користувача (див. опис команди LAR в підрозділі 4.8.1):

```
mov cx,PROT_sel      { В CX – селектор процедури, що викликається }
db 0fh,2,0c1h                                          { LAR AX,CX }
jz @call                                  { Якщо ZF=1 – виклик процедури, }
mov ax,100h                                            { якщо ні  - }
mov bx,0109h                              { виведення рядку s34 на екран }
int 30h
mov ah,4
mov cl,1ch
mov si,offset s34
int 30h
```

Так як команда LAR заборонила виклик процедури, на екран виводиться повідомлення ”**Виклик процедури з заборонений командою LAR**” (рис. 4.5).

По-п’яте на екран виводиться повідомлення ” **Виклик процедури з DPL-0**” та безпосереднє викликається процедура **get_time**:

```
@call:
        db 9ah
        dw offset get_time
        dw PROT_sel.
```

Це призводить до виникнення переривання 13 (останній рядок рисунка 4.5).

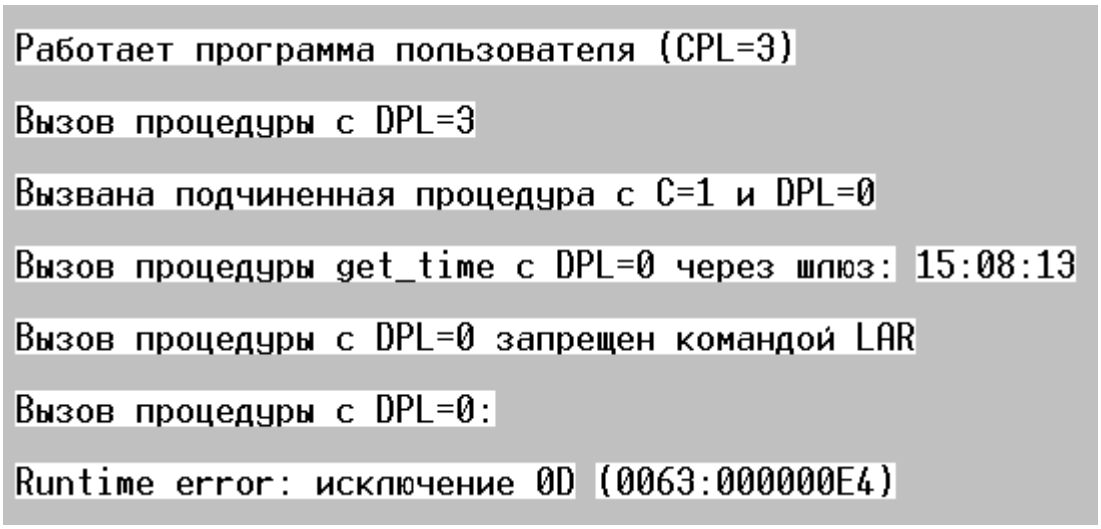

Рисунок 4.5 – Перевірка доступу до сегментів коду

4.13.3. Доступ до портів введення-виведення

Якщо дозвіл для проведення перевірки доступу до портів введення-виведення встановлений –

test_port:byte=1;,

то, по-перше, перевіряється можливість читання даних з порту, доступ до якого є дозволеним. Для перевірки вибраний порт 60h – регістр даних клавіатури:

addr_port:word=$60;.

На екран виводяться повідомлення “**Читання scan-коду**:” та “**Доступ до порту 60h дозволений**” (рис. 4.6).

Дозвіл чи заборона доступу до будь-якого порту визначається відповідним розрядом бітової карти введення-виведення (див. підрозділ 4.11). Оскільки при формуванні сегмента TSS задачі USER_CHECK_TASK доступ до всіх портів в діапазоні 0-255 був заборонений шляхом завдання значення 1 для всіх бітів молодших 32 байтів поля БКВВ -

for i:=0 to 31 do
user_check_tss.BKVV[i]:=$FF;,

то треба дозволити доступ до порту 60h, встановивши значення 0 для біту з номером 96 (60h) поля БКВВ сегмента TSS. Для цього виконуються такі дії:

– а) визначається залишок від ділення номера порту на 8

mov cx,addr_port

and cl,7;

– б) в регістрі DL формується байт який містить значення 1 в біті, номер якого дорівнює значенню залишку

mov dl,1;

@l:

cmp cl,0

jz @in

shl dl,1

dec cl

jmp @l

@in:

– в**) з**дійснюється інвертування регістру DL

not dl *{ DL містить 0 в біті з номером залишку };*

– г) знаходиться частка від ділення адреси порту на 8

mov bx,addr_port

shr bx,3;

– д) встановлюється значення 0 біта поля БКВВ з номером 60h

and byte ptr [offset user_check_tss+360+bx],dl.

Далі здійснюється читання scan-коду з порту 60h

mov dx,addr_port

in al, dx

та виведення його значення на екран (рис. 4.6). Отримане значення 9Dh відповідає scan-коду відтиснення клавіші “Ctrl”. Це правильний результат, оскільки запуск програми на виконання був здійснений по натисненню комбінації клавіш “Ctrl-F9”.

По-друге, перевіряється можливість читання даних з порту, доступ до якого є заборонений. Для цього забороняється доступ до порту 60h шляхом встановлення значення 1 в відповіднім біті поля БКВВ

or byte ptr [offset user_check_tss+360+bx],dl

та здійснюється повторне читання з порту 60h. На цей раз замість виведення значення порту на екрані буде виведене повідомлення про виникнення переривання 13 (рис. 4.6).

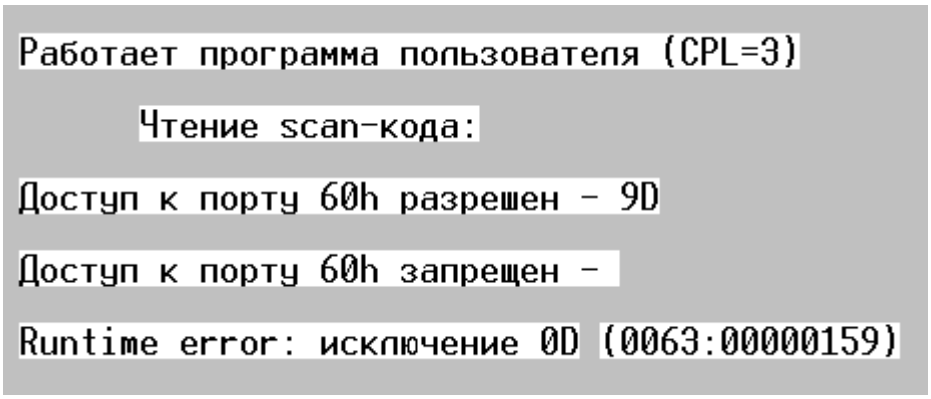

Рисунок 4.6 – Перевірка доступу до пристроїв введення-виведення

4.14. Текст программы P_USER

```
{============Организация работы пользователя (DPL=3)============}
 program p_user;
{---------------Модуль PROT предназначен для поддержки програм,---------------}
{-------------------------работающих в защищенном режиме:-------------------------}
{-------------а) содержит необходимые константы, типы переменных,----------}
{----------------переменные, процедуры и функции;-----------------------------------}
{-------------б) создает базовые таблицы GDT и IDT-------------------------------}

    uses prot;crt,
{------------Модуль USER содержит задачу CHECK_USER_TASK и------------}
{---------процедуру out_str3 уровня привилегий пользователя (DPL=3)----------}
    user,

{--------------------Модуль CONF содержит подчиненную (C=1)--------------------}
{-----------------------процедуру out_str уровня привилегий 0---------------------------}
    conf;
    label
          end_prot,        { Метки окончания работы в защищенном режиме }
          real;                                  { и возврата в реальный режим }
    const
{---------------Данные сегмента данных задачи USER_CHECK_TASK-------------}
          segm_data:byte=$A2;
    var
          base_adr:longint;
          ldt:t_gdt;                                          { Таблица LDT }
```

```
        i:byte;
{----------------Значение сегмента кода в реальном режиме модулей---------------}
        cs_USER,                                                    { USER }
        cs_CONF:word;                                              { и CONF }
{---------------------------------TSS задачи user_check_tss-----------------------------}
        user_check_tss: t_tss;
{-----------------------------------------Стеки-------------------------------------------}

        user_check_stack,                          { задачи USER_CHECK_TASK }
        stack_0:array[0..255] of byte;                          { шлюза уровня 0 }
{=================ОСНОВНАЯ ПРОГРАММА==================}

    begin
{------------Задать цветное (0) или черно-белое (1) изображение экрана----------}
        modeVA:=0;
{-------------Определение значения сегментов кода реального режима------------}

        cs_USER:=Seg(user_check_task);                          { модуля USER }
        cs_CONF:=Seg(out_str);                                  { модуля CONF }
{---------------------------Задание значений для выполнения---------------------------}
{---------------косвенного межсегментного вызова процедуры out_str-------------}

        ofs_call:=ofs(out_str);
        sel_call:=conf_sel;

{=================Формирование таблицы GDT=================}

{---------------------------------Дескрипторы сегментов-------------------------------}
                                                          { кода программы: }
        init_gdt(code_sel,$ffff,longint(Cseg) shl 4,acc_code,0);
                                                       {  видеопамяти (DPL=3): }
        init_gdt(text_sel,4000-1,$b8000,acc_data+$60,0); {+$60}
                                                  {  кода подчиненной программы (C=1): }
        init_gdt(CONF_sel,$ffff,longint(cs_CONF) shl 4,acc_code+4,0);
                                                      {  кода пользователя (DPL=3): }
        init_gdt(USER_sel,$ffff,longint(cs_USER) shl 4, acc_code+$60,0);
                                                        { данных пользователя: }
```

```
{----------Поскольку Паскаль переменные разных модулей помещает----------}
{----------в сегмент данных основной программы, то для того чтобы---------}
{----------использовать переменные, объявленные в модуле USER,----------}
{----------сегмент данных пользователя накладывается на----------}
{----------сегмент данных основной программы (DPL=3):----------}
      init_gdt(data3_sel,$ffff,longint(Dseg) shl 4, acc_data+$60,0);
{----------Дескрипторы TSS задач----------}
                                                { Задачи USER_CHECK_TASK: }
      init_gdt(user_check_sel,sizeof(t_tss)-1,longint(Dseg) shl 4+
                                                ofs(user_check_tss),acc_TSS,0);
{----------Дескриптор шлюза вызова процедуры get_time модуля PROT----------}
      asm
            mov word ptr [offset gdt+gate_sel],offset get_time
            mov word ptr [offset gdt+gate_sel+2],PROT_sel
            mov byte ptr [offset gdt+gate_sel+4],0
            mov byte ptr [offset gdt+gate_sel+5],acc_call+$60
            mov word ptr [offset gdt+gate_sel+6],0
      end;
{----------Дескриптор LDT----------}
      init_gdt(LDT_sel,7,longint(Dseg) shl 4+ofs(ldt),acc_LDT,0);

{----------Формирование данных регистра GDTR и его загрузка----------
}
{---------(параметр: последний используемый в программе селектор GDT)-------}

      init_gdtr(LDT_sel);
{----------Определение базового адреса сегмента пользователя----------}

      base_adr:=longint(Dseg) shl 4 + ofs(segm_data);
{----------Формирование дескриптора данных таблицы LDT ----------}

      ldt.lim_l  :=0;                  { Размер сегмента пользователя - 1 байт }
      ldt.base_l :=base_adr;
      ldt.base_h :=base_adr shr 16;
      ldt.acc :=acc_data+$60;
      ldt.lim_h  :=0;
      ldt.base_hh:=0;
```

```
{==================Формирование сегментов TSS================}
                                                { Задачи Task_User: }
      init_tss(user_check_tss,USER_sel+3,data3_sel,text_sel,
      ofs(user_check_task),ofs(user_check_stack)+ sizeof(user_check_stack));
{----------Задача Task_User с уровнем привилегий пользователя (CPL=3)---------}
{----------------имеет следующие особенности формирования TSS:----------------}

      user_check_tss.ss:=data3_sel+3;          { Сегмент стека с DPL=3 RPL=3 }
      user_check_tss.ldt:=LDT_sel;
      user_check_tss.ss0:=data_sel;          { Адрес стека шлюза для программ }
      user_check_tss.esp0:=ofs(stack_0)+sizeof(stack_0);          { уровня 0 }
      user_check_tss.adr_BKVV:=360;                { Смещение поля БКВВ }
      for i:=0 to 31 do
      user_check_tss.BKVV[i]:=$FF;          { Запретить работу портов 0-255 }
{==========Формирование дескрипторов шлюзов таблицы IDT==========}
{==========тех обработчиков прерываний, что разработаны==========}
{======================в программе P_USER======================}

      init_idt($30,ofs(int_30h),PROT_sel,acc_trap_16+$60);  {}
      init_idt($32,ofs(int_32h),PROT_sel,acc_trap+$60);
{-----------------Сохранение и формирование данных регистра IDTR---------------}
{-----------------с его загрузкой для работы в защищенном режиме---------------}

      init_idtr;
{---------------------Программирование контроллеров прерываний-----------------}
{-----------------------------для работы в защищенном режиме-------------------}

      pic(1);
{-------------------------Определение смещения и селектора-----------------------}
{----------------точки окончания работы МП в защищенном режиме---------------}
{---------------(для выполнения косвенного межсегментного перехода------------}
{----------------------на эту точку при обработке исключений)-----------------}

      asm mov ofs_end_prot,offset end_prot end;
      sel_end_prot:=code_sel;
{----------------Сохранение содержимого сегментных регистров и SP:-----------}
```

```
      real_cs:=Cseg;                                          { CS, }
      memw[0:4*$60]:=Dseg;                                    { DS, }
      real_ss:=Sseg;                                          { SS, }
      asm mov real_es,es end;                                 { ES }
      real_sp:=SPtr;                                          { и SP }

{------------------------------Переход в защищенный режим--------------------------}
{-----------------------для МП 80386 и МП последующих моделей---------------------}
{--------------------путем установки бита PE в регистре управления CR0----------}
      asm
            db 0fh,20h,0c0h                                   { MOV EAX,CR0 }
            or al,1
            db 0fh,22h,0c0h                                   { MOV CR0,EAX }
{----------------------Межсегментный переход на метку @prot----------------------}
{-----------------для загрузки регистра CS и сброса очереди команд----------------}

            db 0eah
            dw offset @prot
            dw code_sel
{==================Работа в защищенном режим================}

{---------------------Загрузка сегментных регистров DS, SS и ES-------------------}
{---------------------------соответствующими селекторами-------------------------}

   @prot:
            mov ds,data_sel
            mov ss,stack_sel
            mov es,text_sel
{-------------------------Загрузка в TR селектора задачи main---------------------}

            db 0fh,0,1eh                                      { LTR main_sel:  }
            dw v_main_sel
{-------------------------------------Очистка экрана-----------------------------------}

            mov ax,102h
            mov cl,0
            mov bx,0
```

```
            int 30h
{----------------------------Задание 16-разрядного фона символов----------------------}

            mov ax,01h                              { Функция 0, подфункция 1:  }
            int 30h
{-------------------------Разрешение маскируемых прерываний-----------------------}

            sti
{-------------------------Вызов задачи USER_TASK командой CALL---------------------}

            db 9ah
            dw 0
            dw user_check_sel
{----------------------------Ожидание нажатия клавиши Esc----------------------------}

   @wait:
            cmp scan,1
            jnz @wait
            end_prot:
{===============Подготовка к возврату в реальный режим============}

{--------------------Установка параметров сегментов DS, SS и ES------------------}
{----------для работы в реальном режиме (Limit=0FFFFh, ED=0, W=1)----------}

            mov ds,real_sel
            mov ss,real_sel
            mov es,real_sel
{----------------------------Запрет маскируемых прерываний---------------------------}

            cli
{-----------------------Восстановление атрибутов таблицы IDT----------------------}
{---------------------------------------для работы в реальном режиме----------------}

            db 0fh,01h,1eh                          { LIDT idtr_r }
            dw idtr_r
{--------------------Возврат в реальный режим по команде MOV---------------------}
{-----------------путем сброса бита PE в регистре управления CR0:----------------}
```

```
            db 0fh,20h,0c0h                                  { MOV EAX,CR0 }
            and al,not 1
            db 0fh,22h,0c0h                                  { MOV CR0,EAX }
{-----------------------Межсегментный переход на метку real----------------------}

            push real_cs
            push offset real
            retf
{==============Работа после возврата в реальный режим=============}

{-------------------------------Восстановление регистров-------------------------------}
        real:
            xor ax,ax
            mov ds,ax
            mov ds,[4*60h]                                    { DS, }
            mov ss,real_ss                                    { SS, }
            mov es,real_es                                    { ES }
            mov sp,real_sp                                    { и SP }
        end;
{------------------Перепрограммирование контроллеров прерываний----------------}
{-----------------------------для работы в реальном режиме--------------------------}

        pic(0);
{-------------------------Разрешение аппаратных прерываний-----------------------}

        asm sti end;                                          { маскируемых }
        port[$70]:=$d;                                        { и немаскируемых }
{-----------------------Сброс состояния клавиш-переключателей--------------------}

        mem[$40:$17]:=0;
{--------------Сохранение, если была нажата клавиша "Print Screen",-------------}
{-----------------содержимого экрана на диске в подкаталоге EKRAN-------------}

        save_scr;
    end.
```

4.15. Текст модуля USER

```
{----------Модуль содержит задачу и процедуру пользователя,----------}
{----------работающие на третьем уровне привилегий (DPL=3)----------}
    unit user;
    interface
    uses prot;
    const
        addr_port:word=$60;
{----------Разрешение/запрет (1/0) проверки защиты----------}

      test_data:byte=0;                    { сегментов данных }
      est_code:byte=0;                     { сегментов кода }
      test_io:byte=1;                      { устройств ввода/вывода }
    var
{----------TSS задачи----------}
      user_check_tss:t_tss;
      ofs_call,sel_call:                   { Смещение и селектор процедуры }
      word;                                { при межсегментном вызове }
      ldtr:word;
    procedure user_check_task;

    implementation

      const
          s1:string='Работает программа пользователя (CPL=3)';
          s21:string='Сегмент данных пользователя (LDT) содержит: ';
          s22:string='Сегмент данных пользователя содержит: ';
          s23:string='Сегмент данных программы содержит: ';
          s31:string='Вызов процедуры с DPL=3';
          s33:string='Вызов процедуры get_time с DPL=0 через шлюз:';
          s34:string='Вызов процедуры с DPL=0 запрещен командой LAR';
          s35:string='Вызов процедуры с DPL=0:';
          s40:string='Чтение scan-кода:';
          s41:string='Доступ к порту 60h разрешен - ';
          s42:string='Доступ к порту 60h запрещен - ';
{----------Процедура уровня привилегий 3 выводит на экран сообщение----------}
```

```
{----------------------------"Вызвана процедура с CPL=3"----------------------------}

    procedure out_str3; assembler;
            asm
                mov ax,100h                                  { Установка маркера }
                mov bx,0103h                                                 { (1,3) }
                int 30h
                mov ah,4                                  { Вывод строки s31 на экран }
                mov cl,1ah
                mov si,offset s31
                int 30h
            end;
    procedure user_check_task;assembler;
            asm
{----------------------------------Вывод на экран строки----------------------------------}
{-------------------------"Работа программы пользователя (CPL=3)"------------------}
                mov ax,100h                                  { Установка маркера: }
                mov bx,0101h                                                 { (1,1) }
                int 30h
                mov ah,4                                  { Вывод строки s1 на экран: }
                mov cl,1eh
                mov si,offset s1
                int $30
                cmp test_data,1                      { Проверять сегмент данных? }
                jnz @cod
{================Проверка защиты сегментов даннях===============}

{------------------Программа пользователя с CPL=3 читает данные----------------}
{-----------------------из сегмента данных, который задан в LDT ---------------------}
{---------------------------и имеет уровень защиты DPL=3----------------------------}

                mov ax,100h                                  { Установка маркера }
                mov bx,103h                                                  { (3,1) }
                int 30h
                mov ah,4                                  { Вывод строки s21 на экран }
                mov cl,1ah
                mov si,offset s21
```

```
                int 30h
                mov ax,4                        { Селектор LDT с индексом 0 }
                push ds
                mov ds,ax               { Загрузка селектора LDT в DS позволяет }
                mov dl,[0]                { читать данные из сегмента данных }
                pop ds                                  { пользователя (LDT) }
                mov ax,300h                     { Вывод данных на экран }
                mov cl,1bh
                int 30h
{------------------Программа пользователя с CPL=3 читает данные----------------}
{----------------------------из сегмента данных, который-----------------------------}
{----------------------------имеет уровень защиты DPL=3-----------------------------}
                mov ax,100h                             { Установка маркера }
                mov bx,105h                                             { (5,1) }
                int 30h
                mov ah,4                        { Вывод строки s22 на экран }
                mov cl,1ah
                mov si,offset s22
                int 30h
                mov dl,[2]          { Читать из сегмента данных пользователя }
                mov ax,300h                     { Вывод данных на экран }
                mov cl,1bh
                int 30h
{-----При попытке программы пользователя с CPL=3 прочитать данные-------}
{------------------из сегмента данных с уровнем защиты DPL=0--------------------}
{--------------возникает исключение 13 "Нарушение общей защиты"--------------}

                mov ax,100h                             { Установка маркера }
                mov bx,107h                                             { (7,1) }
                int 30h
                mov ah,4                        { Вывод строки s23 на экран }
                mov cl,1ah
                mov si,offset s23
                int 30h
                push ds
                mov ds,data_sel
                mov dl,[2]
```

```
                pop ds
                mov ax,301h                     { Вывод данных на экран }
                mov cl,1bh
                int 30h
                jmp @end
        @cod:
                cmp test_code,1                 { Проверять сегмент кода? }
                jnz @IO
{================Проверка защиты сегментов кода================}

{----------Программа пользователя с CPL=3 вызывает процедуру out_str3--------}
{-------------------------с тем же уровнем привилегий (DPL=3)-----------------------}

                db 9ah
                dw offset out_str3
                dw USER_sel
{--------------------Программа пользователя с CPL=3 вызывает---------------------}
{--------------------подчиненную (C=1) процедуру out_str с DPL=0------------------}

                db 0ffh,1eh
                dw ofs_call
{-------------Программа пользователя с CPL=3 вызывает процедуру-------------}
{--------------------------get_time с DPL=0 через шлюз вызова-----------------------}

                mov ax,100h                     { Установка маркера }
                mov bx,0107h                                  { (1,7) }
                int 30h
                mov ah,4                        { Вывод строки s33 на экран }
                mov cl,1ah
                mov si,offset s33
                int 30h
                mov bx,2F07h
                db 9ah
                dw 0
                dw gate_sel

{-------------Проверка с помощью команды LAR возможности вызова-------------}
```

```
{----------программой пользователя процедуры get_time с DPL=0:----------}
{----------------------команда LAR устанавливает флаг ZF=1,----------------------}
{--------------------если проверка прав доступа прошла успешно--------------------}

                mov cx,PROT_sel     { В CX - селектор вызываемой процедуры }
                db 0fh,2,0c1h                                  { LAR AX,CX }
                jz @call                        { Если ZF=1 - вызов процедуры, }
                mov ax,100h                                    { если нет - }
                mov bx,0109h                    { вывод строки s34 на экран }
                int 30h
                mov ah,4
                mov cl,1ch
                mov si,offset s34
                int 30h
{--------------Программа пользователя с CPL=3 вызывает процедуру----------}
{--------------------get_time с DPL=0 непосредственно, что приводит----------}
{------------к возникновению исключения 13 "Нарушение общей защиты"---------}

                mov ax,100h                             { Установка маркера }
                mov bx,010Bh                                       { (1,11) }
                int 30h
                mov ah,4                          { Вывод строки s35 на экран }
                mov cl,1ah
                mov si,offset s35
                int 30h
                @call:
                mov bx,2F0Bh
                db 9ah
                dw offset get_time
                dw PROT_sel
                jmp @end
                @IO:
                cmp test_IO,1                   { Проверять сегмент данных? }
                jnz @end
{==============Проверка защиты устройств ввода/вывода===========}

                mov ax,100h                             { Установка маркера }
```

```
                mov bx,0703h                                    { (7,3) }
                int 30h
                mov ah,4                        { Вывод строки s40 на экран }
                mov cl,1ah
                mov si,offset s40
                int 30h
                mov ax,100h                             { Установка маркера }
                mov bx,0105h                                    { (1,5) }
                int 30h
                mov ah,4                        { Вывод строки s41 на экран }
                mov cl,1ah
                mov si,offset s41
                int 30h
{----------------------------Разрешение доступа к порту 60h----------------------------}
{---------------------с помощью битовой карты ввода/вывода (БКВВ)----------------}

                mov cx,addr_port
                and cl,7                { Остаток от деления адреса порта на 8 }
                mov dl,1
                @l:             { Сдвиг влево значения 1 на величину остатка }
                cmp cl,0
                jz @in
                shl dl,1
                dec cl
                jmp @l
        @in:
                push dx
                not dl                  { DL содержит 0 в бите с номером остатка }
                mov bx,addr_port
                shr bx,3                { Частное от деления адреса порта на 8 }
                push bx
{-----------------------Установка в 0 бита с номером 60h в БКВВ---------------------}
                and byte ptr [offset user_check_tss+360+bx],dl {0FEh}
                mov dx,addr_port
                in al, dx                               { Чтение scan-кода из порта }
                mov dl,al
                mov ax,300h                             { Вывод данных на экран }
```

```
                mov cl,1bh
                int 30h
{----------------------------Чтение из неразрешенного БКВВ ----------------------------}
{----------------------------порта 1 вызывает исключение 13----------------------------}

                mov ax,100h                          { Установка маркера }
                mov bx,0107h                                    { (1,7) }
                int 30h
                mov ah,4                          { Вывод строки s42 на экран }
                mov cl,1ah
                mov si,offset s42
                int 30h
{----------------------Установка в 1 бита с номером 60h в БКВВ----------------------}
                pop bx
                pop dx
                or byte ptr [offset user_check_tss+360+bx],dl
                in al,60h
    @end:
                iret
    end;
end.
```

4.16 Текст модуля CONF

```
{-------------------Модуль содержит подчиненные (C=1) процедуры-----------------}
{-----------------------------------уровня привилегий 0-----------------------------------}
unit conf;
interface
  procedure out_str;
implementation
  const
        s32:string='Вызвана подчиненная процедура с C=1 и DPL=0';
{-------------------Процедура out_str выводит на экран сообщение-----------------}
{---------------------"Вызвана подчиненная процедура с DPL=0"---------------------}
  procedure out_str;assembler;
    asm
        mov ax,100h                          { Установка маркера }
```

```
        mov bx,0105h                                    { (1,5) }
        int 30h
        mov ah,4                                { Вывод строки s5 на экран }
        mov cl,1ah
        mov si,offset s32
        int 30h
    end;{out_str}
end.
```

4.17. Індивідуальні завдання

При виконанні завдань по перевірці правил захисту пам'яті необхідно також розглянути матеріал. Крім того, необхідно познайомитися з прикладами програм, що реалізують роботу мікропроцесора у захищеному режимі, обробку переривань і багатозадачну роботу МП.

Оскільки в правилах по захисту пам'яті (сегментів коду, стека і даних та зовнішніх пристроїв) використовуються такі параметри, як поточний рівень привілей CPL, рівень привілей запиту RPL, рівень захисту сегмента DPL і рівень привілей доступу до зовнішніх пристроїв IOPL, необхідно при рішенні індивідуальних завдань уміти встановлювати необхідні значення цих параметрів.

Значення DPL для кожного сегмента встановлюється в дескрипторі відповідного сегмента в полі байта доступу при формуванні GDT. Значення RPL встановлюється при доступі до сегмента в полі RPL селектора, що завантажується у відповідний сегментний регістр.

Значення IOPL встановлюється шляхом занесення його в регістр прапорів EFLAGS.

Складніше діло з установкою необхідного значення CPL. Як відзначалося вище, в реальному режимі всі програми мають рівень привілей CPL = 0. При переході у захищений режим і при роботі у захищеному режимі цей рівень привілей зберігається, оскільки завжди заборонені переходи від більш привілейованих програм до менш привілейованих (див. підрозділи 4.3, 4.7).

Єдиною можливістю передати управління менш привілейованій програмі є організація багатозадачного режиму роботи мікропроцесора. При цьому кожній задачі можна присвоїти довільний рівень привілеїв. Це робиться шляхом занесення в TSS відповідної задачі в полі регістра CS в розряди 0-1 (RPL) необхідного значення рівня привілеїв.

Задача **Task_User** має найменший рівень привілеїв 3, тобто рівень привілеїв користувача,

Оскільки при виконанні завдань по перевірці правил захисту пам'яті необхідно перейти у захищений режим МП, здійснити обробку переривань (10, 13, а можливо й інших), організувати багатозадачну роботу МП, то для цього вимагається ознайомитися з матеріалом і програмами, наведеними в розд. 1-3.

1. Перевірити дозвіл доступу до сегментів даних при DPL < RPL і заборони доступу при DPL>RPL.

2. Перевірити забобону виклику менш привілейованих програм.

3. Перевірити дозвіл доступу до сегмента стека і заборону доступу при DPL <> RPL.

4. Перевірити виклик програмою користувача (CPL = 3) програми з рівнем привілей 1 через шлюз виклику.

5. Перевірити дозвіл доступу до зовнішніх пристроїв при CPL <= IOPL і заборони доступу при CPL > IOPL.

6. Перевірити правила захисту при виклику програм через шлюз.

7. Перевірити дозвіл доступу до сегментів даних при DPL < CPL і заборону доступу при DPL > CPL.

8. Перевірити виклик більш привілейованих програм, що мають ознаку підпорядкування (C = 1).

9. Перевірити дозвіл доступу до сегмента стека і заборону доступу при DPL <> CPL.

10. Перевірити виклик програмою користувача (CPL = 3) програми з рівнем привілей 2 через шлюз виклику.

11. Встановити і перевірити дозвіл доступу тільки до одного зовнішнього пристрою.

12. Перевірити правила захисту при виклику задач.

13. Перевірити виклик програмою користувача (CPL = 3) програми з рівнем привілей 0 через шлюз виклику.

14. Перевірити дозвіл доступу до сегмента стека і заборони доступу при W = 0.

15. Перевірити правила захисту при виклику задач через шлюз задач.

16. Встановити і перевірити дозвіл доступу тільки до зовнішніх пристроїв з адресами, меншими 100h.

СПИСОК ЛІТЕРАТУРИ

1. Новожилов О.П. Основы компьютерной техники. / учебн. пособие. М.: ИТ Радио-Софт, 2008, 456 с., ил.

2. Рисований О.М., Соколов С.О., Зиков І.С., Скородєлов В.В. / Під ред. Рисованого О.М. Цифрові пристрої та мікропроцесори. Організація та функціонування: Навчальний посібник. Харків: ХВУ, 2002. – 328 с.

3. Кравец В.А., Рисованный А.М., Домнин Ф.А., Зыков И.С., Скороделов В.В., Шеин А.Н. Микропроцессоры и микропроцессорные системы. Кн. 1. Архитектура и функционирование: учебное пособие. Харьков: ХВУ, 2000, 350 с.

4. Кравец В.А., Рисованный А.М., Домнин Ф.А., Зыков И.С., Скороделов В.В., Шеин А.Н. Микропроцессоры и микропроцессорные системы. Кн. 2 Программирование, разработка устройств и систем. Учебное пособие. Харьков: ХВУ, 2000, 350 с.

5. Intel Architecture Software Developer’s Manuals. www.intel.com/design/.../manuals/

6. Intel 64 and IA-32 Architectures Manuals www.intel.com/products/.../manuals/

7. Матвієнко М.П., Розен В.П., Закладний О.М. Архітектура комп'ютерів: Навчальний посібник. К.: Ліра-К, 2013. – 164 с.

8. Рисований О.М. Системне програмування: навч. підручник / О.М. Рисований. – Харків: НТУ “ХПІ”, 2010. – 912 с.

9. Аблязов Р.З. Программирование на ассемблере на платформе x86-64. – М.: ДМК Пресс, 2011. – 304 с.

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

БКВВ – бітова карта введення-виведення
ЗП – зовнішні пристрої
ЗР – захищений режим
МП – мікропроцесор
ОС – операційні системи
ПВВ – пристрої введення-виведення
ПЗЗР – програмне забезпечення захищеного режиму
CPU – central processing unit
GDT – global descriptor table
LDT – local descriptor table

Навчальне видання

ЗИКОВ Ігор Семенович
МЕЖЕРИЦЬКИЙ Сергій Геннадійович
ПОДОРОЖНЯК Андрій Олексійович
ХАВІНА Інна Петрівна

ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ У ЗАХИЩЕНОМУ РЕЖИМІ

Навчально-методичний посібник
для студентів комп'ютерних спеціальностей
вищих навчальних закладів

Роботу рекомендував до видання проф. М. Й. Заполовський
Відповідальний за випуск С. Г. Семенов

Дизайн обкладинки О. В. Любченко

В авторській редакції

План 2018 р., поз. 21
Підписано до друку 30.01.18. Формат 60x84 1/16. Папір офсет.
Друк цифровий. Гарнітура Times New Roman. Ум. друк. арк. 15,34.
Наклад 300 прим. Зам № 0130/9-18. Ціна договірна.

Видавництво ТОВ «ДІСА ПЛЮС»
Свідоцтво суб'єкта видавничої справи: серія ДК № 4047 від 15.04.2011 р.
61029, м. Харків, шосе Салтівське, буд. 154,
тел. (057) 768-03-15, e-mail: disadruk@gmail.com

Надруковано з готових оригінал-макетів у друкарні ФОП Петров В. В.
Єдиний державний реєстр юридичних осіб та фізичних осіб-підприємців.
Запис № 2400000000106167 від 08.01.2009 р.
61144, м. Харків, вул. Гв.Широнінців, 79в, к. 137,
тел. (057) 778-60-34, e-mail:bookfabrik@mail.ua

ISBN 978-617-7384-91-4