Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Архітектура комп′ютерів

Методичні рекомендації для виконання лабораторних робіт студентів денної форми навчання галузі 12 Інформаційні технології

ЗАТВЕРДЖЕНО
на засіданні кафедри кібербезпеки
та програмного забезпечення
(протокол №2 від 29.08.24)

Кропивницький
2024

Архітектура комп′ютерів: Методичні рекомендації до виконання лабораторних робіт для студентів денної форми навчання галузі 12 Інформаційні технології. / уклад. Минайленко Р.М. / М-во освіти і науки України, Центральноукр. нац. техн. ун-т; – Кропивницький: ЦНТУ– 2024. – 81 с.

Укладач: Минайленко Р.М

Рецензенти: Коваленко О.В., докт. техн. наук, доцент;

Улічев О.С., канд. техн. наук.

Зміст

Вступ……..4
1. Лабораторна робота № 1 Використання системного програмування в реальному режимі..... 7
2. Лабораторна робота № 2 Програмування інтервального таймера в IBM PC 20
3. Лабораторна робота № 3 Програмування годинника реального часу (RTC) в IBM PC.....28
4. Лабораторна робота № 4 Таймер операційної системи DOS та Windows 36
5. Лабораторна робота № 5 Програмування контролера переривань i8259 в IBM PC / AT.....43
6. Лабораторна робота № 6 Контролер прямого доступу до пам'яті 48
7. Лабораторна робота № 7 Дескриптори і таблиці глобальних дескрипторів..... 59
8. Лабораторна робота № 8 Програмування текстової відеопам'яті у захищеному режимі..... 68
9. Лабораторна робота № 9 Програмування режимів EGA та VGA відеоадаптера 73
10. Лабораторна робота № 10 Програмування відеоадаптера та текстової відеопам'яті комп'ютера у реальному режимі адресації 74
11. Лабораторна робота № 11 Розробка та ініціалізація власних шрифтів і нестандартних символів за допомогою програмування знакогенератора відеоадаптера ПК 77
12. Лабораторна робота № 12 Програмування принтера.....80
13. Рекомендована література 81

Вступ

Курс «Архітектура комп'ютерів» призначений для здобуття студентами знань та навичок про принципи організації та забезпечення функціонування комп'ютерів і систем, розглядаючи їх як комплекс технічних, інформаційних та програмних засобів, що призначені для вирішення широкого кола завдань забезпечення вирішення інформаційних процесів; формування необхідних теоретичних знань та практичних навичок у галузі побудови й функціонування комп'ютерів та комп'ютерних технологій, можливостей їх використання.

Метою викладання дисципліни «Архітектура комп'ютерів» є опанування особливостей архітектури сучасних обчислювальних систем, процесорів, комп'ютерної периферії та їхньої взаємодії; принципів та методів програмування, розуміння основних тенденцій розвитку та фундаментальних принципів функціонування комп'ютерних систем.

Основними **завданнями** вивчення дисципліни є:

– здатність вирішувати проблеми у галузі архітектури комп'ютерів, набуття практичних навичок що до будови, функціонування і взаємодії основних складових ЕОМ.

– здатність аргументувати вибір методів розв'язування задач, на основі отриманих знань про принципи організації та забезпечення функціонування сучасних обчислювальних систем, процесорів, комп'ютерної периферії та їхньої взаємодії, особливостей архітектури та програмування.

Для денної форми навчання:

Викладання курсу передбачає для засвоєння дисципліни традиційні лекційні заняття із застосуванням мультимедійних презентацій, у поєднанні з лабораторними заняттями.

Формат очний (Face to face)

Для заочної форми навчання:

Під час сесії формат очний (Face to face), у міжсесійний період – дистанційний (online).

Результати навчання

У результаті вивчення навчальної дисципліни «Архітектура комп'ютерів» студент буде:

Знати:

– N1. Наукові положення, що лежать в основі архітектури сучасних обчислювальних систем, процесорів, комп'ютерної периферії та їхньої взаємодії, принципів та методів програмування .

– N3. Існуючі технології в галузі архітектури комп'ютерів, розуміння основних тенденцій розвитку та фундаментальних принципів функціонування.

Вміти:

– N6. Застосовувати знання для розв'язування технічних задач пов'язаних з архітектурою комп'ютерів, а саме оцінювати характеристики комп'ютера на архітектурному та структурному рівнях; розробляти системи команд, формати і структуру даних, способи адресації команд та операндів; алгоритми обміну інформацією між пристроями пам'яті різного рівня; розробляти програмні та апаратні засоби обміну даними між процесором і зовнішніми пристроями в режимі програмного обміну, переривань програми та прямого доступу до пам'яті; розробляти архітектуру пристроїв вводу-виводу даних для різних

режимів взаємодії з процесором; розробляти архітектуру, мікроалгоритми, мікропрограми та програми для комп'ютерів і контролерів на базі мікропроцесорних комплектів ВІС.

- N13. Ідентифікувати, класифікувати та описувати особливості архітектури сучасних обчислювальних систем, процесорів, комп'ютерної периферії та їхньої взаємодії.
- Р11. Оформляти отримані робочі результати у вигляді презентацій, науково-технічних звітів.

Політика дисципліни

Академічна доброчесність:

Очікується, що студенти будуть дотримуватися принципів академічної доброчесності, усвідомлювати наслідки її порушення. Детальніше за посиланням URL http://www.kntu.kr.ua/doc/dobro.pdf

Відвідування занять

Відвідування занять є важливою складовою навчання. Очікується, що всі студенти відвідають лекції і практичні заняття курсу.

Пропущені заняття повинні бути відпрацьовані не пізніше, ніж за тиждень до залікової сесії.

Поведінка на заняттях

Недопустимість: запізнень на заняття, списування та плагіат, несвоєчасне виконання поставленого завдання.

При організації освітнього процесу в Центральноукраїнському національному технічному університеті студенти, викладачі та адміністрація діють відповідно до: Положення про організацію освітнього процесу; Положення про організацію вивчення навчальних дисциплін вільного вибору; Положення про рубіжний контроль успішності і сесійну атестацію студентів ЦНТУ; Кодексу академічної доброчесності ЦНТУ.

Критерії оцінювання. Знання здобувачів вищої освіти оцінюється при проведенні екзаменаційного контролю як з теоретичної, так і з практичної підготовки за такими критеріями:

– "відмінно" – здобувач вищої освіти досконало засвоїв теоретичний матеріал, глибоко і всебічно знає зміст навчальної дисципліни, основні положення наукових першоджерел та рекомендованої літератури, логічно мислить і будує відповіді, вільно використовує набуті теоретичні знання при аналізі практичного матеріалу, висловлює своє ставлення до тих чи інших проблем, демонструє високий рівень засвоєння практичних навичок;

– "добре" – здобувач вищої освіти добре засвоїв теоретичний матеріал, аргументовано викладає його, володіє основними аспектами з першоджерел та рекомендованої літератури, має практичні навички, висловлює свої міркування з приводу тих чи інших проблем, але припускається певних неточностей і похибок у логіці викладу теоретичного змісту або при аналізі практичного матеріалу;

– "задовільно" – здобувач вищої освіти, в основному, володіє теоретичними знаннями з навчальної дисципліни, орієнтується в першоджерелах та рекомендованій літературі, але непереконливо відповідає, додаткові питання викликають невпевненість або відсутність стабільних знань;

відповідаючи на запитання практичного характеру, виявляє неточності у знаннях, не вміє оцінювати факти та явища, пов’язувати їх із майбутньою діяльністю;

– "незадовільно" – здобувач вищої освіти не опанував навчальний матеріал дисципліни, не знає наукових фактів, визначень, майже не орієнтується в першоджерелах та рекомендованій літературі, відсутні наукове мислення, практичні навички не сформовані.

Шкала оцінювання: національна та ЄКТС

Оцінка за шкалою ECTS	Визначення	Оцінка		
		За національною системою (екзамен, диф. залік, курс. проект, курс. робота, практика)	За національною системою (залік)	За системою ЦНТУ
A	ВІДМІННО - відмінне виконання лише з незначною кількістю помилок	5 (відмінно)	Зараховано	90 – 100
B	ДУЖЕ ДОБРЕ - вище середнього рівня з кількома помилками	4 (добре)	Зараховано	82-89
C	ДОБРЕ - в загальному правильна робота з певною кількістю грубих помило			74-81
D	ЗАДОВІЛЬНО - непогано, але зі значною кількістю недоліків	3 (задовільно)	Зараховано	64-73
E	ОСТАТНЬО - виконання задовольняє мінімальні критерії			60-63
FX	НЕЗАДОВІЛЬНО - потрібно попрацювати перед тим, як перескласти	2 (незадовільно)	Не зараховано	35-59
F	НЕЗАДОВІЛЬНО - необхідна серйозна подальша робота			1-34

Лабораторна робота № 1

ТЕМА: Використання системного програмування в реальному режимі роботи центрального процесора для визначення конфігурації ПК IBM PC.

МЕТА: Ознайомитись з основами системного програмування, навчитись програмним шляхом одержувати інформацію про системні характеристики ПК типу IBM PC.

Короткі теоретичні відомості

Системна інформація про комп'ютер зберігається у зарезервованих комірках пам'яті, які завантажуються в молодші 640 Кб адресного простору ОЗП та використовуються системними та прикладними програмами під час роботи ПК.

Розглянемо спрощену схему розподілу пам'яті у реальному режимі (таблиця 1.1).

Таблиця 1.1 – Розподіл пам'яті у реальному режимі

Адреса	Значення
FFFFFF	ПЗП BIOS
F0000	Зарезервовано
C0000	Текстовий відеобуфер
B8000	Графічний відеобуфер
A0000	Область для завантаження транзитних програм (додатків користувача)
	Резидентна частина командного процессора
	Ядро системи MS DOS
	Драйвера пристроїв
00400	Область даних BIOS і DOS
00000	Таблиця векторів переривань

У молодших адресах пам'яті розміщується ядро ОС, перші 1024 байта якої займає таблиця векторів переривань. Кожен елемент цієї таблиці – 32-розрядна адреса підпрограми-обробника переривання (згадайте цю тему з "Мікропроцесорних ВІС"), яка задана у форматі

```
[сегмент] : [зміщення]
```

і називається *вектором переривань*. Ці адреси використовуються CPU виклику підпрограми обробки апаратних та програмних переривань.

Після таблиці векторів переривань розміщено велику ділянку оперативної пам'яті, яка використовується як область даних BIOS та операційної системи (ОС).

Наступні 640 Кбайт ОЗП – область пам’яті, у якій знаходяться драйвери пристроїв ОС. Вони забезпечують керування введенням/виведенням для більшості стандартних пристроїв (наприклад, для клавіатури, дискових накопичувачів, відео адаптера, послідовного та паралельного інтерфейсів).

Як відомо, існує багато моделей ПК IBM PC: Оригінальний IBM PC, XT, AT, PCjr, Compag XT, Sperry PC і ін. Є три способи визначення моделі та отримання деякої інформації про конфігурацію комп’ютера:

1) прочитати дану інформацію з комірок ПЗП BIOS;
2) викликати одну з функцій переривання 15h, яка повертає адресу таблиці конфігурації;
3) за допомогою системної утиліти debug (дебагер).

Спосіб 1. У постійній пам’яті BIOS за адресою F000:FFFE міститься байт, значення якого є ідентифікацією типу комп’ютера (див. таблицю 1.2). За адресою F000:FFF5 міститься дата виготовлення BIOS, яка записана у коді ASCII в американському форматі (MM/DD/YY – місяць/день/рік) та займає 8 байт.

Таблиця 1.2 – Значення байту ідентифікації типу ПК

Код	Тип комп’ютера
FF	Оригінальний IBM PC
FE	XT, Portable PC
FD	PCjr
FC	AT
FB	XT з пам’яттю 640K на материнській платі
FA	PS/2 модель 25 або 30
F9	Convertible PC
F8	PS/2 моделі 55SX,70,80
9A	Compag XT, Compag Plus
30	Sperry PC
2D	Compag PC, Compag Deskpro

Для читання вмісту даної комірки області пам’яті BIOS можна використати стандартну функцію TC peekb():

```
unsigned char Val = peekb(Seg, Offset) ,
```

де параметр `Seg` – значення сегменту, `Offset` – зміщення.

Приклад використання способу 1 наведено в лістингу 1.1.

Лістинг 1.1 – Програма одержання коду типу ПК IBM PC.

```
#include <stdio.h>
#include <conio.h>

int main (void)
{
  unsigned char KOD;
```

```
KOD = peekb(0xF000,0xFFFE);
  printf("Код типу комп'ютера: %xh\n", KOD);
  return 0;
 }
```

Спосіб 2. Більш детальну інформацію можна отримати, викликавши функцію C0h переривання BIOS INT 15h. Для генерації цього переривання необхідно:

на вході: AH=C0h

{виклик переривання}

на виході: ES:BX = адреса таблиці конфігурації

CF = 0, якщо виклик переривання пройшов успішно

CF = 1, якщо дана версія BIOS не підтримує виконання функції C0h.

Після виконання переривання регістри ES:BX будуть вказувати на таблицю в області ПЗП BIOS, значення якої наведено у таблиці 1.3.

Таблиця 1.3 – Таблиця ПЗП BIOS ідентифікації комп'ютера

Зміщення	Розмір, байт	Опис
(+0)	2	Розмір таблиці в байтах
(+2)	1	Код моделі
(+3)	1	Додатковий код моделі
(+4)	1	Версія BIOS revision: 0 – для першої реалізації; 2 - для другої і т.д.
(+5)	1	Байт конфігурації обладнання
	біт 7	канал 3 DMA використовується дисковою системою базового введення/виведення (дисковою BIOS)
	біт 6	Встановлено 2й контролер переривання 8259;
	біт 5	Встановлено RTC;
	біт 4	Після кожного виклику переривання від клавіатури INT 9h викликається функція 4Fh переривання INT 15h
	біт 3	У BIOS реалізовано функцію очікування зовнішньої події
	біт 2	Використовується розширена область даних BIOS
	біт 1	Якщо встановлено – використовується шина Micro Channel, ні – ISA
	біт 0	Зарезервовано
6	2	Зарезервовано і містить 0
8	2	Зарезервовано і містить 0

Довідка: в TC регістри зарезервовані як _AH, _BX, _CX, _ES і т.д.

Для генерації переривання в ASM використовується команда `INT` або функція TC

```
geninterrupt(Num),
```

де параметр функції `Num` – номер переривання (наприклад, geninterrupt(0x15);).

Приклад використання способу 1 наведено в лістингу 1.1.

Лістинг 1.2 – Програма виклику функції C0h переривання 15h.

```
#include <stdio.h>
#include <conio.h>
```

```
int main (void)
     {
       _AX=0x0C;
       geninterrupt(0x15);
       printf("Адреса таблиці конфігурації ПК: %x:%x\n", _ES, _BX);
       return 0;
     }
```

Спосіб 3. Системна програма DEBUG, яка у ОС Windows знаходиться в системній директорії WINDOWS\system32, дозволяє переглядати пам'ять, вводити програми і здійснювати трасування їх виконання, а також може бути використана для одержання системної інформації про ПК.

Після запуску програми debug.exe (Пуск → Выполнить → debug) або завантаження її з диска у пам'ять з'явиться запрошення до роботи – з'явиться дефіс, що і є сигналом готовності програми до приймання команд.

Перегляд пам'яті за допомогою дебагера

Перевіримо, насамперед, розмір доступної для роботи пам'яті. Дані значення знаходяться у чарунках пам'яті 413h і 414h. Це значення можна переглянути із Debug за адресою, яка складається з двох частин:

- 400 – адреса сегменту, який необхідно записати як 40 (останній нуль не пишеться);
- 13 – зміщення від початку сегменту.

Таким чином, для виконання описаної раніше операції необхідно у дебагері у командному рядку задати:

```
D 40:13
```

Перші два байти у шістнадцятковій системі лічби означатимуть об'єм пам'яті у Кб, при чому байти будуть розміщення у зворотній послідовності.

Серійний номер

Серійний номер комп'ютера "зашито" у ROM за адресою FE000h зі зміщенням 00h.

```
D FE00:0
```

Дата ROM BIOS

Дата ROM BIOS в форматі MM/DD/YY знаходиться за адресою FFFF5h, тобто необхідно задати:

```
D FFFF:05
```

Машинні (асемблерні) коди

Проілюструємо просту програму на машинній мові, її представлення у пам'яті і результати її виконання. Програма:

```
B82301      mov AX,0123h
052500      add AX,0025h
8BD8        mov BX,AX
03D8        add AX,BX
8BCB        mov CX,BX
2BCB        sub AX,AX
```

```
   90          nop
CB          повернення у DOS
```

Треба зауважити, що машинні команди мають різну довжину: один, два або три байти. Машинні коди знаходяться у пам'яті безпосередньо один за іншим. Виконання програми починаєтьсяз першої команди і послідовно виконується до команди завершення роботи.

Можна ввести наведену вище програму безпосередньо у пам'ять комп'ютера і виконати її покомандно. В той же час можна переглянути вміст регістрів після виконання кожної команди.

Для того, щоб роздрукувати результати роботи дебагера необхідно натиснути <Ctrl>+<PrtScr>.

Для безпосеренього введення програми на машинній мові необхідно ввести наступну команду:

```
E DS:100 B8 21 01 05 25 00
```

Команда E означає введення (Enter). Параметр CS:100 визначає адресу пам'яті, куди будуть вводитись команди, – 100_{16} (265) байт від початку сегменту кодів (звичайна стартова адреса для машинних кодів у відлагоджувача Debug). Команда E записує кожну пару шістнадцяткових цифр у пам'ять у вигляді байту, починаючи з адреси CS:100 до дареси CS:105.

Тому наступною командою має бути

```
E CS:106 8B D8 03 D8 8B CB
E CS:10C 2B C8 2B C0 90 CB
```

Правильність введених у пам'ять кодів треба обов'язково перевіряти і у разі помилки обов'язково повторити запис необхідних значень.

Для перегляду стану всіх регістрі процесора необхідно ввести команду **R** і відлагоджувач виведе на екран стан всіх регістрів. Наприклад,

```
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0100     NV UP EI PL NZ NA PO NC
0B2B:0100 B82301        MOV     AX,0123
```

Стан регістру IP (вказівник команд) виводиться у вигляді IP=0100, що є свідченням того, що команда, яка виконається, знаходиться зі зміщенням 100h байт від початку сегмента кодів.

Відразу за значенням регістру IP виводиться флаги флагового регістру

```
NV UP EI PL NZ NA PO NC
```

Як видно, команда R також показує за зміщенням 100h першу машинну команду, яка виконається (MOV AX,0123).

Для покрокового виконання програми необхідно дати команду T.

```
-E CS:100 B8 23 01 05 25 00
-E CS:106 8B D8 03 D8 8B CB
-E CS:10C 2B C8 2B C0 90 CB
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0100     NV UP EI PL NZ NA PO NC
0B2B:0100 B82301        MOV     AX,0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0103     NV UP EI PL NZ NA PO NC
0B2B:0103 052500        ADD     AX,0025
-T
```

```
AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0106   NV UP EI PL NZ NA PE NC
0B2B:0106 8BD8          MOV     BX,AX
-T

AX=0148  BX=0148  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0108   NV UP EI PL NZ NA PE NC
0B2B:0108 03D8          ADD     BX,AX
-T

AX=0148  BX=0290  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=010A   NV UP EI PL NZ AC PE NC
0B2B:010A 8BCB          MOV     CX,BX
-T

AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=010C   NV UP EI PL NZ AC PE NC
0B2B:010C 2BC8          SUB     CX,AX
-T

AX=0148  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=010E   NV UP EI PL NZ AC PE NC
0B2B:010E 2BC0          SUB     AX,AX
-T

AX=0000  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0110   NV UP EI PL ZR NA PE NC
0B2B:0110 90            NOP
-T

AX=0000  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0111   NV UP EI PL ZR NA PE NC
0B2B:0111 CB            RETF
```

Запис CS=0B2B означає, що початок сегмента кода програми знаходиться зі зміщенням 0B2B0h. Значення 0B2B:0100 означає 100_{16} байт від початкової адреси 0B2Bh у регістрі CS.

Для перегляду програми у машинних кодах в сегменті кодів треба ввести D:

```
-D CS:100
0B2B:0100  B8 23 01 05 25 00 8B D8-03 D8 8B CB 2B C8 2B C0   .#..%. ..... +.+.
0B2B:0110  90 CB E9 C9 D7 E9 C3 D7-89 7E 02 80 34 00 1A 0B   .........~..4...
0B2B:0120  3F 2E B9 08 00 F3 AA 86-C4 AA 86 C4 B1 03 F3 AA   ?...............
0B2B:0130  32 C0 AA C3 BE 69 9B BF-2F 96 B4 60 CD 21 BE      2....i../..`.!..
0B2B:0140  D8 95 BF 2F 96 E8 CB E3 C3-33 C0 89 3E 93 9C A2   ../.....3..>....
0B2B:0150  96 9C A2 97 9C 8A F8 9C 57-33 C9 88 0E C2 9A AC   .......W3.......
0B2B:0160  E8 73 E3 75 1D 3C 20 74 F6-3C 09 74 F2 86 06      s.u.< t.<.t.....
0B2B:0170  97 9C 0A C0 74 EA F6 C7 80 74-05 C6 06 C2 9A      ..t....t. ... ..-
           01 E9 2D
```

Для того, щоб ще раз провести виконання програми необхідно скинути регістр IP, що робить команда

```
R  IP,
```

після чого на екрані з’явиться запит зміщення:

```
-R IP
IP 0100
:100
```

Тепер можна повторити трасування командою T.

Завершення роботи Debug здійснюється командою Q (Quit – вихід).

Визначення даних

Визначимо облать даних, яка містить відповідні значення:

Адреса в DS	Значення	Номери байтів
0000	2301	0 і 1
0002	2500	2 і 3
0004	0000	4 і 5
0006	2A2A2A	6, 7 і 8

A10000	Переслати слово, яке починається в DS за адресою 0000, у регістр AX
03060200	Додати слово, яке починається у DS за адресою 0002, до AX
A30400	Переслати вміст AX у слово, яке починається у DS за адресою 0004
CB	Повереннення в DOS

Введемо команди E для сегмента даних:

```
-E DS:00 23 01 25 00 00 00
-E DS:06 2A 2A 2A
```

Перша команда записує три слова (шість байтів) у початок сегменту даних, DS:00. Зверніть увагу, що кожне слово вводиться у зворотній послідовності.

Друга команда записує три зірочки (***) для того, щоб їх можна було побачити після виконання команди **D** (Dump) – іншого значення ці символи не мають.

Запишемо тепер код команди

```
mov AX,[0000]
```

для завантаження у регітр AX записаних у сегмент даних зі зміщенням 0000 даних:

```
-E cs:100 A1 00 00 03 06 02 00
-E CS:107 A3 04 00 CB
-D DS:000
0B2B:0000  23 01 25 00 00 00 2A 2A-2A F0 4F 03 78 05 8A 03   #.%...***.O.x...
0B2B:0010  78 05 17 03 78 05 1D 04-01 01 01 00 02 FF FF FF   x...x...........
0B2B:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 27 05 4E 01   ............ '.N.
0B2B:0030  4F 0A 14 00 18 00 2B 0B-FF FF FF FF 00 00 00 00   O.....+.........
0B2B:0040  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0B2B:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20   .!...........
0B2B:0060  20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20            .....
0B2B:0070  20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00            ........
-D CS:100
0B2B:0100  A1 00 00 03 06 02 00 A3-04 00 CB CB 2B C8 2B C0   ........... +.+.
0B2B:0110  90 CB E9 C9 D7 E9 C3 D7-89 7E 02 80 34 00 1A 0B   .........~..4...
0B2B:0120  3F 2E B9 08 00 F3 AA 86-C4 AA 86 C4 B1 03 F3 AA   ?...............
0B2B:0130  32 C0 AA C3 BE 69 9B BF-2F 96 B4 60 CD 21 BE D8   2....i../..`.!..
0B2B:0140  95 BF 2F 96 E8 CB E3 C3-33 C0 89 3E 93 9C A2 96   ../.....3..>....
0B2B:0150  9C A2 97 9C 8A F8 9C 57-33 C9 88 0E C2 9A AC E8   .......W3.......
0B2B:0160  73 E3 75 1D 3C 20 74 F6-3C 09 74 F2 86 06 97 9C   s.u.< t.<.t.....
0B2B:0170  0A C0 74 EA F6 C7 80 74-05 C6 06 C2 9A 01 E9 2D   ..t....t.... ..-
```

Тепер введіть R для перевірки стану регістрі і мнемоніки команди.

```
-R
AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0100  NV UP EI PL NZ AC PE NC
0B2B:0100 A10000      MOV AX,[0000]          DS:0000=0123
```

Виконаємо цю команду і перевіримо результат:

```
-T

AX=0123  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0103  NV UP EI PL NZ AC PE NC
0B2B:0103 03060200  ADD AX,[0002]            DS:0002=0025
-T

AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=0107   NV UP EI PL NZ NA PE NC
0B2B:0107 A30400  MOV [0004],AX                          DS:0004=0000
-T

AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B2B  ES=0B2B  SS=0B2B  CS=0B2B  IP=010A   NV UP EI PL NZ NA PE NC
```

```
0B2B:010A CB            RETF
```

Отже, після команди T було виконано команду mov AX,[0000], яка пересилає у регістр AX вміст слова, яке знаходиться з нулевим зміщенням. Після її виконання регістр AX містить 0123. Наступна команда ADD виконує додавання вмісту AX (0123) і даних зі зміщенням 0002 – 0025, тобто 0148. Команда mov [0004],AX переміщує результат додавання у чарунку зі зміщення 0004.

```
-D DS:004
0B2B:0000              48 01 2A 2A-2A F0 4F 03 78 05 8A 03       H.***.O.x...
0B2B:0010  78 05 17 03 78 05 1D 04-01 01 01 00 02 FF FF FF    x...x...........
0B2B:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 27 05 4E 01    ............ '.N.
0B2B:0030  4F 0A 14 00 18 00 2B 0B-FF FF FF FF 00 00 00 00    O.....+.........
0B2B:0040  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
0B2B:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20    .!...........
0B2B:0060  20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20    .....
0B2B:0070  20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00    ........
0B2B:0080  00 0D 20 20                                         ..
```

Машинна адресація

Для доступу до машинної команди процесор визначає її адресу із регістру CS з зміщенням у регістрі IP. Наприклад, нехай регістр CS містить 04AFh (дійсна адреса 04AF0), а регістр IP містить 0023h, тоді

```
          CS:     04AF0
          IP:      0023
Адреса команди:   04B13
```

Якщо, наприклад, за адресою 04B13 знаходиться команда

Адреса 04B13: `A11200  mov AX,[0012]`

то у пам'яті за адресою 04B13 знаходиться перший байт команди. Процесор отримує доступ до цього байту і по коду команди (A1) визначає довжину команди – три байта.

Для доступа до даних зі зміщенням [0012] процесор визначає адресу, виходячи з вмісту регістру DS (як правило) і зміщення в операторі команди. Якщо DS містить 04B1h (реальна адреса 04B10), то результуючиа адреса даних визначається наступним чином:

```
          DS:     04B10
Зміщення:          0012
Адреса даних:     04B22
```

Якщо, наприклад, за адресою 04B22 знаходиться число 24, а за 04B23 – число 01, то процесор вибирає значення 24 і розміщує його у регістр AL, а значення 01 – у регістр AH. Таким чином, регістр AX буде мати значення 0124. У процесі вибірки кожного байту команди процесор збільшує значення регістр IP на одиницю (інкрементує) і до початку виконання наступної команди у вище наведеному прикладі IP буде містити зміщення 0026. Таким чином, процесор тепер готовий для виконання наступної команди, яку він отримує із адреси, яка знаходиться у регістрі CS (04AF0) зі зміщенням, значення якого знаходиться у регістрі IP (0026), тобто 04B16.

Парна адресація

Сучасні процесори функціонують більш, оскільки у них у програмі забезпечується доступ до слів, розміщеним по парних адресах.

Наприклад, нехай команда повинна виконати вибірку слова, яке починається з адреси 04B23 і завантажити його у регістр AX:

```
Пам’ять:        |xx|24|01|xx|
                          |
Адреса :   04B23
```

Спочатку процесор отримує доступ до байтів за адресою 4B22 та 4B23 і пересилає байт з чарунки 4B23 у регістр AL. Потім він отримує доступ до батів за адресами 4B24 та 4B25 і пересилає байт з чарунки 4B23 у регістр AH. У результаті регістр AX буде містити 0124. семблер має директиву EVEN, яка викликає вирівнювання даних і команд на парні адреси пам’яті.

Визначення розміру пам’яті

BIOS має у своїй ROM підпрограму, яка здійснює визначення об’єму пам’яті. Можна викликати дану функцію BIOS перериванням 12h, в результаті чого BIOS поверне у регістр AX розмір пам’яті у Кбайтах.

```
E CS:100 CD 12 CB
```

Результат виконання підпрограми BIOS:

```
-T
AX=0280  BX=0000  CX=0000  DX=0000  SP=FFB4  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=09AA     NV UP DI PL NZ NA PE NC
0210:09AA C4C4          LES     AX,SP
-T

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFB4  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=064C     NV UP DI PL NZ NA PE NC
0210:064C 1E            PUSH    DS
-T

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFB2  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=064D     NV UP DI PL NZ NA PE NC
0210:064D 50            PUSH    AX
-T

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFB0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=064E     NV UP DI PL NZ NA PE NC
0210:064E B84000        MOV     AX,0040
-T

AX=0040  BX=0000  CX=0000  DX=0000  SP=FFB0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=0651     NV UP DI PL NZ NA PE NC
0210:0651 8ED8          MOV     DS,AX
-T

AX=0040  BX=0000  CX=0000  DX=0000  SP=FFB0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=0653     NV UP DI PL NZ NA PE NC
0210:0653 F70614030024  TEST    WORD PTR [0314],2400              DS:0314=3000
-T

AX=0040  BX=0000  CX=0000  DX=0000  SP=FFB0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=0659     NV UP DI PL NZ NA PE NC
0210:0659 754F          JNZ     06AA
-T

AX=0040  BX=0000  CX=0000  DX=0000  SP=FFB0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=06AA     NV UP DI PL NZ NA PE NC
0210:06AA 58            POP     AX
-T

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFB2  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=06AB     NV UP DI PL NZ NA PE NC
0210:06AB 1F            POP     DS
-T

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFB4  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0210  IP=06AC     NV UP DI PL NZ NA PE NC
```

```
   0210:06AC CF            IRET
-T

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFBA  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=0105  IP=0102     NV UP DI PL NZ NA PE NC
0105:0102 CB            RETF
-T

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFBE  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=3002  IP=0210     NV UP DI PL NZ NA PE NC
3002:0210 0000          ADD     [BX+SI],AL                          DS:0000=F8
```

Як видно, регістр AX містить розмір пам'яті у шістнадцятковому форматі. Для виходa із підпрограми BIOS і повернення у програму використовується команда IRET.

Спеціальні засоби відлагоджувача

- ***Команда A*** (Assembler) переводить Debug у режим прийому команд ассемблера і перетворення їх у машинні коди. Наприклад,

```
-A 100
3002:0100 MOV AL,25
3002:0102 MOV BL,32
3002:0104 ADD AL,BL
3002:0106 RET
```

Після натискання <Enter> без команди у рядку Debug припиняє прийом команд. Покроково виконаємо введену програму для перевірки

```
-T
AX=0225  BX=0032  CX=0000  DX=0000  SP=FFC0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=3002  IP=0102   NV UP DI PL NZ NA PO NC
3002:0102 B332          MOV     BL,32
-T
AX=0225  BX=0032  CX=0000  DX=0000  SP=FFC0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=3002  IP=0104   NV UP DI PL NZ NA PO NC
3002:0104 00D8          ADD     AL,BL
-T
AX=0257  BX=0032  CX=0000  DX=0000  SP=FFC0  BP=0000  SI=0000  DI=0000
DS=0040  ES=0B2B  SS=0B2B  CS=3002  IP=0106   NV UP DI PL NZ NA PO NC
3002:0106 C3            RET
```

- ***Команда U*** (Unassembler) виводить на екран машинні коди команд ассемблера. Необхідно задати адреси першої і останньої команди, які необхідно переглянути. Перевіримо код програми, яку ввели за допомогою команд *A*.

```
-U 100,106
3002:0100 B025          MOV     AL,25
3002:0102 B332          MOV     BL,32
3002:0104 00D8          ADD     AL,BL
3002:0106 C3            RET
```

Збереження програми із відлагоджувача

Можна використовувати Debug для збереження програм на диск. Для цього використовується ***команда W***. Для цього необхідно виконати наступні дії:

— присвоїти програмі ім'я: ***N Name.com***;

— за допомогою команди *R* у регістр CX записати довжину файлу (у нашому прикладі 106 – 100 = 6 байт).

— дати Debug команду запису файлу – ***W***.

Для прикладу, збережемо введену раніше програмку:

```
-N dorensky.com
-R CX
CX 0000
:6
-W
Запис 00006 байт
```

Якщо ж програму було завантажено з диску та необхідно зберегти зроблені у ній зміни, то треба дати Debug тільки команду *W*. Наприклад, зробимо зміни у програмі KNTU.com і збережемо їх:

```
-U 100,106
0B2B:0100 B025          MOV     AL,25
0B2B:0102 B332          MOV     BL,32
0B2B:0104 00D8          ADD     AL,BL
0B2B:0106 C3            RET
-A 102
0B2B:0102                                               MOV BL,10
0B2B:0104
-U 100,106
0B2B:0100 B025          MOV     AL,25
0B2B:0102 B310          MOV     BL,10
0B2B:0104 00D8          ADD     AL,BL
0B2B:0106 C3            RET
-W
Запис 00006 байт
```

ЗАВДАННЯ

1. Розробити блок-схему алгоритма і програму, яка виводить на екран ПК:
   1.1 Тип персонального комп'ютера IBM PC (див. табл. 1.2);
   1.2 Дату виготовлення BIOS у повному форматі прописом "*число місяць рік*" (наприклад, "01 лютого 2013 року");
2. Результати роботи програм 1.1-1.2 перевірити за допомогою дебагера (debbug.exe).

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Поясніть суть і призначення BIOS.
2. Що таке переривання? Які є види переривань?
3. Що таке вектор переривань, таблиця векторів переривань? Для чого вони служать і де знаходяться?
4. За допомогою яких стандартних функцій ТС можна звертатись до портів в/в?
5. Чим чарунка пам'яті відрізняється від порту?
6. Що таке дебагер? Де знаходиться дебагер у ОС Windows?
7. Основні команди дебагера, їх формати і призначення.
8. Що таке "сегмент" та "зміщення" і для чого вони використовуються?
9. Якими стандартними функціями ТС можна звернутись до чарунки пам'яті (R/W)?
10. За допомогою якої стандартної функції ТС можна згенерувати задане переривання?

ПРИКЛАД ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №1

МЕТА: Ознайомитись з основами системного програмування, навчитись програмним шляхом одержувати інформацію про системні характеристики комп'ютера IBM PC AT.

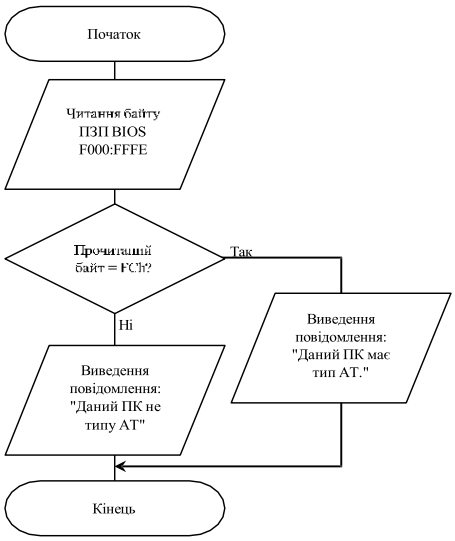
ЗАВДАННЯ

1. Написати програму, яка визначає, чи даний ПК відноситься до типу IBM PC AT.
2. Написати програму, яка виводить на екран дату виготовлення BIOS.
3. Перевірити результати виконання програм 1 та 2 за допомогою дебагера.

ВИКОНАННЯ

1. Як відомо, існує багато моделей ПК. Є дві можливості визначення моделі та отримання деякої інформації про конфігурацію комп'ютера: 1) Прочитати дану інформацію з чарунок ПЗП BIOS; 2) Викликати одну з функцій переривання 15h, яка повертає адресу таблиці конфігурації. У постійній пам'яті BIOS за адресою F000:FFFE міститься байт, значення якого є ідентифікацією типу комп'ютер. Прочитати цю чарунку пам'яті можна за допомогою функції TC peekb(). Якщо її значення FCh – ПК типу AT.

Блок-схема алгоритму



Лістинг програми

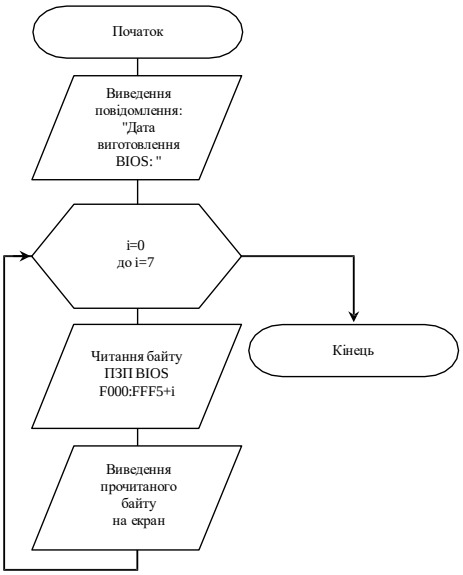
```
#include <stdio.h>
#include <conio.h>

int main (void)
{
     unsigned char KOD;
     KOD = peekb(0xF000,0xFFFE);
     if (KOD == 0xFC)
        printf("Даний ПК має тип AT.\n");
     else
        printf("Даний ПК не типу AT.\n");
     return 0;
}
```

Результат роботи програми:

```
Даний ПК має тип AT.
```

2. За адресою F000:FFF5 ПЗП BIOS міститься дата виготовлення BIOS. Вона записана у коді ASCII та займає 8 байт. Її формат американський: `MM/DD/YY`.

Блок-схема алгоритму



Лістинг програми

```
#include <stdio.h>
#include <conio.h>

int main (void)
{
     unsigned char DATA;
     int i;
     printf("Дата виготовлення BIOS: ");
     for (i=0;i<8;i++)
     {
        DATA = peekb(0xF000,0xFFF5+i)
        printf("%c",DATA);
     }
     return 0;
}
```

Результат роботи програми:

```
Дата виготовлення BIOS: 03/28/06
```

3. Системну програму debug.exe для перевірки результатів роботи програм 1 та 2 можна запустити за допомогою команди debug в меню Пуск → Выполнить або запустити її з системної папки WINDOWS\system32. За допогою команди D дебагера читаємо чарунки ПЗП BIOS: F000:FFFE – тип ПК, F000:FFF5-F000:FFFD – дата виготовлення BIOS.

Результат роботи дебагера:

```
-d f000:fffe
F000:FFF0                                                FC 00                .. -
d f000:fff5
F000:FFF0                30 33 2F-32 38 2F 30 36 00 FC 00                03/28/06..
_
```

Код типу IBM PC AT відпов. табл. 1.2

Дата виготовлення BIOS

Лабораторна робота № 2

ТЕМА: **Програмування інтервального таймера в IBM PC.**

МЕТА: Набути навичок з програмування мікросхеми i8253/i8254, використання інтервального таймера на платі IBM PC / AT для розв'язку інженерних задач та розробки системних функцій.

Короткі теоретичні відомості

Сучасні комп'ютери оснащено двома підсистемами таймерів, які паралельно відраховують поточний час. Один таймер розміщено у мікросхемі з низьким споживання енергії (КМОП-мікросхема), яка після вимкнення живлення комп'ютера продовжує функціонувати, отримуючи енергію від вбудованого у комп'ютер аккумулятора (батареї). Цей таймер, як правило, називають *годинником реального часу* (RTC), більш детально з який познайомимось у лабораторній роботі №3. Інший таймер, реалізований мікросхемою i8253 (вітчизняний аналог – КР580ВН53, в АТ – i8254), який надалі будемо називати *інтервальним* (іноді в літературі – системним), працює, як і решта вузлів комп'ютера, тільки коли ПК увімкнений. Його канал 0 генерує сигнали з частотою приблизно 18,206 Гц, яка викликає апаратні переривання рівня 0 (вектор 08h). Обробник даного переривання, який входить у систему BIOS, з кожним перериванням інкрементує вміст 4-байтної чарунки за адресою 40h:6Ch, яка розміщена у області даних BIOS і називається *таймером BIOS* або *системним таймером*. В процесі початкового завантаження комп'ютера програма BIOS чатає показники RTC (години, хвилини та секунди) і, перетворивши їх в кількість секунд від початку поточної доби, множить отриману велечину на 18,206, щоб отримати поточний час, виражене у кількості тактів системного таймера. Ця величина записується у чарунку пам'яті за адресою 40h:6Ch, яка у подальшому інкрементується, що є паралельним з RTC відліком часу системи, поки комп'ютер увімкнений. Саме з цієї чарунки пам'яті системні функції читають поточний час.

Інтервальний 3-канальний таймер, який, як правило, входить до складу багатофункціональної мікросхеми програмованого периферійного контролера, забезпечує в IBM PC три функції: відлік системного часу, керування регенерацією динамічної пам'яті та генерація звука в динаміку комп'ютера (таблиця 2.1). В останній процедурі приймає участь один з портів периферійного контролера (порт 61h). На рисунку 2.1 наведено спрощену схему взаємодії цих вузлів.

Таблиця 2.1 – Призначення каналів системного таймера в IBM PC

Канал	Призначення	Режим роботи
0	системний годинник (IRQ0)	3, лічильник = 0 (65536)
1	запит для каналу 0 DMA	2, лічильник=18
2	генерація звуку	задається прикладною програмою

Підсистема таймера працює незалежно від процесора (паралельно з ним) від власного генератора, який генерує сигнали з частотою 1,19318 МГц, тобто кожний такт має тривалість 0,84 мксек. До складу таймера входять буфер шини даних,

схема управління введенням-виведенням та три незалежних канали, кожний з яких містить регістр режиму, схему управління каналом, буфер та 16- розрядний лічильник. Фіксатори каналів (регістри константи перерахунку) адресуються через порти 40h, 41h, 42h відповідно (дуже легко запам'ятати адреси: порт 4**0**h – канал **0**, порт 4**1**h – канал **1**, порт 4**2**h – канал **2**), 43h – порт керуючого слова (таблиця 2.1).

Таблиця 2.2 – Адреси портів системного таймера та їх призначення

Адреса	Операція	Призначення
40h	Запис	Завантаження лічильника каналу 0
	Читання	Читання лічильника каналу 0
41h	Запис	Завантаження лічильника каналу 1
	Читання	Читання лічильника каналу 1
42h	Запис	Завантаження лічильника каналу 2
	Читання	Читання лічильника каналу 2
43h	Запис	Запис керуючого слова системного таймера

Програмування всіх каналів таймера здійснюється однаково: в регістр команд (порт 43h) записується керуюче слово, формат якого наведено на рисунку 2.1, потім у фіксатор завантажується константа перерахунку (один або два байти) і відразу після чого канал починає роботу.

Під час увімкнення комп'ютера у регістр константи перерахунку канала 0 системного таймера BIOS записує максимально можливе число 65535 (FFFFh, 65536 декрементів), в результаті чого сигнали на виході OUT0 генеруються з частотою 18,2 Гц ($\frac{1193181\,Гц}{65535} = 18{,}206\,Гц$, тобто кожні 55 мсек). Дані сигнали, як вже відмічалось вище, поступають на IRQ0 контролера переривань, викликаючи переривання 08h, яке обробляється підпрограмою BIOS і здійснює відлік поточного часу (шляхом інкременту таймера BIOS – чарунки пам'яті за адресою 40h:6Ch).

Переривання від канала 0 можна використовувати для часової синхронізації програми (наприклад, для періодичного виведення на екран певної інформації).

Канал 1 таймера налаштований BIOS для регенерації динамічної пам'яті, тому його перепрограмування може призвести до порушення даного процесу і втрати даних ОЗП.

Вхід GATE каналів 0 та 1 завжди мають високий рівень, тому рахунок на цих каналах дозволяється завжди.

Вихід канал 2 зв'язаний з динаміком та використовується для генерації звука (рисунок 2.1). Вхід GATE каналу 2 керується бітом 0 порту PB інтерфейса 8255, який зв'язаний з портом 61h.

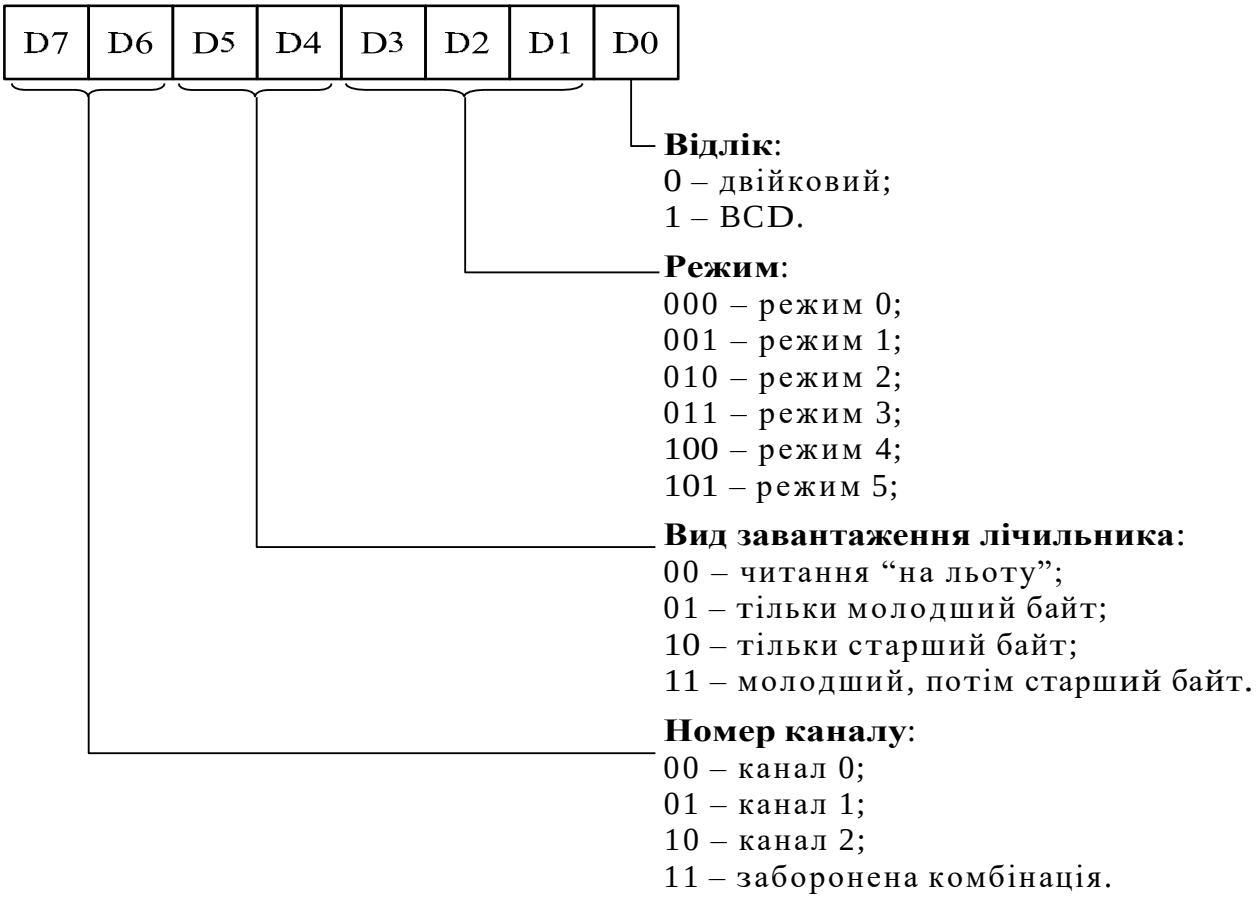


Рисунок 2.1 – Формат керуючого слова таймера (порт 43h)

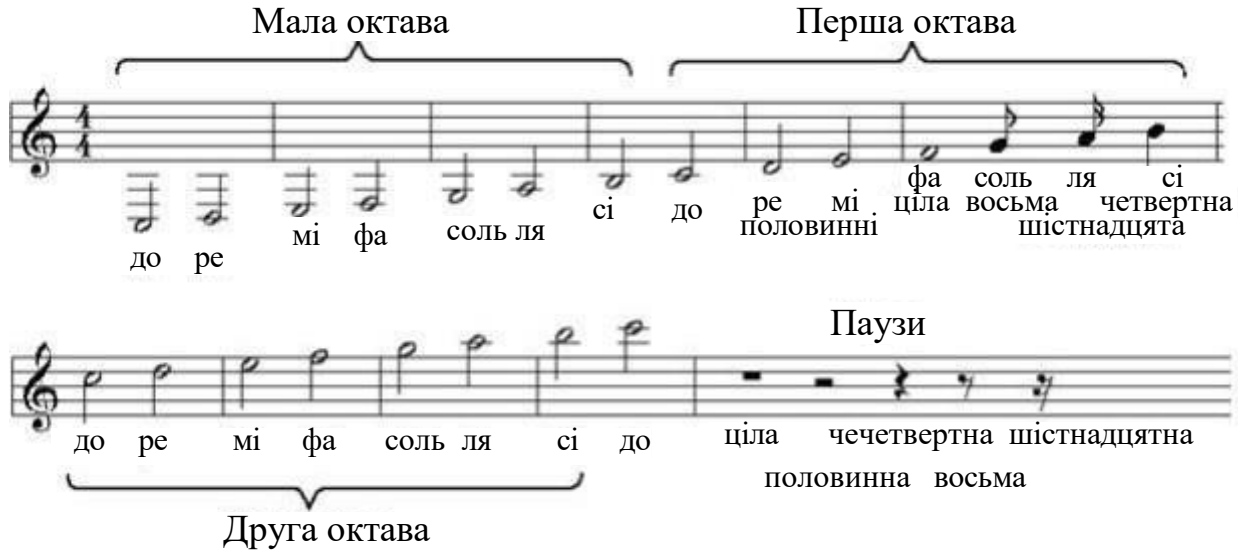
На вхід звукогенератора надходить логічне "і" двох сигналів: виходу OUT 2-го каналу таймера та змісту біта 1 порта PB інтерфейса 8255. Тому найпростіший спосіб генерації звука полягає в програмуванні канала 2 системного таймера таким чином, щоб він генерував прямокутний імпульс вказаної частоти, що належить звуковому діапазону від 20 Гц до 20 КГц. Змінюючи вміст регістра констати перерахунку (фіксатора), можна змінювати частоту сигналів, які поступають на динамік, від 18,2 Гц до 1,19 МГц (реально для збудження звука можна використовувати частоти не вище 10КГц). Для цього необхідно використовувати режим 3 з відповідним початковим значенням лічильника. Якщо потім встановити біти 0 та 1 порта PB, то імпульс почне надходити на вхід звукогенератора (біт 0 – вхід GATE канала 2, що дозволяє рахунок, а біт 1 – дозвіл видачі OUT на вхід звукогенератора). Для вимкнення звуку достатньо скинути біти 0-1 в PB. Перевага цього методу полягає в тому, що, запустивши генерацію звуку, ЦП може виконувати інші дії. Значення лічильника каналу 2 обчислюється за формулою

$$n = \frac{F}{f}, \tag{1}$$

де n – значення константи перерахунку (фіксатора),
F – тактова частота генератора системного таймера (1193181 Гц),
f – вихідна частота канала (частота OUT).

Таблиця 2.3 – Частоти для генерації звуків відповідних нот

До	До#	Ре	Ре#	Мі	Фа	Фа#	Соль	Соль#	Ля	Ля#	Сі	Октава
1	2	3	4	5	6	7	8	9	10	11	12	Контроктава
33	35	37	39	41	44	46	49	52	55	58	62	
13	14	15	16	17	18	19	20	21	22	23	24	Велика
65	69	73	78	82	87	92	98	104	110	117	123	
25	26	27	28	29	30	31	32	33	34	35	36	Мала
131	139	147	156	165	175	185	196	208	220	233	247	
37	38	39	40	41	42	43	44	45	46	47	48	Перша
262	277	294	311	330	349	370	392	415	440	466	494	
49	50	51	52	53	54	55	56	57	58	59	60	Друга
523	554	587	622	659	698	740	784	831	880	932	988	
61	62	63	64	65	66	67	68	69	70	71	72	Третя
1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976	
73	74	75	76	77	78	79	80	81	82	83	84	Четверта
2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951	



Другий спосіб генерації звука полягає в тому, щоб генерувати звукові імпульси не виходом таймера, а встановленням та скидом біта 1 в регістрі PB. Для керування затримкою можна використовувати програмний цикл.

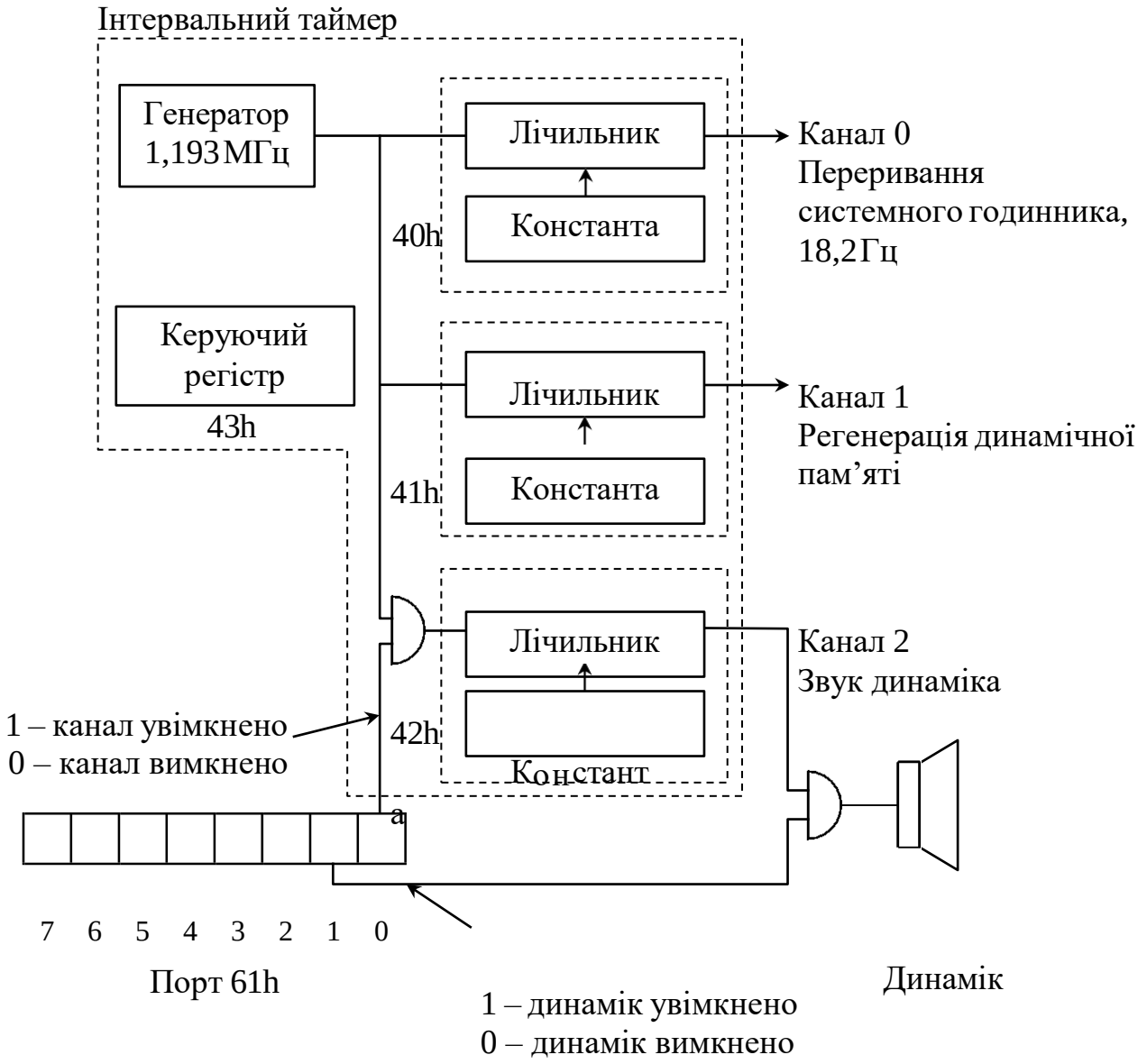


Рисунок 2.2 – Елементи комп'ютера, які приймають участь у генерації звука

Розглянемо приклад. В лістингу 2.1 наведено лістинг програми, яка перепрограмовує системний таймер (канал 0) таким чином, що відлік системного часу прискорюється в десять разів.

Лістинг 2.1 – Програмування системного таймера: прискорення системного часу у 10 разів шляхом збільшення частоти OUT0 з частоти 18,206 Гц до 182,06 Гц

```
#include <stdio.h>
#include <dos.h>

int main (void)
{
   printf("ПРОГРАМА ПРИСКОРЕННЯ У 10 РАЗІВ СИСТЕМНОГО ЧАСУ.");
   outportb(0x43,0x36); //запис керуючого слова (00110110b)
   outportb(0x40,0x99); //запис молодшого байту константи 1999h
   outportb(0x40,0x19); //запис старшого байту константи 1999h
   printf("\nКАНАЛ 0 СИСТЕМНОГО ТАЙМЕРА ПЕРЕПРОГРАМОВАНО.");
   return 0;
}
```

Для того, щоб прискорити відлік системного часу у 10 разів необхідно у 10

разів частіше викликати обробник 08h (IRQ0), який виконує інкремент таймера

BIOS. Відповідно, треба збільшити вихідну частоту канала 0 в 10 разів, тобто до частоти 182,067 Гц (у нормальному відліку, як відомо, вона рівна 18,2067 Гц). Після цього час прискориться в 10 разів.

За допомогою формули (1) обчислимо необхідну константу перерахунку для вихідної частоти OUT 182 Гц:

$$n = \frac{F}{f} = \frac{1193181}{182{,}067} = 6553 = 1999h\,.$$

Отже, записавши керуюче слово 00110111B (канал 0; завантаження молодшого, потім старшого байту; режим 3; двійковий відлік), завантаживши у регістр-фіксатор константу перерахунку, за допомогою, наприклад, Norton Commander можна спостерігати, що системний час прискорився у 10 разів (прискорилось блимання ":"). Тобто а саме 1 хв системного часу проходить за 6 с реального.

ЗАВДАННЯ

Розробити блок-схему алгоритма та програму, яка виконує наступні функції:

1 Перепрограмовує інтервальний таймер таким чином, щоб хід системного часу прискорився у *12N* разів, де *N* – номер студента у списку журнала академгрупи.

2 За допомогою канала 2 інтервального таймера визначає час виконання 5N ітерацій циклу `for (int i=0; i<=5*N; i++) printf("%5d | ", i);` (визначити кількість CLK), де *N* – номер студента у списку журнала академгрупи.

3 Виведення стану лічильника каналу 0 кожні 3 секунди до натиснення користувачем клавіші Esc.

4 За допомогою системного динаміка генерує сигнал тривалістю $^1\!/_2 N$ секунд, де *N* – номер студента у списку журнала академгрупи.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1 Призначення і функції інтервального таймера у комп'ютері. На базі якої мікросхеми його реалізовано в IBM PC? Основні характеристики мікросхеми.
2 Яке основне призначення ТІ в обчислювальній системі?
3 Яку роль відіграє ТІ у формуванні системного часу в IBM PC?
4 З якою тактовою частотою працює кварцовий генератор? Де він знаходиться?
5 Скільки режимів роботи таймера? Їх основні характеристики й особливості.
6 Призначення каналів інтервального таймера в IBM PC AT.
7 Який обробник BIOS в IBM PC оброблює збудження виходу канала 0?
8 За допомогою якого переривання від канала 0 можна організувати часову синхронізацію програми?
9 Яким чином відбувається генерація звуку в IBM PC?
10 Чому читання комірки за адресою 40h:6Ch (046Ch) за допомогою дебагера кожного разу дає різні значення?

ПРИКЛАД ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №2

МЕТА: Набути навичок у програмуванні мікросхеми i8253/i8254, використання інтервального таймера на платі IBM PC AT для вирішення прикладних задач і системних функцій.

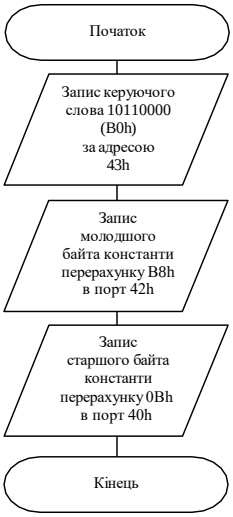
ЗАВДАННЯ

4. Написати програму, яка перепрограмовує канал 2 інтервального таймера на режим роботи 0 з константою перерахунку 3000.
5. Написати програму, яка шляхом перепрограмування відповідного канала інтервального таймера прискорює хід системного часу у 10 разів.

ВИКОНАННЯ

1. Для перепрограмування канала 2 системного таймера на режим 0 і константу перерахунку 3000 необхідно: записати керуюче слово 10110000 (B0h); так як 3000 – двобайтове (BB8h), записати спочатку молодший B8h, потім старший байт константи – 0Bh.

Блок-схема алгоритму



Лістинг програми

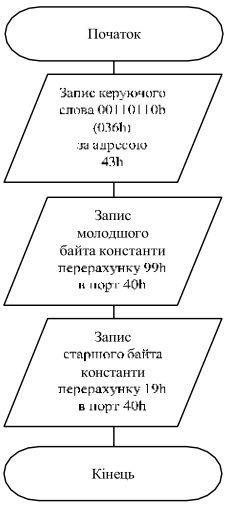
```
#include <stdio.h>
#include <conio.h>

int main (void)
{
     outportb(0x43,0xB0);
     outportb(0x40,0xB8);
     outportb(0x40,0x0B);
     printf("Канал 2 системного таймера
              успішно
              перепрограмовано.\n");
     return 0;
}
```

2. Відповідно до рисунку 2.2, для прискорення ходу системного часу у 10 разів, необхідно перепрограмувати канал 0 інтервального таймера (який відповідає за ведення системного часу за допомогою виклику переривання 08h) таким чином, щоб збільшити у 10 разів вихідну частоту OUT0, тобто становила 182,06 Гц (18,206·10=182,06). Таким чином, необхідно обчислити контанту перерахунку n для цієї частоти:

$$n = \frac{F}{f} = \frac{1193181}{182{,}067} = 6553 = 1999h,$$

де F – тактова частота генератора системного таймера (1193181 Гц), f – вихідна частота канала (частота OUT).

Блок-схема алгоритму



Лістинг програми

```
#include <stdio.h>
#include <dos.h>

int main (void)
{
     outportb(0x43,0x36); //кер. слово
     outportb(0x40,0x99); //молодний байт
     outportb(0x40,0x19); //старший байт
     printf("Системний час прискорено в
              10 разів.\n");
     return 0;
}
```

Перевірити результат роботи програми 2 можна, наприклад, за допомогою оболонки VC або FAR, у яких у правому верхньому кутку виводиться системний час. Після запуску програми 2 має пришвидчитись хід годинника (рисунок 2.3).

Рисунок 2.3

Лабораторна робота № 3

ТЕМА: Програмування годинника реального часу (RTC) в IBM PC.

МЕТА: Набути навичок з програмування годинника реального часу (RTC) та CMOS-пам’яті для розв’язку інженерних задач.

Короткі теоретичні відомості

Сучасні комп’ютери оснащено двома підсистемами таймерів, які паралельно відраховують поточний час. Один таймер розміщено у мікросхемі з низьким споживання енергії (КМОП-мікросхема), яка після вимкнення живлення комп’ютера продовжує працювати, отримуючи енергію від вбудованого у комп’ютер аккумулятора (батареї). Цей таймер, як правило, називають *годинником реального часу* (RTC); окрім кварцевого генератора і систем керування він має внутрішню пам’ять, у якій зберігається й постійно нарощується значення поточного часу. Оновлення часу здійснюється щосекунди, причому ця операція виконується на апаратному рівні, не задіюючи в даному процесі ні процесор, ні оперативну пам’ять.

Підсистема ГРЧ включає в себе контролер та невеликий блок пам’яті об’ємом 64 байт. Перші 14 байт використовуються для відліку часу; решта 50 байт зберігають інформацію про конфігурацію системи. Підсистема забезпечує наступні функції:

– відлік поточного часу з точністю до 1 с;

– роботу будильника, який у встановлений час генерує сигнал переривання лінією IRQ8, закріпленним за вектором 70h;

– режим періодичних переривань (лінією IRQ8), частоту яких можна програмно налаштовувати у межах від 2 Гц до 8 КГц;

– зберігання даних про конфігурацію системи (об’єм базової та розширеної пам’яті, типи магнітних дисків і т.д.);

– зберігання даних прикладних програм.

У таблиці 3.1 наведено призначення окремих байтів КМОП-пам’яті (0-3Fh).

Таблиця 3.1 – Адресний простір пам’яті КМОП-мікросхеми

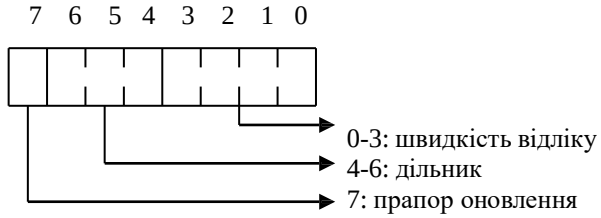
Адреса поля	Кількість байт	Призначення
1	2	3
00h	1	Секунди в BCD
01h	1	Секунди будильника в BCD
02h	1	Хвилини в BCD
03h	1	Хвилини будильника в BCD
04h	1	Години в BCD
05h	1	Години будильника в BCD
06h	1	День тижня (може бути відсутній)
07h	1	Число в BCD

Продовженя таблиці 3.1

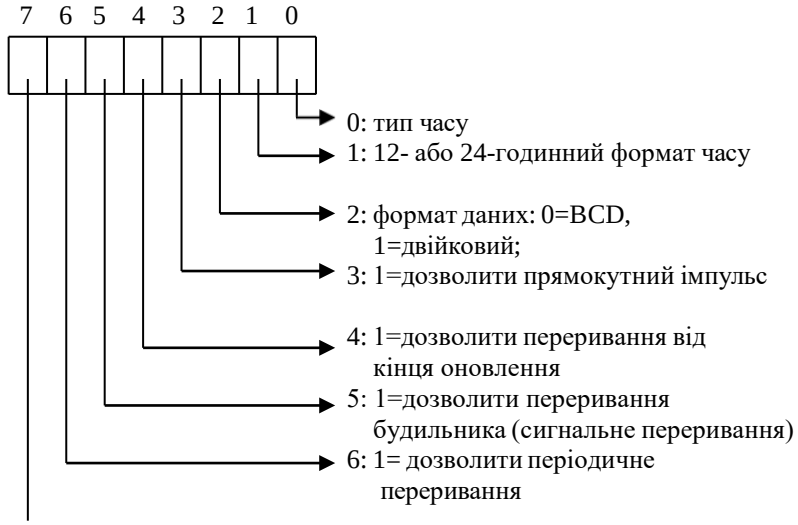
1	2	3
08h	1	Місяць (січень – 1, лютий – 2 і т.д.) в BCD
09h	1	Рік, молодші дві цифри в BCD
0Ah	1	**Регістр A**
0Bh	1	**Регістр B**
0Ch	1	**Регістр C**
0Dh	1	**Регістр D**
0Eh	1	Байт діагностування
0Fh	1	Байт кода скиду процесора
10h	1	Типи HDD (якщо менше 15)
11h	1	Зарезервовано
12h	1	Типи дискет
13h	1	Зарезервовано
14h	1	Склад встановленого обладнання
15h	2	Об’єм базової пам’яті, Кбайт
17h	2	Об’єм розширеної пам’яті, Кбайт
19h, 1Ah	2	Тип першого HDD (якщо більше 15)
1Bh	14	Зарезервовано
2Eh	2	Контрольна сума байтів 10h-2Dh
30h	2	Об’єм розширеної пам’яті, Кбайт
32h	1	Рік, перші дві цифри в BCD
33h	1	Системна інформація
34h	12	Зарезервовано

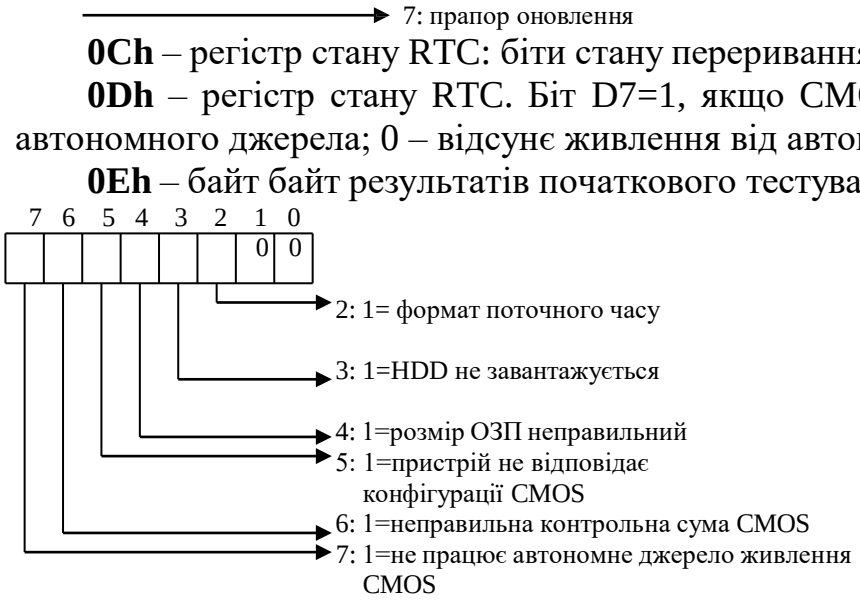
Байти з номерами 0Ah, 0Bh, 0Ch та 0Dh виконують функції керуючих регістрів.

0Ah – регістр стану RTC:



0Bh – регістр стану RTC:





0Ch – регістр стану RTC: біти стану переривання (тільки чатання).

0Dh – регістр стану RTC. Біт D7=1, якщо CMOS отримує живлення від автономного джерела; 0 – відсунє живлення від автононого джерела.

0Eh – байт байт результатів початкового тестування.

0Fh – байт стану перезавантаження. Цей байт зчитується після скиду ЦП, щоб визначити, чи не було скиду, викликаного виведенням 80286 з захищеного режима. Він може мати наступні значення:

0 – гарячий рестарт (Ctrl-Alt-Del) або неочікувана зупинка;

1 – зупинка після визначення розміру ОЗП;

2 – зупинка після тестування пам’яті;

3 – зупинка після виявлення помилки паритета пам’яті;

4 – рестарт за запитом початкового завантажувача;

5 – рестарт за скидом контролера переривань та JMP FAR PTR [0:467h];

6, 7, 8 – зупинка після теста захищеного режиму;

9 – рестарт за JMP FAR PTR [0:467h].

Звертання до байтів КМОП-пам’яті здійснюється за допомогою портів 70h та 71h у два етапи: спочатку в порт 70h записується номер необхідного байта пам’яті; потім через порт 71h виконується читання (in) чи запис (out) байта пам’яті.

Наприклад, прочитаємо та виведемо на екран поточний рік. На лістингу 3.1 наведено вихідний текст програми.

Лістинг 3.1 – Програма читаня з RTC інформації про об’єм розширеної пам’яті

```
#include <stdio.h>
#include <dos.h>

int main (void)
{
    unsigned char XMS_H,XMS_L;
    int XMS;
    outportb(0x70,0x17);
    XMS_L=inportb(0x71);
    outportb(0x70,0x18);
    XMS_H=inportb(0x71);
    XMS=XMS_H;
    XMS=XMS>>8;
    XMS+=XMS_L;
    printf("Об’єм розширеної пам’яті = %dKb",XMS+1024);
```

```
        return 0;
}
```

Після виконання програми 2-байтна змінна XMS матиме значення на 1024 менше, ніж повний об'єм пам'яті (у Кбайтах), встановленої на комп'ютері. Аналогічно можна отримати, наприклад, об'єм базової пам'яті (байти 15h та 16h).

Описаним вище способом можна звертатись до всіх чарунок CMOS- пам'яті, за винятком перших десяти. Справа в тому, що КМОП-мікросхема один раз за секунду виконує корекцію поточного часу та перевірку стану будильника. На час корекції ділянка КМОП-пам'яті, яка відноситься до годин, календаря та будильника, відключається від системної магістралі та стає недоступним для програмного звертання. На це й же час встановлюється біт D7 регістра A. Тому перед зверненням до адрес 00h-09h CMOS-пам'яті необхідно спочатку дочекатись скиду біта D7 регістра A, що вказує на завершення процесу корекції і тільки потім здійснювати звернення до пам'яті. В лістингах 3.2-3.3 продемонстровано читання чарунок КМОП-пам'яті з очікуванням закінчення цикла корекції.

Лістинг 3.2 – Програма читання з RTC та виведення на екран поточного року

```
#include <stdio.h>
#include <dos.h>

int main (void)
{
    unsigned char YearH, YearL;
    do{
         outportb(0x70,0x0A);
    }while ((inportb(0x71)&0x80)==1);  //біт D7 рег. A
                                       // встановлено?
    outportb(0x70,0x32);  //байт 32h - старші 2 цифри року
    YearH=inportb(0x71);
    outportb(0x70,0x09);  //байт 9h - молодші 2 цифри року
    YearL=inportb(0x71);
    printf("Поточний рік: %02x%02x",YearH,YearL);
    return 0;
}
```

Лістинг 3.3 – Програма читання з RTC та виведення на екран поточної дати

```
#include <stdio.h>
#include <dos.h>

int main (void)
{
    unsigned char Day, Mon;
    do{
      outportb(0x70,0x0A);
    }while ((inportb(0x71)&0x80)==1); //корекцію завершено (D7
Ah)?
    outportb(0x70,0x07);  //07 – число в BCD (з табл. 3.1)
```

```
    Day=inportb(0x71);
    outportb(0x70,0x08);  //08 – місяць в BCD (з табл. 3.1)
    Mon=inportb(0x71);
    printf("Поточна дата: %02x.%02x",Day,Mon);
    return 0;
}
```

ГРЧ працює від внутрішнього кварцового генератора, частоту якого підібрано таким чином, щоб сигнали на його виході (після перерахунку) мають частоту точно 1 Гц (щосекунди). Ці сигнали використовуються для відліку поточного часу в байтах годин та календаря КМОП-пам'яті. Окрім постійного перерахунку, який забезпечує частоту 1 Г, в КМОП-схему включено ще вузол настроюваного перерахунку, вихідні сигнали якого поступають на лінію IRQ8, ініціалізуючи перілдичні переривання через вектор 70h. Коефіцієнт перерахунку і, відповідно, частоту періодичних переривань можна програмного налаштовувати, змінюючи біти D0-D3 регістра A. Для дозволу/заборони періодичних переривань використовується біт D6 регістра B.

Встановлення будильника здійснюється записом необхідного часу розбудження в байти секунд, хвилин та годин КМОП-пам'яті (див. таблицю 3.1). Під час запису в ці чарунки необхідно передбачити очікування кінця циклів корекції, як це показано у лістингу 3.2. Схеми таймера періодично порівнюють поточний час з часом, який записаний у байтах будильника, і при досягненні рівності збуджує сигнал сигнального переривання на лінії IRQ8. Для дозволу/заборони сигнальних переривань необхідно ініціалізувати біт D5 регістра B.

Якщо відповідні значення встановлено у всіх трьох байтах будильника, сигнальні переривання буде формуватись у вказаний час щодоби. Проте програмно можна встановити в одном або декількох байтах будильника "довільний" код – будь-яке число від C0h до FFh. Якщо довільний код встановлено у байті годин будильника, то сигнальні переривання збуджуються щогодини. Після запису довільного кода у байт годин та хвилин – сигнал генерується щохвилини, а за наявності довільного коду у всіх трьох чарунках – щосекунди.Особливість цього режима заключається в тому, що переривання формуються не просто заданої частоти, а у заданий момент кожної хвилини або кожної години. Наприклад, якщо у байті сукунди записано число 30h, а в байтах хвилин та годин – C0h, то переривання будуть збуджуватись точно на 30-й секунді кожної хвилини.

При обробці переивань від CMOS-мікросхеми необхідно враховувати, що сигнали, призначені для збудження переривань (сигнальних та періодичних), поступають на вхід контролера переивань не безпосередньо, а через розряд D7 регістра C, який виконує функції прапора переривань. Програма обробки переривань повинна скинути прапор переривання, інакше подальші надходження сигналів переривань будуть заблокованими. Скид прапора переривань здійснюється шляхом читання регістру C.

Для читання та зміни показників RTC передбачено переривання BIOS 1Ah, функції якого звертаються безпосереньо до КМОП-пам'яті і дозволяють не тільки отримати чи встановити час та дату, але й керувати будильником.

Основні функції переривання BIOS **1Ah**:

00h – отримання системного часу;
01h – встановлення системного часу;
02h – отримання часу від CMOS-годинника реального часу;
03h – встановлення часу у CMOS-годиннику реального часу;
04h – отримання дати від CMOS-календаря реального часу;
05h – встановлення дати у CMOS-календаря реального часу;
06h – встановлення будильника у CMOS-годиннику реального часу;
07h – відміна будильника у CMOS-годиннику реального часу.

Є ще переривання 4Ah, яке служить для перехоплення прикладною програмою сигнала від будильника RTC. INT 4Ah включено у системний обробник переривання 70h від будильника в RTC. Системний обробник цього переривання фактично виконує лиш команду IRET; прикладна програма може встановити власний обробник переривання 4Ah, який буде активізуватись сигналом будильника.

Як видно з таблиці 3.1, адреси CMOS 10h-2Dh захищено контрольною сумою, яка зберігається за адресою 2Eh. Тому зміна вмісту даних чарунок пам'яті необхідно супроводжувати зміною і контрольної суми.

Порт 70h використовується не тільки для індексування адреси CMOS-чарунки, але й для дозволу/заборони NMI (немасковане переривання). Якщо біт D7 скинуто, то NMI дозволено, якщо втсановлено – NMI заборонено.

ЗАВДАННЯ

3. Розробити блок-схему алгоритма та програму, яка:
   3.1 Виводить на екран інформацію про встановлений формат часу (12- чи 24-годинний) годинника реального часу (RTC).
   3.2 Виводить з комірок RTC на екран поточні дату й час, при чому назву місяця – прописом (наприклад, "`1 лютого 2013 року, 09:20:25`"), а час має продовжувати йти до натиснення користувачем клавіші Esc.
   3.3 Програмує будильник RTC на час, введений користувачем з клавіатури, та вмикає сигнальне переривання будильника (біт D_5 регістра 0Bh).
   3.4 Визначає, чи отримує CMOS-мікросхема живлення від автономного джерела (акумулятора) та виводить відповідну інформацію на екран.
   3.5 Записує у комірки CMOS-пам'яті за адресами 34h-3Fh рядок символів – прізвище студента.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Що таке годинник реального часу (RTC)? Чим забезпечується його енергоне-залежність?
2. Що таке CMOS-пам'ять? Який об'єм її пам'яті та особливості організації?
3. Яким чином здійснюється читання/запис CMOS-пам'яті ?

4. Яка кількість чарунок RTC?
5. Чому не бажано використовувати у прикладних програмах резерв CMOS-пам’яті для зберігання даних?
6. В якому форматі зберігаються дані RTC? Чому?
7. Поясніть функціональну взаємодію системного таймера, RTC та таймера ОС?
8. Що відбудеться, якщо акумулятор (батарея) RTC повністю розрядиться?
9. Яка особливість зберігання в RTC значення поточного року?
10. Чому чарунки CMOS з адресами 10h-2Dh захищені контрольною сумою, а решта – ні?

ПРИКЛАД ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №3

МЕТА: **Набути навичок програмування та роботи з годинником реального часу (RTC).**

ЗАВДАННЯ

6. Написати програму, яка виводить на екран встановлений час будильника RTC.

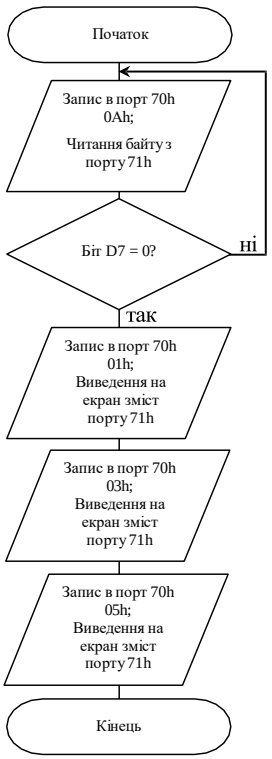
ВИКОНАННЯ

Для одержання значень годин, хвилин та секунд будильника RTC необхідно прочитати байти RTC 01h, 03h, 05h відповідно (табл. 3.1). Здійснюється доступ до даних КМОП-пам’яті за допомогою порту 70h та 71h за аглоритмом:

1) в порт 70h записати номер байта, з якогонеобхідно прочитати інформацію.

2) з порту 71h прочитати значення байта, номер якого записано в порт 70h.

Блок-схема алгоритму



Лістинг програми

```
#include <stdio.h>
#include <dos.h>

int main (void)
{
      unsigned char Day, Mon;
      do{
            outportb(0x70,0x0A);
      }while ((inportb(0x71)&0x80)==1);
      outportb(0x70,0x01);
      printf("Час будильника
              RTC:  %02x:",inportb(0x71)
              );
      outportb(0x70,0x03);
      printf("%02x:",inportb(0x71));
      outportb(0x70,0x05);
      printf("%02x",inportb(0x71));
      getch();
      return 0;
}
```

Результат роботи програми:

```
  Час    будильника    RTC:
00:00:00
```

Лабораторна робота №4

ТЕМА: Таймер операційної системи DOS та Windows.

МЕТА: Набути практичних навичок з програмування таймера операційної системи DOS та системного часу ОС Windows.

Короткі теоретичні відомості

Таймер операційної системи DOS

Комп'ютери IBM PC у своєму складі містять годинники системного та реального часу. Функціональна схема з'єднань пристроїв, які утворюють системний та реальний час ПК наведено на рис. 1.

Системний час утворюється операційною системою (ОС) комп'ютера за допомогою інтервального таймера (1), комірок пам'яті за адресою 046Ch-0470h (4) та годинника реального часу (5).

Для утворення системного часу в IBM PC використовується канал 0 інтервального таймера, який запрограмований на генерацію прямокутних сигналів з частотою 18,206 Гц (тіків). Вихід канала 0 (OUT0) з'єднано зі входим IR0 контролера переривань (2, рис. 1). Таким чином генерується апаратне переривання IRQ0. Контролер переривань видає адресу програми обробника цього переривання (INT 8h). Виконуючи цей обробник, центральний процесор (3) інкрементує комірки таймера ОС 046Ch-046Fh з частотою 18,206 Гц, а при досягненні кількості тіків, яка рівна добі (1572480 тіків), у комірці пам'яті за адресою 0470h обробник встановлює 1, яка є сигналом для ОС про виклик переривання INT 1Ah, яке здійснює скид значень комірок пам'яті 046Ch-046Fh (запис значення 0) та корекцію календаря. Таким чином, відбувається перехід системного таймера на наступну добу.

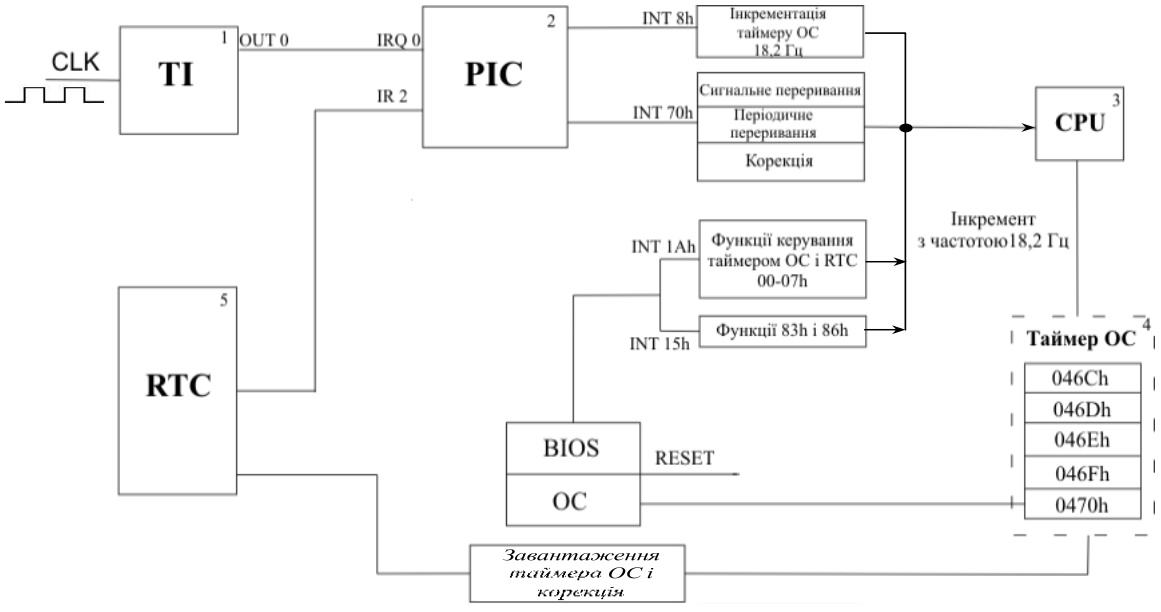


Рисунок 1 – Функціональна схема системного та реального часу на платі IBM PC:

1 – інтервальний таймер; 2 – контролер переривань; 3 – центральний процесор; 4 – таймер операційної системи; 5 – годинник реального часу (RTC).

При ввімкненні персонального комп'ютера (ПК) системний час встановлюється на основі даних годинника реального часу (RTC) і надалі працює незалежно від нього. Велика частина прикладних та системних програм використовує цей час для розв'язування різних задач. Наприклад, ОС записує на диск дату та час створення файлів, встановлює різного роду паузи під час роботи тощо. Прикладні програми можуть періодично виконувати певні команди. Наприклад, зберігати документ на диску через кожні дві хвилини. Крім того, різні системи керування використовують системний час для своїх потреб, наприклад, для визначення термінів різних операцій, планування руху транспорту тощо.

Реальний час комп'ютера IBM PC реалізовано на базі мікросхеми MC46818, яка живиться від акумулятора, що дозволяє контролювати час незалежно від стану ввімкнення/вимкнення ПК. RTC містить 64 комірки пам'яті, які дозволяють утворювати календар, годинник, будильник, який може видавати сигнал у встановлений програмістом час, а також зберігає інформацію про склад і налаштування комп'ютера.

Інтервальний таймер (2) реалізовано на базі мікросхеми i8253/i8254, яка є однокристальним програмуємим пристроєм, призначеним для отримання програмно-керованих часових затримок та генерації прямокутних імпульсів програмно-керованої частоти.

Програма обробки переривання INT 1Ah здійснює керування таймером ОС, а також встановлення та зчитування даних RTC. Для отримання часових затрмок і інтервалів використовуються функції 83h та 86h програмного переривання INT 15h.

Слід зазначити, що при вимкненні ПК дані годинника операційної системи втрачаються, а RTC, який живиться від автономного джерела, – продовжує відлік часу і календаря.

Структурну схему реалізації системного часу на в IBM PC наведено на рис. 2.

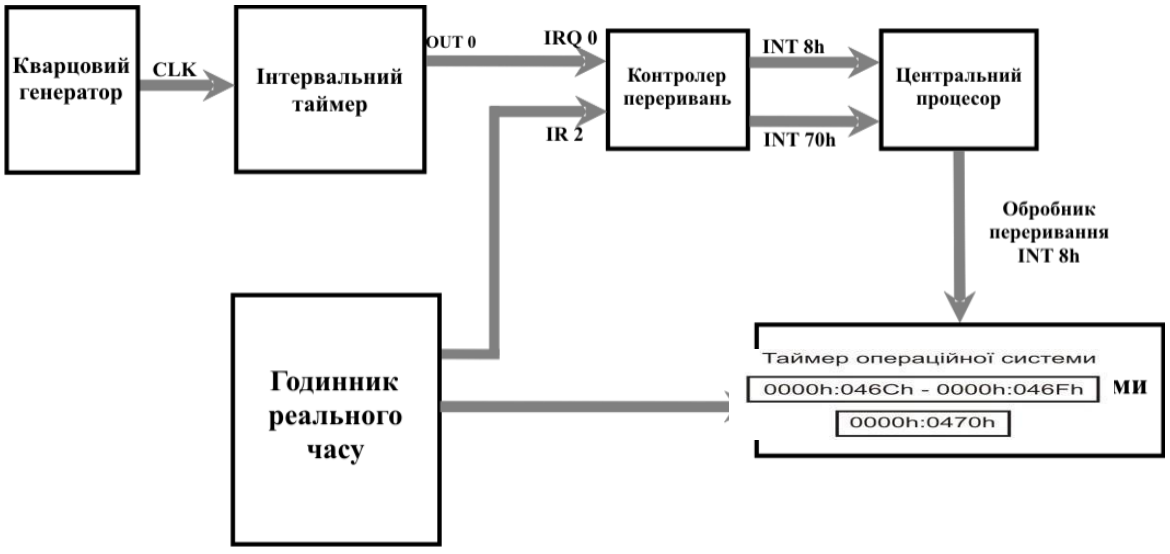


Рисунок 2 – Структурна схема системного часу на платі IBM PC

Таймер операційної системи (інколи його ще називають таймером BIOS) – це 32-розрядна комірка пам'яті, яка знаходиться за початковою адресою 0000:046Ch. Ці комірки пам'яті інкрементуються перериванням Int 8h, яке генерується запитом на контролер переривань IRQ0 від OUT0 інтервального таймера (рис. 1).

При досягненні значення цих комірок, яке рівне 24 год., ОС скидає таймер в 0 і встановлює 1 в комірці пам'яті 0470h. Це служить сигнатурою кінця доби і виклику переривання корекції таймера ОС.

На лістингу 1 наведено приклад читання поточного (системного) часу з таймера ОС та виведення його у правий куток екрана.

Лістинг 4.1 – Виведення системного часу DOS

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void main()
{
unsigned int hour, min, sec;
unsigned long ticks;
min=peek(0x0000,0x046c);
hour=peek(0x0000,0x046e);
ticks=hour;
ticks<<=16;
ticks|=min;
hour=ticks/65543;

min=(ticks%65543)/1092;
sec=((ticks%65543)%1092)/18.2;
printf("%02d:%02d:%02d",hour, min, sec);
}
```

читання 4-байтової комірки таймера ОС

перетворення тіків у год., хв., сек.

Системний та локальний час Windows

Після завантаження Windows встановлює значення часу, використовуючи годинник реального часу RTC. Для забезпечення роботи ПК у глобальних мережах та зберігання правильного часу для файлыв й папок під час копіювання інформації ведення системного часу здійснюється на основі всесвітнього часу (UTC від англ. Universal Time Coordinated).

Системний час Windows – поточне значення дати й часу за Гринвічем, у нульовому часовому поясі. Додаток (програма) Windows може отримати значення системного часу викликом підпрограми (функції) *GetSystemTime*, яка заповнює структуру *TSystemTime*. Структура *struct_SYSTETIME* наступна:

```
Typedef struct_SYSTETIME
{
     word wYear;
     word wMonth;
```

```
    word wDayOfWeek;
    word wDay;
    word wHour;
    word wMinute;
    word wSecond;
    word wMillisecond;
 }
```

Локальний (місцевий) час може відрізнятись від системного та залежить від налаштувань Windows. Додаток може отримати поточне значення локального часу викликом функції *GetLocalTime*.

Для роботи з таймером ОС Windows використовує функції *GetSystemTime* та *SetSystemTime*, прототипи яких містяться у бібліотеці winbase.h. *GetSystemTime* здійснює вибір даних зі структури *SystemTime*, в якій містяться дані часу.

Приклад – Программа виведення локального часу ОС Windows

```
#include <stdio.h>
#include <winbase.h>

void main()
{
   SYSTIME st;

//читання системного часу Ос Windows GetlocalTime;

   GetLocalTime(&st);

//виведення часу у форматі год:хв:сек:мілісек

   printf (" Системний час Windows: %02d:%02d:%02d:%02d",
              st.wHour, st.wMinute, st.wSecond, st.wMillisecodns);

}
```

Приклад – Программа виведення локальної дати ОС Windows

```
#include <stdio.h>
#include <winbase.h>

void main ()
{
 SYSTEMTIME st;

 //читання ситемного часу та календаря ОС Windows

 GetLocalTime(&st);

 //виведення дати у форматі "день.місяць.рік"

 printf("Дата: %02d.%02d.%02d", st.wDay, st.wMonth, st.wYear);

}
```

Огляд компілятора C++ Builder 6

C++ Builder – це середовище швидкої розробки, в якому використовується мова C++. На сьогоднішній день цей продукт є найбільш популярним для розробки додатків (програм) для ОС Windows.

Початок роботи. Вигляд екрану після завантаження C++ Builder дещо незвичний. Замість одного вікна на екрані з'являються п'ять:

- головне вікно C++ Builder 6;
- вікно стартової форми – Form1;
- вікно редактора властивостей об'єктів – Object Inspector;
- вікно перегляду списку об'єктів – Object TreeView;
- вікно редактора коду – Unit1.cpp.

У головному вікні знаходиться меню команд, панелі інструментів та палітра компонентів.

В рядку заголовку головного вікна відображається ім'я відкритого у даний момент проекта. Рядок меню вміщує команди, необхідні для розробки та тестування додатків, а також команди керування ними.

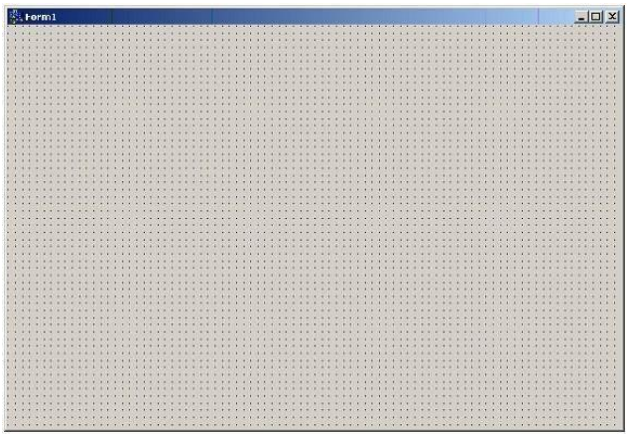
Панель інструментів вміщує кнопки, що відповідають певним командам меню, наприклад, командам Save, Run тощо.

У палітрі компонентів відображаються компоненти, за допомогою яких користувач створює свої додатки.

Компоненти є основними елементами кожного Builder-додатка і, одночасно, основою бібліотеки візуальних компонентів (VCL) C++ Builder. Піктограми стандартних компонентів розділені на групи відповідно функціям, які вони виконують. Кожна з цих груп піктограм розміщена на окремій сторінці палітри компонентів. Після запуску C++Builder активною є сторінка Standard.

Вікно стартової **форми** (**Form1**) представляє собою заготовку головного вікна додатка, що розробляється. І робота над новим проектом починається саме зі створення стартової форми, оскільки кожен додаток має хоча б одну форму, яка служить головним вікном. Стартова форма створюється шляхом зміни значень властивостей форми Form1 та додавання до форми необхідних компонентів (командних кнопок, позначок, полів введення текстової інформації тощо) з палітри компонентів.

Для розміщення компонента в формі необхідно вибрати в палітрі компонентів



кнопку, що відповідає необхідному компоненту. Далі клацнути на формі. Компонент з'явиться на формі, причому його верхній лівий кут буде там, де знаходився курсор миші в момент кліка.

Крім того можна легко зробити відразу кілька копій компонента. Для цього під час вибора компонента необхідно утримувати натиснутою клавішу Shift. В такому режимі при кожному кліці на формі

буде розміщуватися новий компонент. Для зупинки цього процесу необхідно клікнути на кнопці покажчика (кнопка зі стрілкою) в палітрі компонентів.

Ще C++ Builder забезпечує спрощений спосіб розміщення компонентів на формі. Якщо просто двічі клікнути на потрібній кнопці в палітрі компонентів, то компонент з'явиться на формі. При цьому він буде відцентрований горизонтально та вертикально. Такий компонент буде мати розмір за замовчуванням. Вікно **Object TreeView** містить ієрархічний список об'єктів, що є елементами проекту.

Для зміни значень властивостей об'єктів, в тому числі і форми, використовується інспектор об'єктів (**Object Inspector**).

У термінології візуального проектування **об'єкти** – це діалогові вікна та елементи управління (поля введення та виведення, командні кнопки, перемикачі тощо). А властивості об'єкта – це характеристики, які визначають вигляд, положення та поведінку об'єкта.

На вкладці **Properties** в лівій колонці перераховані властивості вибраного об'єкта, а в правій – вказані їхні значення.

На вкладці **Events** перелічені оброблювачі подій вибраного об'єкта.

Подія (Event) – це те, що відбувається під час роботи програми. В C++Builder кожній події надано ім'я. Наприклад, клік кнопкою миші – це подія OnClick, подвійний клік мишею – подія OnDblClick, натиснення клавіші клавіатури – подія OnPressKey.

Реакцією на подію має бути якась дія. Наприклад, натиснення кнопки „Close" має призводити до завершення роботи додатка, а натиснення кнопки „Обчислити" – до обчислення результату за певною формулою.

В C++Builder реакція на подію реалізується як функція обробки події. Т. б., для того, щоб програма виконала певну роботу у відповідь на дії користувача, програміст повинен написати функцію обробки відповідної події.

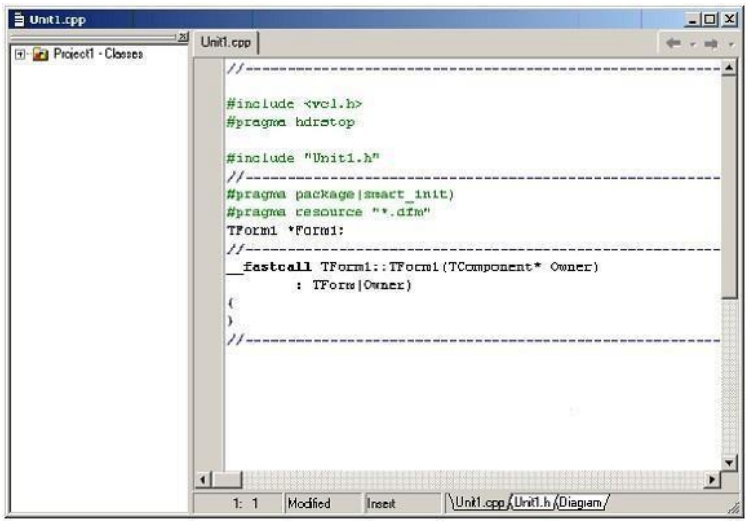
Отже, в лівій колонці укладки **Events** перераховано події, які може сприймати вибраний компонент. В правій колонці з'являється згенероване C++Builder ім'я функції *обробки відповідної події*.

C++Builder надає функції обробки події ім'я, що складається з двох частин. Перша частина ідентифікує форму, що містить об'єкт компонент, для якого створена функція обробки події. Друга частина імені ідентифікує сам об'єкт та подію. *Наприклад*:

```
void _fastcall TForm1::Button1Click(Tobject *Sender) void
fastcall TMainForm::FormCreate(TObject *Sender)
```

Вікно редактора коду (**Unit1.cpp**) розділене на дві частини. У правій частині слід набирати програмний код. Проте, навіть на початку роботи над новим проектом

ця частина вікна редактора коду не порожня. Вона вже містить шаблон програми C++ Builder.



Ліва частина, яка називається *навігатор класів*, полегшує навігацію по тексту програми. В ієрархічному списку, структура якого залежить від проекту, перелічені об'єкти проекту (форми і компоненти) та функції обробки подій. Вибравши елемент списку, можна швидко перейти до потрібного фрагменту коду.

Під час набирання тексту програми редактор коду автоматично виділяє елементи програми: напівжирним шрифтом – ключові слова мови програмування, курсивом – коментарі. Це робить текст програми більш виразнішим, що полегшує сприйняття структури програми.

Запуск програми на виконання виконується за допомогою команди Run (Пуск) головного меню C++ Builder, кнопки Run панелі інструментів або клавішею F9.

ЗАВДАННЯ

4. Розробити блок-схеми алгоритмів і програм для операційної системи DOS, які:
   4.1 Виводить на екран значення таймера операційної системи (4-байтової комірки пам'яті з початковою адресою 046Ch);
   4.2 Встановлює системний час ОС DOS *N*:*N+10*:*N+15*, де *N* – порядковий номер студента у списку;
5. Розробити блок-схеми алгоритмів і додатки (програми) для операційної системи Windows, які:
   5.1 Після натиснення кнопки «Дата» виводить системну дату Windows;
   5.2 Після натиснення кнопки «Системний час» виводить системний час Windows, а після натиснення «Локальний час» – локальний час.

Лабораторна робота №5

ТЕМА: Програмування контролера переривань i8259 в IBM PC / AT.

МЕТА: Набути навичок з програмування контролера переривань, розробляти й встановлювати власні обробники апаратних та програмних переривань.

Короткі теоретичні відомості

З кожним перериванням зв`язана та чи інша подія. Система повинна розпізнати яке переривання з яким номером відбулося і яку відповідну процедуру треба виконати. Відомі 2 види переривань: апаратне і програмне. Програмні переривання зручно використовувати для організації доступу до окремих спільних для всіх програм модулям. Прикладні програми можуть самі встановлювати свої обробники переривань, для їх послідуючого використовування іншими програмами. Для цього обробники переривань повинні бути резидентними у пам`яті.

Апаратні переривання викликаються фізичними пристроями і проходять асинхронно. Ці переривання інформуть систему про подію зв`язану з роботою перефирійних пристроїв. Використовування переривань при роботі з повільними зовнішніми пристроями дозволяє сумістити ввод\вивід з обробкою даних у ЦП і завдяки цьому підвищується працездатність системи.

Деякі переривання зарезервовані для використання самим процесором. Іноді бажано зробити систему, яка не реагує на всі або окремі переривання, для цього використовують маскування переривань.

Складання програм обробника переривань і заміна стандартних обробників MS DOS I BIOS є відповідальною роботою. Необхідно врахувати усі тонкощі роботи апаратури і взаємодію програмного і апаратного забезпечення.

Таблиця векторів переривань

Для того, щоб зв`язати адресу обробника переривань з номером переривання використовують таблицю векторів переривань, яка займає 1 Кб оперативної пам`яті від 0000:0000 до 0000:03FF. Таблиця має 256 елементів FAR обробників. Ці елементи звуться векторами переривань. У першому слові елемента таблиці записано зміщення, а у другому адреса сегмента обробника переривань. void (* interrupt_table [256]);

Апаратні переривання: масковані, немасковані.
Програмні переривання: BIOS, DOS, користувача.
Виключні ситуації: помилки, пастки, аварійне завершення.

IRQ_0 – системний таймер;
IRQ_1 – клавіатура;
IRQ_2 – каскадування КП;
IRQ_3 – COM2;
IRQ_4 – COM1;
IRQ_5 – порт принтера RTP2;
IRQ_6 – гнучкі магнітні диски;
IRQ_7 - порт принтера RTP1;
IRQ_8 – RTC;
IRQ_9 – відеоадаптер;
IRQ_{10} – резерв користувача;
IRQ_{11} – резерв користувача;
IRQ_{12} – резерв користувача;
IRQ_{13} – співпроцесор;
IRQ_{14} – жорсткі магнітні диски HMD;
IRQ_{15} – резерв користувача.

Помилки – це виключні ситуації, які знаходяться і обслуговуються після вибірки до виконання команди, що обслуговується.

Пастки – це виключні ситуації проя кі повідомляється одразу після виконання команди, яка привела до появи даної ситуації.

Аварійне завершення – це виключні ситуації при неможливості точно локалізувати джерело помилки і використовуються при виявленні глобальних помилок.

Заміна таблиці векторів переривань

Якщо потрібно організувати обробку деяких переривань, то програма повинна переназначити вектор переривань на власний обробник. Це можна зробити, змінивши зміст відповідного елемента таблиці вектора переривань.

Послідовність дій для нерезидентних програм обробки переривань:

1. Прочитати зміст елементів таблиці вектора переривань для вектора з потрібним вам номером.
2. Запам`ятати цей зміст із області даних програми.
3. Встановити нову адресу у таблиці векторів переривань так, щоб вона відповідала початку вашої програми обробки переривань.
4. Перед завершенням роботи програми прочитати із області даних адресу старого обробника переривань і записати його у таблицю векторів переривань.

Для читання вектора використовується функція 35h переривання INT 21h.

У лістингу 6.1 наведено код програми, який встановлює власний обробник переривання на вектор 1Ch, який постійно виводить символ “#” у верхньому правому кутку екрану.

Лістинг 6.1 – Програма постійного виведення символа “#” у верхньому правому кутку екрану за допомогою встановлення обробника переривання 1Ch

```
#include <stdio.h>
#include <dos.h>
#pragma check_stack(off)
#pragma check_pointer(off)
#define INTR 0X1C
#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt (*oldhandler)(__CPPARGS);

void interrupt handler(__CPPARGS)  //власний обробник handler, який
                                   //у знакомісце 0 текстової відеопам’яті
                                   //виводить символ "#"
{
      pokeb(0xb800,0,'#');         //веведення символа "#"
      oldhandler();                //виклик попереднього обробника 1Ch
}

int main(void)
{
      printf("Програма встановлення власного обробника 1Ch\n");
      oldhandler = getvect(INTR); //збереження попереднього вектора 1Ch
      setvect(INTR, handler);     //встановлення нового обробника handler
      printf("\tОБРОБНИК 1Ch ВСТАНОВЛЕНО!\nДля виходу натисніть будь-яку
             клавішу.\n");
      getch();
      return 0;
}
```

Примітка: для перевірки роботи програми необхідно запустити її за допомогою оболонки VC або FAR (рисунок 6.1).

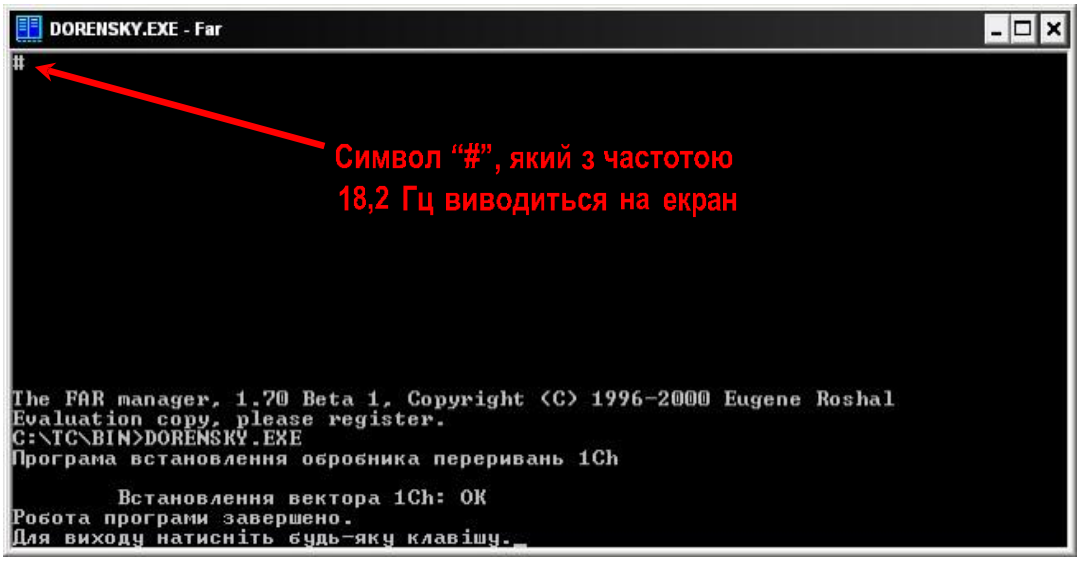


Рисунок 6.1 - Перевірка роботи програми

Особливості обробки апаратних переривань

Поступаючі переривання запам`ятовуються у регістрі запиту на переривання IRR. Кожний біт із восьми у цьому регістрі відповідає перериванню. На обробку переривання інформація береться із регістра обслуговування переривань ISR. Перед видачею запиту на переривання у процесор, перевіряється зміст регістра маски переривань IMR. Якщо переривання даного рівня не замасковане, то видається запит на переривання.

Найбільш цікавими з точки зору програмування контролера переривань являються регістри маски переривань.

ЗАВДАННЯ

6. Розробіть структурну схему організації системного часу в IBM PC / AT (BIOS → Інтервальний таймер i8254 /канал 0/ → КП /IRQ0: INT 8h/, BIOS → RTC → Таймер ОС).
7. Розробити та ініціалізувати власний обробник переривання 1Ch, який за допомогою таймера операційної системи (комірки пам’яті за адресою 046Ch) о 12:30 год. переводить системний час ПК згідно варіанта:
   Варіант 0) на 8 годин 59 хвилин 40 секунд назад;
   Варіант 1) на 2 години 30 хвилин вперед;
   Варіант 2) на 1 годину 10 хвилин назад;
   Варіант 3) на 5 годин 8 хвилин 9 секунд назад.
   Варіант 4) на 5 годин 35 секунд вперед;
   Варіант 5) на 1 годину 25 хвилин 5 секунд назад;
   Варіант 6) на 3 години 2 хвилини вперед;
   Варіант 7) на 1 годину 12 хвилин вперед;

Варіант 8) на 7 хвилин 10 секунд вперед;
Варіант 9) на 9 годин 50 хвилин 1 секунду назад;

(!) Примітка: ([номер_варіанта] = [номер_у_списку] % 10).

8. Розробити блок-схему алгоритма та програму, яка за допомогою перепрограмування контролера переривань замасковує обробку переривання від floppy-дисковода.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

11. На базі якої мікросхеми реалізовано КП у IBM PC?
12. Призначення та місце КП в обчислювальній системі. Система переривань.
13. Алгоритм функціонування КП.
14. Що таке вектор переривання? Суть і призначення таблиці векторів переривань. Де вона знаходиться?
15. Роль і місце КП в організації системного часу в ПК IBM PC.
16. Алгоритм програмування КП.
17. Що відбувається з запитами на переривання, якщо відповідний IRQ замасковано?
18. Особливості обробки апаратних переривань.
19. Рівні пріоритетів обробки переривань. Як здійснюється їх зміна?
20. Яке місце і роль КП у принципі відкритості системи? Чи можливо організувати принцип відкритості без КП?

Лабораторна робота №6

ТЕМА: **Контролер прямого доступу до пам’яті.**
МЕТА: Опанування основ роботи і програмуванням контролера прямого доступу до пам’яті в комп’ютері IBM PC.

Теоретичні відомості

Контролер прямого доступу до пам'яті (ПДП, DMA - Direct Memory Access) забезпечує високошвидкісний обмін даними між пристроями введення-виведення й ОЗП без використання центрального процесора, що дозволяє звільнити процесор для виконання обчислень паралельно з обміном і незалежно від нього. Найбільше часто можливості ПДП використовуються при роботі з дисковими накопичувачами, однак реалізоване використання ПДП адаптерами накопичувачів на магнітній стрічці і поруч інших пристроїв. Відчутні переваги дає використання ПДП у процесі обміну з пристроями, що приймають чи передають дані досить великими порціями з високою швидкістю.

У IBM PC-подібних комп'ютерах функції контролера ПДП виконує мікросхема 8237A фірми INTEL (радянський аналог КР580ВТ57) чи її аналоги 8237A-4 і 8237A-5, що працюють з тактовою частотою 4 і 5 Мгц відповідно (стандартна мікросхема 8237A працює на частоті 3 Мгц). Контролер має 4 незалежних канали, кожний з який може обслуговувати один периферійний пристрій.

Для здійснення ПДП контролер повинен виконати наступні операції:

1. прийняти запит (DRQ) від пристрою введення/виведення;
2. видати HRQ у процесор на захоплення шини (HOLD);
3. прийняти сигнал HLDA, який дозволяє захоплення шини;
4. сформувати сигнал (DACK), який повідомляє пристрій про початок обміну даними.

Призначення каналів ПДП для IBM AT.

Номер каналу	Призначення
0, 5, 6, 7	Зарезервовано
1	Керування синхронною передачею даних (SLDC)
2	Обмін з контролером гнучких дисків
3	Обмін з контролером жорстких дисків
4	Для каскадного з’єднання з 1-м контролером ПДП

Принцип роботи контролера ПДП

У роботі ПДП розрізняються 2 головних цикли: цикл чекання (Idle cycle) і активний цикл (Active cycle). Кожен цикл підрозділяється на ряд станів, що

займають за часом один період часів (тик). З циклу чекання контролер може бути переведений у стан програмування (Program Condition) шляхом подачі на вхід RESET сигналу високого рівня, тривалістю не менше 300 нс і наступної за ним подачі сигналу низького рівня (рівня 0) на вивід CS (Chip Select). У стані програмування контролер буде знаходиться доти, поки на виводі CS збережеться сигнал низького рівня. У процесі програмування контролеру задаються:

- початкова адреса пам'яті для обміну;
- зменшене на одиницю число переданих байтів;
- напрямок обміну, а також установлюються необхідні режими роботи (дозволити чи заборонити циклічну зміну пріоритетів, авто-ініціалізацію, задати напрямок зміни адреси при обміні і т.д.).

Завантаження 16-розрядних регістрів контролера здійснюється через 8-розрядні порти введення-виведення. Перед завантаженням першого (молодшого) байта повинен бути скинутий (очищений) тригер-засувка (тригер перший/останній, First/Last flip-flop), що змінює свій стан після виведення в порт першого байта й у такий спосіб дає можливість наступною командою виведення в той же порт завантажити старший байт відповідного регістра.

Запрограмований канал повинен бути демаскований (біт маски каналу встановлюється при цьому в 0), після чого він може приймати сигнали "Запит на ПДП", що генеруються тим зовнішнім пристроєм, що обслуговується через цей канал. Сигнал "Запит на ПДП" може бути також ініційований установкою в 1 біта запиту даного каналу в регістрі запитів контролера. Після появи сигналу запиту контролер входить в активний цикл, у якому виконується обмін даними. Обмін може здійснюється в одному з чотирьох режимів:

1. Режим одиночної передачі (Signle Transfer Mode). Після кожного циклу передачі контролер звільняє шину процесору, але відразу ж починає перевірку сигналів запиту і, як тільки виявляє активний сигнал запиту, ініціює наступний цикл передачі.

2. Режим блокової передачі (Block Transfer Mode). У цьому режимі наявність сигналу запиту потрібно тільки до моменту видачі контролером сигналу "Підтвердження запиту на ПДП" (DACK), після чого шина не звільняється аж до завершення передачі всього блоку.

3. Режим передачі за вимогою (Demand Transfer Mode). Даний режим є проміжним між двома першими: передача йде безупинно доти, поки активний сигнал запиту, стан якого перевіряється після кожного циклу передачі. Як тільки пристрій не може продовжити передачу, сигнал запиту скидається ним і контролер припиняє роботу. Цей режим застосовується для обміну з повільними пристроями, що не дозволяють по своїх тимчасових характеристиках працювати з ПДП у режимі блокової передачі.

4. Каскадний режим (Cascade Mode). Режим дозволяє включити в підсистему ПДП більш одного контролера в тих випадках, коли недостатньо чотирьох каналів ПДП. У цьому режимі один з каналів ведучого контролера використовується для каскадування з контролером другого рівня. Для роботи в каскаді сигнал HRQ

("Запит на захоплення") відомого контролера подається на вхід DREQ ("Запит на ПДП") ведучого, а сигнал DACK ("Підтвердження запиту") ведучого подається на вхід HDLA ("Підтвердження захоплення") відомого. Така схема підключення аналогічна підключенню ведучого (першого) контролера до мікропроцесора, з яким він обмінюється сигналами HRQ і HDLA.

Типи передач

1. Передача пам'ять-пам'ять (Memory-to-memory DMA). Використовується для передачі блоку даних з одного місця пам'яті в інше. Вихідна адреса визначається в регістрах нульового каналу, вихідний - у регістрах першого каналу. Число циклів обміну (число байт мінус 1) задається в регістрі числа циклів каналу 1. Передача відбувається з використанням робочого регістра контролера як проміжної ланки для збереження інформації. При передачі пам'ять-пам'ять може бути заданий спеціальний режим фіксації адреси (Address hold), при якому значення поточного адреси в регістрі нульового каналу не змінюється, при цьому весь вихідний блок пам'яті заповнюється тим самим елементом даних, що знаходиться по заданій адресі.

2. Авто-ініціалізація (автозавантаження, Autoinitialization). Після завершення звичайної передачі використаний канал ПДП маскується і повинен бути перепрограмований для подальшої роботи з ним. При авто-ініціалізації маскування каналу після закінчення передачі не відбувається, а регістри поточного адреси і лічильник циклів автоматично завантажуються з відповідних регістрів з початковими значеннями. У такий спосіб для продовження (повторення) обміну досить виставити сигнал запиту на ПДП по даному каналі.

3. Режим фіксованих пріоритетів. У цьому режимі канал 0 завжди має максимальний пріоритет, а канал 3 - мінімальний. Це означає, що будь-яка передача по каналу з більш високим пріоритетом буде виконуватися раніше, ніж по каналу з більш низьким пріоритетом.

4. Циклічне обертання пріоритетів. Дозволяє уникнути "забивання" шини одним каналом при одночасній передачі по декількох каналах. Кожному каналу, по якому пройшла передача, автоматично привласнюється нижчий пріоритет, після чого право на передачу одержує канал з найвищим пріоритетом, для якого передача в даний момент можлива. Таким чином, якщо на початку роботи розподіл пріоритетів був звичайним (канал 0 - найвищий), і прийшли сигнали запиту на ПДП по 1-му і 2-му каналах, то спочатку буде виконуватися передача по першому каналу, потім він одержить нижчий пріоритет (а канал 2, відповідно, вищий, тому що обертання пріоритетів циклічне) і передача виконається по 2-му каналу, що потім одержить нижчий пріоритет, а вищий пріоритет одержить, відповідно, канал 3, що і буде мати переважне право на передачу.

5. Стиск часу передачі (Compressed transfer timing). У випадку, якщо тимчасові характеристики швидкодії пристроїв, що обмінюються, збігаються, ПДП може скоротити час виконання кожного такту передачі на 2 цикли часів за рахунок тактів чекання, що входять у кожен цикл передачі.

Опис внутрішніх регістрів ПДП

Контролер має 344 біта внутрішньої пам'яті, організованої у вигляді регістрів. Опис внутрішніх регістрів ПДП приведено в таблиці.

Назва регістру	Розрядність (біт)	Кількість регістрів
Регістр початкової адреси (BAR)	16	4
Регістр початкового лічильника циклів (BWCR)	16	4
Регістр поточної адреси (CAR)	16	4
Регістр поточного лічильника циклів (CWCR)	16	4
Робочий регістр адреси (TAR)	16	1
Робочий регістр лічильника циклів (TWCR)	16	1
Регістр стану (SR)	8	1
Регістр команд (CR)	8	1
Регістр режиму (MR)	8	4
Робочий регістр (TR)	8	1
Регістр масок (Mask Register)	4	1
Регістр запитів (RR)	4	1

Регістр початкової адреси (Base Address Register). У цьому регістрі задається стартова адреса ОЗУ, з якого починається передача. Регістр містить 16 розрядів і визначає адреса усередині заданої сторінки пам'яті розміром 64 кб. Завдання номера сторінки пам'яті здійснюється через спеціальні сторінкові регістри (Page Registers), підтримувані зовнішньою логікою. Кожен канал ПДП має свій регістр початкової адреси і сторінковий регістр. Такий розподіл пам'яті на сторінки не дозволяє здійснити обмін із блоком пам'яті, що знаходиться на перетинанні двох сторінок. Кожна сторінка починається із сегментної адреси, кратного 1000h (0, 1000h, 2000h, ..., 9000h).

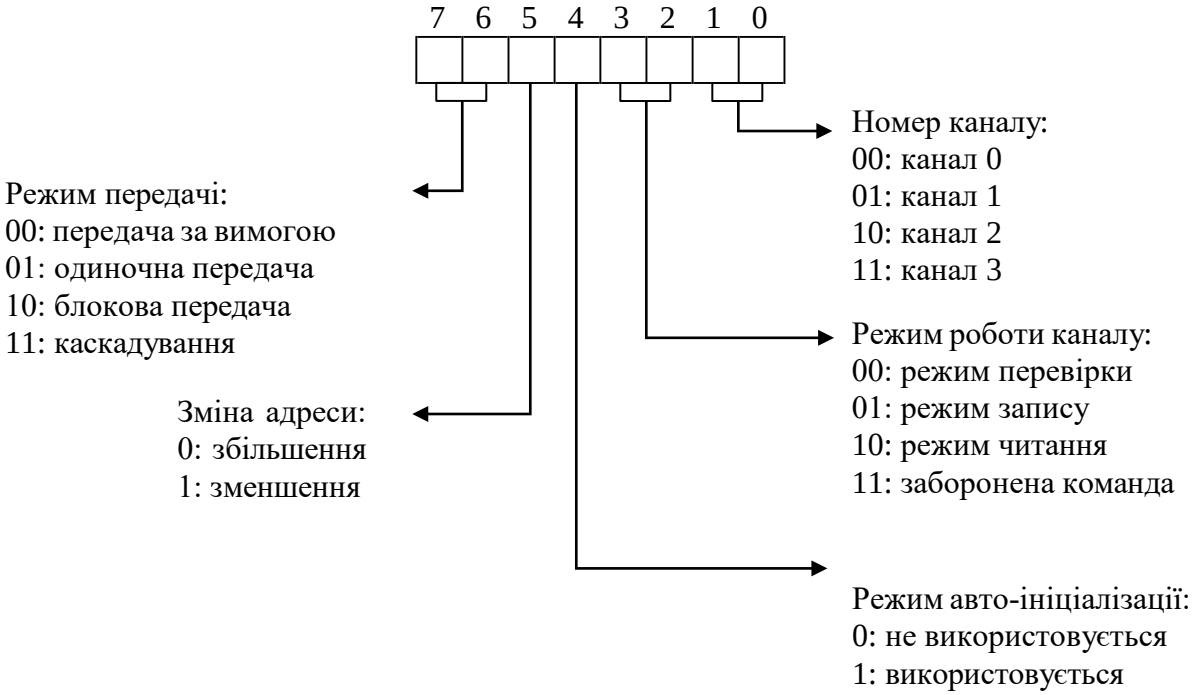
Регістр початкового лічильника циклів (Base Word Count Register). У цьому регістрі задається початкове число циклів передачі для програмувального каналу. Фактичне число переданих під час роботи ПДП елементів даних на одиницю перевищує задане число циклів.

Регістр поточного адреси (Current Address Register). Початкове значення заноситься в цей регістр одночасно з регістром початкової адреси. Надалі в ході передачі значення поточного адреси автоматично чи збільшується зменшується (конкретний напрямок зміни задається при програмуванні в регістрі режиму). Якщо дозволена авто-ініціалізація, то після закінчення передачі в регістр автоматично встановлюється значення з регістра початкової адреси.

Регістр поточного лічильника циклів (Current Word Count Register). Регістр містить поточне значення лічильника циклів (число циклів передачі, що залишилися). Відображуване в ньому число циклів завжди на одиницю менше числа ще не переданих елементів даних, тому що зміна значення в цьому регістрі проводиться наприкінці циклу передачі, уже після фактичної передачі елемента

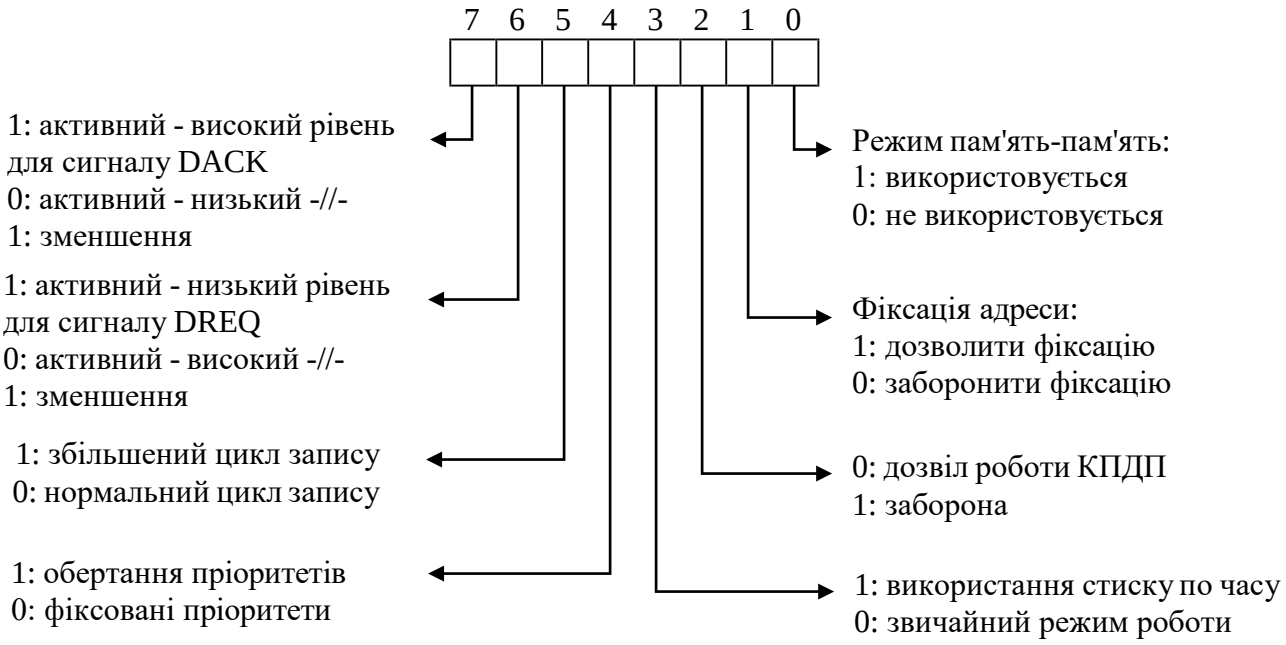
даних, а кінець передачі фіксується в момент переповнення лічильника (зміна його значення з 0 на 0FFFFh).

Регістр режиму (Mode Register). Даний регістр задає режими роботи свого каналу контролера.



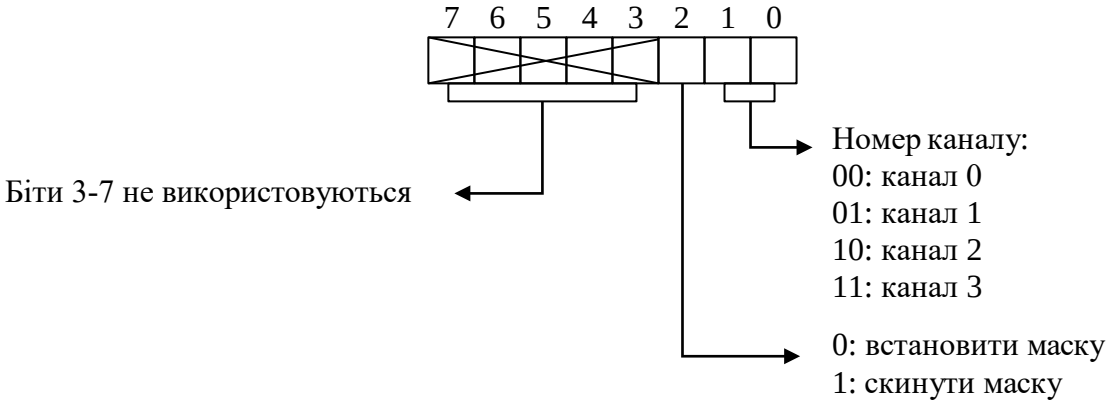
Кожний з чотирьох каналів ПДП має свій набір регістрів, описаних вище. Крім того, мається наступний набір регістрів, загальних для всіх каналів.

Регістр команд (Command Register). Цей 8-бітний регістр керує роботою контролера. Він програмується, коли контролер знаходиться в стані програмування й очищається командами скидання "Reset" і "Master Clear". Призначення бітів регістра команд приведено нижче

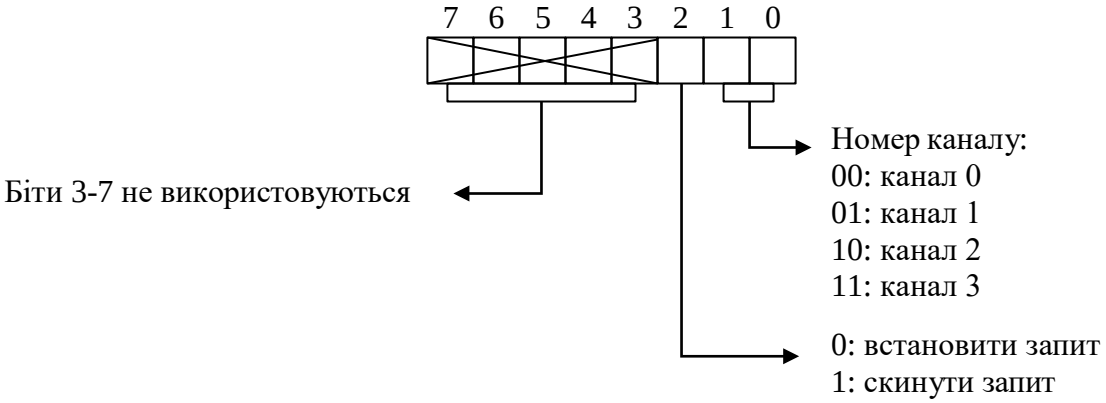


Регістр стану (Status Register). Регістр відображає поточний стан запитів і передач по всім чотирьох каналах. Біти 0 - 3 встановлюються в одиницю після завершення передачі по каналах 0 - 3 (біт 0 - канал 0, біт 1 - канал 1 і т.д.), якщо не заданий режим авто-ініціалізації. Ці біти очищаються після команди скидання контролера і після кожної операції зчитування стану з регістра стану. Біти 4 - 7 указують по якому з каналів 0 - 3 активний в даний момент сигнал запиту на ПДП.

Регістр масок (Mask Register). Кожен біт цього 4-бітового регістра маскує/демаскує свій канал ПДП, при цьому значення 1 маскує канал, значення 0 демаскує канал і дозволяє прийом сигналу запиту по цьому каналу.



Регістр запитів (Request Register). Сигнал запиту на ПДП (DREQ) може бути виданий як пристроєм, що обслуговується, так і програмно. Для програмного видання сигналу запиту по одному з 4-х каналів ПДП необхідно установити відповідний біт у 4-розрядному регістрі запитів. Запит на ПДП може бути відмінений записом нульового значення у відповідний біт регістра. Біт запиту очищається автоматично при закінченні передачі по даному каналу. Усі біти запитів очищаються при скиданні контролера. Для того, щоб сприймати програмні запити на ПДП, канал повинен знаходиться в режимі блокової передачі.



Робочий регістр (Temporary Register). Цей 8-розрядний регістр використовується для збереження елемента даних, переданого в режимі фіксованої адреси при передачі пам'ять або для тимчасового збереження переданого байта при всіх інших режимах передачі.

Програмне керування контролером ПДП

Програмне керування контролером ПДП здійснюється через порти введення-виведення. Доступ до кожного регістра контролера може бути здійснений через свої порти введення-виведення.

У таблиці приведений опис портів введення-виведення, призначених для керування контролером ПДП.

Порт	Режим	Призначення
0, 2, 4, 6	запис	Запис початкової адреси в регістр початкової адреси відповідно каналу 0, 1, 2, 3
	читання	Читання поточної адреси в регістр початкової адреси відповідно каналу 0, 1, 2, 3
1, 3, 5, 7	запис	Запис в регістр початкового лічильника циклів і в регістр поточного лічильника циклів відповідно каналу 0, 1, 2, 3
	читання	Читання поточного значення з регістру поточного лічильника циклів відповідно каналу 0, 1, 2, 3
8	запис	Запис регістра команд КПДП
	читання	Читання регістру стану КПДП
9	запис	Запис в регістр запитів КПДП
0Ah	запис	Запис біта маски для каналів 0-3 КПДП
0Bh	запис	Запис регістра режимів для каналів 0-3 КПДП
0Ch	запис	Очищення (скидання) тригера-засувки
0Dh	запис	Програмне скидання контролера
0Eh	запис	Очищення бітів масок каналів 0-3
0Fh	запис	Запис регістра масок для каналів 0-3
	читання	Читання робочого регістра КПДП

Порти 0h - 7h призначені для запису вихідних значень у регістри початкової і поточної адреси, початкового і поточного лічильника циклів для каналів 0-3. В силу того, що порти 8-розрядні, а регістри, в які через них заносяться дані – 16-розрядні, то запис виконується в два прийоми. Перед першою командою виведення в необхідний порт необхідно скинути тригер-засувку, для чого виконується команда виведення довільного значення в **порт 0Ch**, після чого в необхідний порт виводиться молодший байт 16-розрядного значення і потім старший байт наступною командою виведення в той же порт.

Приклад завантаження початкової адреси каналу 1:

Мова Asm86:

```
mov al, 0
out 0Ch, al  ; Скидання тригера-засувки перед записуванням потрібних
             ; значень в регістр початкової адреси каналу 1

mov al, LowByte
out 2h, al; завантаження молодшого байту початкової адреси каналу 1
mov al, HighByte
out 2h, al; завантаження старшого байту початкової адреси каналу 1
```

….

Мова C:

….

```
outportb(0x0C, 0xB6);  // Скидання тригера-засувки перед записуванням
                       // потрібних  значень в регістр початкової адреси
                       // каналу 1
outportb(0x2, char(Address));   // завантаження молодшого байту
                                // початкової адреси каналу 1
outportb(0x2, char(Address>>8));  // завантаження старшого байту
                                  // початкової адреси каналу 1
```

Виведення в **порт 8h** дозволяє занести значення в регістр команд ПДП.

Читання з порту 8h зчитує регістр стану ПДП. Опис бітів регістра стану та регістра команд було приведено вище.

Запис у **порт 9h** дозволяє встановити або скинути біт запиту в регістр запитів для одного з каналів.

Запис у **порт 0Ah** дозволяє встановити або скинути біт маски в регістрі масок для одного з каналів.

Запис у **порт 0Bh** встановлює значення в регістрі режимів одного з 4-х каналів ПДП. Біти 0 і 1 задають номер каналу (00 - 0, 01 - 1, 10 - 2, 11 - 3). У біти 2 - 7 заносяться значення, передані відповідно в біти 0 - 6 регістра режимів.

Запис у **порт 0Dh** задає програмне скидання контролера (Master Clear). Виведення будь-якого байта в цей порт має той самий ефект, що й апаратне скидання контролера. При програмному скиданні очищаються регістри команд, стану, запитів і робочий регістр. Так само скидається тригер-засувка і

встановлюються всі біти масок у регістрі масок.

Після програмного скидання контролер переходить у цикл чекання.

Виведення будь-якого байта в **порт 0Eh** очищає регістр масок - скидає біти масок усіх 4-х каналів ПДП і в такий спосіб дозволяє прийом запитів на ПДП по всіх каналах.

Через **порт 0Fh** можна задати довільне значення регістра масок ПДП. Для цього необхідно в бітах 0 - 3 регістри AL установити необхідне значення масок каналів 0 - 3 відповідно і вивести це значення в порт.

Доступ до **регістрів сторінок** ПДП здійснюється через наступні порти:

Номер каналу	**Номер порту**
0	87h
1	83h
2	81h
3	82h
5	8Bh
6	89h
7	8Ah

IBM PC AT и AT-подібні ПЕОМ мають два контролера ПДП 8237A, які працюють в каскадному режимі.

Керування каналами 4 - 7 здійснюється аналогічно PC/XT. Канали 4 - 7 призначені для обміну 16-розрядними словами. У зв'язку з цим виникає ряд відмінностей у роботі з цими каналами:

- біт 0 у даних, що заносяться в регістри початкового і поточного адреси, завжди мається на увазі рівним 0, тому через ці регістри передаються біти 1 - 16 повної 23-розрядної адреси (а не біти 0 - 15 повної 20-розрядної адреси, як це реалізовано на XT – подібних ПЕОМ), по цій же причині в сторінкові регістри каналів 4 - 7 заносяться біти 17 - 23 повної адреси, а не біти 16 - 20, як це треба зробити при роботі з каналами 0 - 3;

- оскільки передача здійснюється 16-розрядними словами, у регістри поточного і початкового лічильника циклів заноситься не число байт, а число слів, зменшене на одиницю;

- розміри сторінок пам'яті, у межах яких можливий обмін протягом однієї передачі складають 2000h байтів.

У таблиці приведений опис портів введення-виведення, призначених для керування другим контролером ПДП.

Порт	Режим	Призначення
0C0h, 0C4h, 0C8h, 0CCh	запис	Запис початкової адреси в регістр початкової адреси відповідно каналу 4, 5, 6, 7
	читання	Читання поточної адреси в регістр початкової адреси відповідно каналу 4, 5, 6, 7
0C2h, 0C6h, 0CAh, 0CEh	запис	Запис в регістр початкового лічильника циклів і в регістр поточного лічильника циклів відповідно каналу 4, 5, 6, 7
	читання	Читання поточного значення з регістру поточного лічильника циклів відповідно каналу 4, 5, 6, 7
0D0h	запис	Запис регістра команд КПДП
	читання	Читання регістру стану КПДП
0D2h	запис	Запис в регістр запитів КПДП
0D4h	запис	Запис біта маски для каналів 4-7 КПДП
0D6h	запис	Запис регістра режимів для каналів 4-7 КПДП
0D8h	запис	Очищення (скидання) тригера-засувки КПДП
0DAh	запис	Програмне скидання контролера
0DCh	запис	Очищення бітів масок каналів 4-7
0DEh	запис	Запис регістра масок для каналів 4-7
0DAh	читання	Читання робочого регістра КПДП

Формати даних, що вводяться/виводяться через ці порти збігаються з форматами даних аналогічних портів першого контролера. При роботі з цими портами варто тільки враховувати, що нумерація каналів починається не з 0, як у першого контролера, а з 4 (канал 4 - це канал 0 другого контролера, канал 5 - це його канал 1 і т.д.).

Завдання

Розробити блок-схему алгоритма та програму, яка ініціалізує DMA на запис блоку даних на дискету.

Контрольні питання

1. Призначення контролера прямого доступу до пам’яті;
2. Призначення каналів КПДП для IBM AT;
3. Які порти використовуються для роботи з КПДП;
4. Принцип роботи КПДП;

5. Типи передач, що здійснює КПДП;
6. Призначення регістрів КПДП;
7. Каскадування контролерів;
8. Відмінність КПДП для IBM AT і IBM XT.

Лабораторна робота № 7

ТЕМА: Дескриптори і таблиці глобальних дескрипторів.

МЕТА: Навчитись програмно описувати дескриптори у таблиці глобальних дескрипторів та завантажувати в регістр процесора GDTR, інформацію про таблицю глобальних дескрипторів**.**

Короткі теоретичні відомості

При вмиканні процесора в ньому автоматично встановлюється режим реальної адреси. Перехід у захищений режим здійснюється програмно шляхом виконання відповідної послідовності команд. Оскільки багато деталей функціонування процесора в реальному і захищеному режимах істотно відрізняються, програми, призначені для захищеного режиму, повинні бути написані особливим чином. Реальний і захищений режими є несумісними.

Розглянемо програму, яка переключає процесор у захищений режим.

```
.386P           ;Дозвіл трансляції привілейованих команд
;Структура для опису дескрипторів сегментів
descr    struc
limit     dw    0         ;Межа (біти 0..15)
base_l   dw    0         ;База, біти 0..15
base_m   db    0         ;База, біти 16..23
attr_1      db    0         ;Байт атрибутів 1
attr_2   db    0         ;Межа (біти 16..19) і атрибути 2
base_h   db    0         ;База, біти 24..31
descr    ends

;Сегмент даних
data          segment   use16      ;16-розрядна програма
```

;**Таблиця глобальних дескрипторів GDT**

```
gdt_null    descr <0,0,0,0,0,0>  ;Селектор 0 - обов'язковий
                                 ;нульовий дескриптор
gdt_data descr <data_size-1,0,0,92h,0,0> ;Селектор 8,сегмент даних
gdt_code descr <code_size-1,0,0,98h,0,0> ;Селектор 16,сегмент команд
gdt_stack descr <255,0,0,92h,0,0>  ;Селектор 24, сегмент стеку
gdt_size=$-gdt_null   ;Розмір GDT
pdescr   dq 0         ;Псевдодескриптор для lgdt
real_sp  dw 0         ;Змінна для збереження SP
mes      db 27, '[31;42m Повернулися в реальний режим!  ',27,'[0m$'
data_size=$-gdt_null  ;Розмір сегменту даних
data          ends    ;Кінець сегменту даних

;Сегмент команд
text     segment   'code' use16     ;16-розрядна програма
         assume CS:text, DS:data
main     proc
         mov   AX, data             ;Ініціалізація DS
```

```
            mov   DS, AX                ;у реальному режимі
;Знайдемо 32-бітову лінійну адресу сегменту даних
         mov   DL,0
         shld  DX,AX,4
         shl   AX,4
         mov   BX,offset gdt_data
         mov   [BX]. base_l,AX
         mov   [BX]. base_m,DL
;Знайдемо 32-бітову лінійну адресу сегменту команд
         mov   AX, CS
         mov   DL,0
         shld  DX,AX,4
         shl   AX,4
         mov   BX,offset gdt_code
         mov   [BX]. base_l,AX
         mov   [BX]. base_m,DL
;Знайдемо 32-бітову лінійну адресу сегменту стеку
         mov   AX,SS
         mov   DL,0
         shld  DX,AX,4
         shl   AX,4
         mov   BX,offset gdt_stack
         mov   [BX]. base_l,AX
         mov   [BX]. base_m,DL
```

;Підготуємо дескриптор і завантажимо регістр GDTR

```
         mov   BX,offset gdt_data    ;Адреса GDT
         mov   AX,[BX].base_l;Одержимо та занесемо у pdescr
         mov   word ptr pdescr+2, AX ;базу, біти 0..15
         mov   DL,[BX]. base_m;Одержимо і занесемо у pdescr
         mov   byte ptr pdescr+4, DL ;базу, біти 16..23
         mov   word ptr pdescr, gdt_size-1 ;Межа GDT
         lgdt  pdescr              ;Завантажимо регістр GDTR
 ;Підготуємося до повернення в реальний режим
         mov   AX, 40h         ;Налаштуємо ES на область
         mov   ES,AX           ;даних BIOS
  mov word ptr ES:[67h],offset return;Зсув точки повернення
         mov   ES:[69h],CS     ;Сегмент точки повернення
;Підготуємося до переходу в захищений режим
         cli                ;Заборона апаратних переривань
         mov   AL,0Fh       ;Вибірка байту стану відключення
         out   70h,AL       ;Порт КМОП-мікросхеми
         mov   AL, 0Ah      ;Установка режиму відновлення
         out   71h,AL       ;після скидання процесора
;Перехід в захищений режим
         smsw  AX           ;Одержимо слово стану комп'ютера
         or    AX,1         ;Встановимо біт PE
         lmsw  AX           ;Запишемо слово стану
```

;Процесор працює в захищеному режимі

```
         mov   CX,1000       ; Кількість ітерацій циклу
m1:      mov  BX,AX
         loop  m1       ;Цикл
;Повернемося в реальний режим
         mov   real_sp,SP  ;Збережемо SP
         mov   AL,0FEh     ;Команда скидання процесора
         out   64h,AL      ;у порт 64h
```

```
            hlt          ;Зупиняемо процесор до кінця скидання
;Процесор працює в реальному режимі
;Відновимо операційне середовище реального режиму
return:  mov    AX,data     ;Відновимо адресуємість даних
         mov    DS,AX
         mov    SP,real_sp  ;Відновимо адресуємість стеку
         mov    AX,stk
         mov    SS,AX
         sti                ;Дозволимо апаратні переривання
;Виконуємо перевірку функцій DOS після повернення в ;реальний режим
         mov    AH, 09h     ;Функція виведення на екран рядка
         mov    DX, offset mes ;Адреса рядка
         int    21h
         mov    AX, 4C00h   ;Завершимо програму звичайним чином
         int    21h
main     endp               ;Кінець головної процедури
code_size=$-main            ;Розмір сегменту команд
text     ends               ;Кінець сегменту команд

stk      segment   stack 'stack' ;Початок сегменту стеку
         db   256 dup ('^')
stk      ends                    ;Кінець сегменту стеку
         end main                ;Кінець програми
```

Програма починається з опису структури дескриптора сегменту. На відміну від реального режиму, у якому сегменти визначаються їхніми базовими адресами, що задаються програмістом у явній формі, у захищеному режимі для кожного сегменту програми повинен бути визначений дескриптор – 8-байтове поле, в якому у визначеному форматі записуються базова адреса сегменту, його довжина і деякі інші характеристики (рисунок 1).

Рисунок 1– Дескриптор сегменту

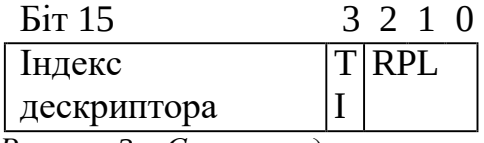


Рисунок 2 – Селектор дескриптора

Тепер для звертання до необхідного сегменту програміст заносить у сегментний регістр не сегментну адресу, а так званий селектор (рисунок 2), до складу

якого входить номер (індекс) відповідного сегменту дескриптора. Процесор

за цим номером знаходить потрібний дескриптор, витягає з нього базову адресу сегменту, збільшує її на зазначене в конкретній команді зміщення (відносну адресу), формує адресу чарунки пам'яті. Індекс дескриптора (0, 1, 2 і т.д.) записується в селектор починаючи з біту 3, що еквівалентно множенню його на 8. Таким чином, можна вважати, що селектори послідовних дескрипторів являють собою числа 0, 8, 16, 24 і т.д. Інші поля селектора, що для нашого випадку приймають значення 0, будуть описані пізніше.

Структура descr є шаблоном для дескрипторів сегментів, що полегшує їхнє формування. Розглянемо вміст дескриптора. Межа (limit) сегменту являє собою номер останнього байту сегменту. Так, для сегменту розміром 375 байтів межа дорівнює 374. Поле межі складається з 20 бітів і розбите на дві частини. З рисунку 1 видно, що молодші 16 бітів межі займають байти 0 і 1 дескриптора, а старші 4 біти входять до байту атрибутів_2, займаючи в ньому біти 0...3. Виходить, що розмір сегменту обмежений розміром 1 Мбайт. Насправді це не так. Межа може вказуватися або в байтах (і тоді, дійсно, максимальний розмір сегменту дорівнює 1 Мбайт), або в блоках по 4 Кбайти (і тоді розмір сегменту може досягати 4 Гбайт). Одиниці, в яких задається межа, визначаються старшим бітом байту атрибутів_2, який називається бітом дробовості. Якщо він дорівнює 0, межа вказується в байтах; якщо 1 - у блоках по 4 Кбайти.

База сегменту (32 біти) визначає початкову лінійну адресу сегменту в адресному просторі процесора.

Поле бази, як і поле межі, розбито на 2 частини: біти 0...23 займають байти 2, 3 і 4 дескриптора, а біти 24...31 - байт 7. Для зручності програмного звертання в структурі descr база описується трьома полями: молодшим словом (base_l) і двома байтами: середнім (base_m) і старшим (base_h).

У байті атрибутів_1 задається низка характеристик сегменту. У прикладі використовуються сегменти двох типів: сегмент команд, для якого байт attr_l повинний мати значення 98h, і сегмент даних (або стеку) із кодом 92h.

Сегмент даних data оголошений із типом використання use16 (так само буде оголошений і сегмент команд). Він повідомляє, що в даному сегменті будуть використовуватися 16-бітові адреси. Якби ми готували нашу програму для роботи під керуванням операційної системи захищеного режиму, що реалізує всі можливості мікропроцесора, тип використання був би use32. Проте наша програма буде працювати під керуванням DOS, яка працює в реальному режимі з 16-бітовими адресами й операндами.

Сегмент даних починається з опису найважливішої системної структури - **таблиці глобальних дескрипторів**. Як вже відзначалося вище, звертання до сегментів у захищеному режимі можливо тільки через дескриптори цих сегментів. Таким чином, у таблиці дескрипторів повинно бути описано стільки дескрипторів, скільки сегментів використовує програма. У нашому випадку до таблиці включені, крім обов'язкового нульового дескриптора, що завжди займає перше місце в таблиці, чотири дескриптори для сегментів даних, команд, стеку і додаткового сегменту даних, який ми накладемо на відеобуфер, щоб забезпечити можливість виведення в нього символів. Порядок дескрипторів у таблиці (крім нульового) не має значення.

Крім єдиної **таблиці глобальних дескрипторів**, що позначається GDT (Global Descriptor Table), у пам'яті може знаходитися багато таблиць локальних

дескрипторів (LDT - Local Descriptor Table). Різниця між ними полягає в тому, що сегменти, які описуються глобальними дескрипторами, доступні всім задачам, виконуваним процесором, а до сегментів, що описуються локальними дескрипторами, може звертатися тільки та задача, у якій ці дескриптори описані. Оскільки ми маємо справу з однозадачним режимом, локальна таблиця нам не потрібна.

У дескрипторі gdt_data, що описує сегмент даних програми, заповнюється поле межі сегменту (фактичне значення розміру сегменту data_size буде обчислено транслятором), а також байт атрибутів_1. Код 92h вказує на те, що це сегмент даних із дозволом запису і читання. Базу сегменту, тобто фізичну адресу його початку, прийдеться обчислити програмно і занести в дескриптор вже на етапі виконання.

Дескриптор gdt_code сегменту команд заповнюється таким же чином. Код атрибута 98h вказує, що це виконуємий сегмент, до якого заборонене звертання з метою читання або запису. Таким чином, сегменти команд у захищеному режимі не можна модифікувати по ходу виконання програми.

Дескриптор gdt_stack сегменту стеку має, як і будь-який сегмент даних, код атрибуту 92h, що дозволяє його читання і запис, і явно задану межу 255 байт, що відповідає розміру стеку. Базова адреса сегменту стеку так само буде обчислена на етапі виконання програми.

Перед переходом у захищений режим процесору треба повідомити фізичну адресу таблиці глобальних дескрипторів і її розмір (точніше, межу). Розмір GDT визначається на етапі трансляції. А також треба завершити формування дескрипторів сегментів програми, у яких залишилися незаповненими базові адреси сегментів. Базові (32-бітові) адреси визначаються шляхом множення значень сегментних адрес на 16. Для цього ми зсуваємо значення в AX на 4 біти ліворуч, заносячи 4 старші біти у DL. Командами

```
mov   BX,offset gdt_data
mov   [BX]. base_l,AX
mov   [BX]. base_m,DL
```

вміст AX відправляється в поле base_l дескриптора gdt_data, а вміст DL - у поле base_m.

Аналогічно обчислюються 32-бітові адреси сегментів команд і стеку, що поміщаються в дескриптори gdt_code і gdt_stack.

Наступний етап підготовки до переходу в захищений режим - завантаження в регістр процесора GDTR (Global Descriptor Table Register, регістр таблиці глобальних дескрипторів) інформації про таблицю глобальних дескрипторів. Ця інформація містить у собі лінійну базову адресу таблиці і її межі і розміщується в 6 байтах поля даних, який називається псевдодескриптором. Для завантаження GDTR передбачена спеціальна привілейована команда Lgdt (load global descriptor table, завантаження таблиці глобальних дескрипторів), що потребує в якості операнда ім’я псевдодескриптора. Формат псевдодескриптора наведений на рисунку 3.

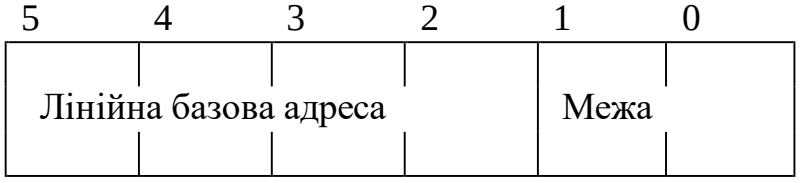


Рисунок 3 – Формат псевдодескриптора

У принципі, ми вже можемо перейти в захищений режим, проте в захищеному режимі заборонені будь-які звертання до функцій DOS або BIOS. Причина цього цілком очевидна - і DOS, і BIOS є програмами реального режиму, у яких широко використовується сегментна адресація реального режиму. У захищеному ж режимі в сегментні регістри завантажуються не сегментні адреси, а селектори. Крім того, звертання до функцій DOS і BIOS здійснюється за допомогою команд int, а в захищеному режимі ці команди призведуть до цілком інших результатів. Таким чином, програму, що працює в захищеному режимі, не можна завершити засобами DOS. Спочатку її треба повернути в реальний режим.

Повернення в реальний режим можна здійснити скиданням процесора. Дії процесора після скидання визначаються однією з чарунок КМОП-мікросхеми - байтом стану відключення, що розташовується за адресою Fh. Зокрема, якщо в цьому байті записаний код Ah, після скидання керування негайно передається за адресою, що витягається з двослівної чарунки 40h:67h, розташованої в області даних BIOS. Таким чином, для підготування повернення в реальний режим ми повинні в чарунку 40h:67h записати адресу повернення, а в байт Fh КМОП-мікросхеми занести код Ah. У рядках

```
mov   AX, 40h
mov   ES,AX
mov   word ptr ES:[67h], offset return
mov   ES:[69h],CS
```

заносимо повну адресу точки повернення return за адресою 0h:67h.

Ще одна важлива операція, яку необхідно виконати перед переходом у захищений режим, полягає в забороні всіх апаратних переривань. Справа в тому, що в захищеному режимі процесор виконує процедуру переривання не так, як у реальному. При надходженні сигналу переривання процесор не звертається до таблиці векторів переривань у першому кілобайті пам'яті, як у реальному режимі, а витягає адресу програми обробки переривання з таблиці дескрипторів переривань, побудованої схоже з таблицею глобальних дескрипторів і розташованій в програмі користувача (або в операційній системі). У прикладі такої таблиці немає, і на час роботи програми переривання прийдеться заборонити. Заборона всіх апаратних переривань здійснюється командою cli.

У рядках

```
mov    AL,0Fh
out    70h,AL
mov   AL, 0Ah
out   71h,AL
```

у порт 70h заноситься код 0Fh, що вибирає для запису байт Fh КМОП-мікросхеми, а потім у порт даних 71h посилається код 0Ah, що визначає режим відновлення (перехід за адресою, яка вибирається з області даних BIOS).

Перехід у захищений режим здійснюється встановленням біту 0 слова стану машини. Оскільки інші біти цього слова нам не відомі, спочатку ми читаємо в регістр AX слово стану машини, потім встановлюємо в ньому біт 0 і, нарешті, записуємо модифіковане слово стану назад у процесор. Всі наступні команди виконуються вже в захищеному режимі.

Хоча захищений режим встановлений, проте налагодження системи ще не закінчені. Дійсно, в усіх сегментних регістрах, які використовуються у програмі зберігаються не селектори дескрипторів сегментів, а базові сегментні адреси, що не мають сенсу в захищеному режимі. Звідси можна зробити висновок, що після переходу в захищений режим програма не повинна працювати, тому що в регістрі CS поки ще немає селектора сегменту команд, і процесор не може звертатися до цього сегменту. У дійсності це не зовсім так.

У процесорі для кожного із сегментних регістрів є так званий тіньовий регістр дескриптора, що має формат дескриптора (рисунок 4). Тіньові регістри недоступні програмісту; вони автоматично завантажуються процесором із таблиці дескрипторів щоразу, коли процесор завантажує відповідний сегментний регістр. Таким чином, у захищеному режимі програміст має справу із селекторами, тобто номерами дескрипторів, а процесор - із самими дескрипторами, що зберігаються в тіньових регістрах. Саме вміст тіньового регістра (у першу чергу, лінійна адреса сегменту) визначає область пам'яті, до якої звертається процесор при виконанні конкретної команди.

	Сегментні регістри	Тіньові регістри		
CS	Селектор	База	Межа	Атрибути
SS	Селектор	База	Межа	Атрибути
DS	Селектор	База	Межа	Атрибути
ES	Селектор	База	Межа	Атрибути
FS	Селектор	База	Межа	Атрибути
GS	Селектор	База	Межа	Атрибути

Рисунок 4. Сегментні та тіньові регістри

У реальному режимі тіньові регістри заповнюються не з таблиці дескрипторів, а безпосередньо самим процесором. Зокрема, процесор заповнює поле бази кожного тіньового регістра лінійною базовою адресою сегменту, отриманим шляхом множення на 16 вмісту сегментного регістра, як це і потрібно в реальному режимі. Тому після переходу в захищений режим у тіньових регістрах знаходяться правильні лінійні базові адреси, і програма буде виконуватися правильно, хоча з погляду правил адресації захищеного режиму вміст сегментних регістрів позбавлений сенсу.

Проте після переходу в захищений режим насамперед варто завантажити у сегментні регістри (і, зокрема, у регістр CS) селектори відповідних сегментів. Це дозволить процесору правильно заповнити всі поля тіньових регістрів із таблиці дескрипторів. Доки ця операція не виконана, деякі поля тіньових регістрів (зокрема, межі сегментів) можуть містити невірну інформацію.

Завантажити селектори в сегментні регістри DS, SS і ES не завдає клопоту. Але як завантажити селектор у регістр CS? Для цього можна скористатися штучно сконструйованою командою дальнього переходу, яка призводить до зміни вмісту IP і CS. Рядки

```
db    0EAh              ;Код команди far jmp
dw    offset continue   ;Зсув
dw    16                ;Селектор сегменту команд
```

демонструють цю методику. У реальному режимі ми помістили б у друге слово адреси сегментну адресу сегменту команд, у захищеному ж ми записуємо в нього селектор цього сегменту (число 16).

Команда дальнього переходу, крім завантаження в CS селектора, виконує ще одну функцію - вона очищує чергу команд у блоці передвиборки команд процесора. Як відомо, у сучасних процесорах із метою підвищення швидкості виконання програми використовується конвейєрна обробка команд програми. Одночасно з виконанням поточної (першої) команди здійснюється вибірка операндів наступної (другої), дешифрування третьої і вибірка з пам'яті четвертої команди. Таким чином, у момент переходу в захищений режим уже можуть бути розшифровані декілька таких команд і обрані з пам'яті їхні операнди. Проте ці дії виконувалися, очевидно, по правилах реального, а не захищеного режиму, що може призвести до порушень у роботі програми. Команда переходу очищує чергу передвиборки, примушуючи процесор заповнити її наново вже в захищеному режимі.

Як вже відзначалося вище, для того, щоб не зруйнувати працездатність DOS, процесор варто повернути в реальний режим, після чого можна буде завершити програму звичайним чином. Перейти в реальний режим можна різними засобами; у цьому прикладі скидання процесора виконується засиланням команди FEh у порт 64h контролера клавіатури. Ця команда збуджує сигнал на одному з виводів контролера клавіатури, що насамкінець призводить до появи сигналу скидання на виводі RESET мікропроцесора. Перед виконанням скидання поточний вміст SP зберігається в чарунці real_sp, щоб після переходу в реальний режим можна було відновити стан стеку.

Після скидання процесор працює в реальному режимі, причому керування передається програмам BIOS. BIOS аналізує вміст байту стану відключення (Fh) КМОП-мікросхеми і, оскільки ми записали туди код 0Ah, здійснює передачу керування за адресою, що зберігається в чарунці 40h:67h області даних BIOS. У нашому випадку перехід здійснюється на помітку return.

Команда hlt (halt, останов) дозволяє організувати чекання скидання процесора, що виконується не миттєво.

Передача керування на мітку return здійснюється програмами BIOS, які звичайно ж використовують регістри процесора. Зокрема, регістри SS:SP і DS вказують на поля даних BIOS, і їх варто ініціювати заново.

Для відновлення працездатності системи варто дозволити переривання, після чого програма може працювати вже в реальному режимі. У розглянутому прикладі для перевірки працездатності системи на екран виводиться деякий текст за допомогою функції DOS 09h. Для наочності в нього включені Esc-послідовності зміни кольору символів, тому програму варто виконувати при встановленому драйвері ANSI.SYS.

Програма завершується звичайним чином функцією DOS 4Ch. Нормальне завершення програми і перехід у DOS теж якоюсь мірою свідчить про її правильність.

ЗАВДАННЯ

Розробити та відлагодити програму, яка:

- описує таблицю глобальних дескрипторів;
- описує дескриптори сегменту даних, сегменту команд, сегменту стеку програми;
- ініціалізує дескриптори, завантажує в регістр процесора GDTR дані про таблицю глобальних дескрипторів та переводить ЦП у захищений режим;
- виконує $10N$ ітерацій циклу, де N – порядковий номер студента у списку журнала, та повертає процесор у реальний режим.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Які можливості архітектури МП реалізуються при переході в захищений режим?
2. Що таке дескриптор сегменту?
3. Яку інформацію зберігає дескриптор сегменту?
4. Як формується лінійна адреса в захищеному режимі?
5. Що таке таблиця глобальних дескрипторів?
6. Чи можна модифікувати сегменти команд в захищеному режимі під час виконання програми?
7. Що таке регістр таблиці глобальних дескрипторів?
8. Які таблиці треба підготувати для виходу в захищений режим?
9. У який спосіб можна повернутися з захищеного режиму до реального?
10. Яке призначення тіньових регістрів? Чи доступні вони програмісту?

Лабораторна робота № 8

ТЕМА: **Програмування текстової відеопам'яті у захищеному режимі**.

МЕТА: Навчитись виводити текстову інформацію на екран комп'ютера у захищеному режимі роботи процесора шляхом програмування текстової відеопам'яті.

Короткі теоретичні відомості

Під час увімкнення комп'ютера, у процесорі автоматично встановлюється режим реальної адреси (реальний режим). Перехід у захищений режим здійснюється програмно. Оскільки багато деталей функціонування процесора в реальному і захищеному режимах істотно відрізняються, програми, призначені для захищеного режиму, повинні бути написані особливим чином. Реальний і захищений режими є несумісними.

Розглянемо програму, яка у захищеному режимі виводить рядок символів з заданим атрибутом (колір символа і його фон) на монітор шляхом програмування текстової відеопам'яті комп'ютера.

```
.386P                  ;Дозвіл трансляції привілейованих команд
                        ;Структура для опису дескрипторів сегментів
descr    struc
limit     dw    0         ;Межа (біти 0..15)
base_l   dw    0        ;База, біти 0..15
base_m   db    0        ;База, біти 16..23
attr_1      db    0        ;Байт атрибутів 1
attr_2   db    0        ;Межа (біти 16..19) і атрибути 2
base_h   db    0        ;База, біти 24..31
descr    ends

;Сегмент даних
data          segment   use16      ;16-розрядна програма
;Таблиця глобальних дескрипторів GDT
gdt_null    descr <0,0,0,0,0,0>  ;Селектор 0 - обов'язковий
                                  ;нульовий дескриптор
gdt_data   descr <data_size-1,0,0,92h,0,0> ;Селектор 8,сегмент даних
gdt_code   descr <code_size-1,0,0,98h,0,0> ;Селектор 16,сегмент команд
gdt_stack   descr <255,0,0,92h,0,0>  ;Селектор 24, сегмент стеку
gdt_scree   descr <4095,8000h,0Bh,92h,0,0>  ;Селектор 32,відеобуфер
gdt_size=$-gdt_null   ;Розмір GDT
pdescr   dq 0         ;Псевдодескриптор для lgdt
real_sp  dw 0         ;Змінна для збереження SP
sym      db 1         ;Символ для виведення на екран
attr     db 1eh       ;Його атрибут
mes      db 27, '[31;42m Повернулися в реальний режим!  ',27,'[0m$'
data_size=$-gdt_null  ;Розмір сегменту даних
data          ends    ;Кінець сегменту даних

;Сегмент команд
text     segment   'code' use16     ;16-розрядна програма
         assume CS:text, DS:data
main     proc
```

```
            mov   AX, data              ;Ініціалізація DS
        mov   DS, AX                ;у реальному режимі
;Знайдемо 32-бітову лінійну адресу сегменту даних
        mov   DL,0
        shld  DX,AX,4
        shl   AX,4
        mov   BX,offset gdt_data
        mov   [BX]. base_l,AX
        mov   [BX]. base_m,DL
;Знайдемо 32-бітову лінійну адресу сегменту команд
        mov   AX, CS
        mov   DL,0
        shld  DX,AX,4
        shl   AX,4
        mov   BX,offset gdt_code
        mov   [BX]. base_l,AX
        mov   [BX]. base_m,DL
;Знайдемо 32-бітову лінійну адресу сегменту стеку
        mov   AX,SS
        mov   DL,0
        shld  DX,AX,4
        shl   AX,4
        mov   BX,offset gdt_stack
        mov   [BX]. base_l,AX
        mov   [BX]. base_m,DL
```

;Підготуємо дескриптор і завантажимо регістр GDTR

```
        mov   BX,offset gdt_data    ;Адреса GDT
        mov   AX,[BX].base_l;Одержимо та занесемо у pdescr
        mov   word ptr pdescr+2, AX ;базу, біти 0..15
        mov   DL,[BX]. base_m;Одержимо і занесемо у pdescr
        mov   byte ptr pdescr+4, DL ;базу, біти 16..23
        mov   word ptr pdescr, gdt_size-1 ;Межа GDT
        lgdt  pdescr              ;Завантажимо регістр GDTR
 ;Підготуємося до повернення в реальний режим
        mov   AX, 40h         ;Налаштуємо ES на область
        mov   ES,AX           ;даних BIOS
  mov word ptr ES:[67h],offset return;Зсув точки повернення
        mov   ES:[69h],CS     ;Сегмент точки повернення
;Підготуємося до переходу в захищений режим
        cli                 ;Заборона апаратних переривань
        mov   AL,0Fh        ;Вибірка байту стану відключення
        out   70h,AL        ;Порт КМОП-мікросхеми
        mov   AL, 0Ah       ;Встановлення режиму відновлення
        out   71h,AL        ;після скидання процесора
```

;Перехід в захищений режим

```
        smsw  AX            ;Одержимо слово стану машини
        or    AX,1          ;Встановимо біт PE
        lmsw  AX            ;Запишемо слово стану
```

;Процесор працює в захищеному режимі

```
;Завантажуємо в CS:IP селектор:зсув точки continue
        db    0EAh              ;Код команди far jmp
        dw    offset continue  ;Зсув
        dw    16                ;Селектор сегменту команд
continue:
;Робимо щоб адресувалися дані
        mov   AX,8          ;Селектор сегменту даних
        mov   DS,AX
;Робимо щоб адресувався стек
        mov   AX,24         ;Селектор сегменту стеку
        mov   SS,AX
```

```
    ;Ініціалізуємо ES і виводимо символи
         mov   AX,32       ;Селектор сегменту відеобуферу
         mov   ES,AX
         mov   BX,800      ;Початковий зсув на екрані
         mov   CX,640      ;Кількість виведених символів (циклів)
         mov  AX,word ptr sym    ;Початковий символ з атрибутом
screen:  mov   ES:[BX],AX  ;Вихід у відеобуфер
         add   BX,2        ;Зсунемося у відеобуфері
         inc   AX          ;Наступний символ
         loop  screen      ;Цикл виведення на екран
;Повернння ЦП у реальний режим
         mov   real_sp,SP  ;Збережемо SP
         mov   AL,0FEh     ;Команда скидання процесора
         out   64h,AL      ;у порт 64h
         hlt          ;Зупиняемо процесор до кінця скидання
;Процесор працює в реальному режимі
;Відновлення операційного середовища реального режиму
return:  mov   AX,data     ;Відновимо адресуємість даних
         mov   DS,AX
         mov   SP,real_sp  ;Відновимо адресуємість стеку
         mov   AX,stk
         mov   SS,AX
         sti               ;Дозволимо апаратні переривання
         mov   AH, 09h    ;Функція виведення на екран рядка
         mov   DX, offset mes ;Адреса рядка
         int   21h
         mov   AX, 4C00h   ;Завершимо програму звичайним чином
         int   21h
main     endp              ;Кінець головної процедури
code_size=$-main           ;Розмір сегменту команд
text     ends              ;Кінець сегменту команд

stk      segment   stack 'stack' ;Початок сегменту стеку
         db   256 dup ('^')
stk      ends                    ;Кінець сегменту стеку
         end main                ;Кінець програми
```

Дескриптор ***gdt_screen*** описує відеосторінку 0 текстової відеопам'яті. Розмір відеосторінки, як відомо, складає 4096 байт, тому в поле межі записано число 4095. Базова фізична адреса відеосторінки відома (B8000h). Молодші 16 бітів бази (число 8000h) заповнюють слово base_l дескриптора, біти 16...19 (число 0Bh) – байт base_m. Біти 20...31 базової адреси рівні 0, оскільки відеобуфер розміщується в першому мегабайті адресного простору.

Фрагмент програми

```
mov AX,32  ;Селектор сегменту відеобуфера mov
ES,AX
mov   BX,800     ;Початковий зсув на екрані
mov   CX,640     ;Кількість виведених символів
mov   AX,word ptr sym
screen: mov   ES:[BX],AX ;Вихід у відеобуфер
add   BX,2       ;Зсунемося у відеобуфері
inc   AX         ;Наступний символ
loop  screen     ;Цикл виведення на екран
```

забезпечує циклічне виведення символа (змінна `sym`) з атрибутом 1eh (`attr`).

ЗАВДАННЯ

Розробити та відлагодити програму, яка:

- описує таблицю глобальних дескрипторів;
- описує дескриптори відеосторінки 0 текстової відеопам’яті, сегментів даних, сегменту команд, сегменту стеку програми;
- ініціалізує дескриптори, завантажує в регістр процесора GDTR дані про таблицю глобальних дескрипторів та переводить ЦП у захищений режим;
- виводить рядок символів (ПІБ студента, група) з атрибутом згідно варіанта:

Номер варіанта	Колір символа	Колір фону
o	чорний	білий
o	синій	рожевий
o	червоний	синій
o	фіолетовий	жовтий
o	жовтий	синій
o	фіолетовий	жовтий
o	жовтий	блакитний
o	зелений	жовтий
o	фіолетовий	сірий
o	синій	блакитний
o	коричневий	білий
o	зелений	рожевий
o	фіолетовий	синій
o	білий	жовтий
o	синій	зелений
o	коричневий	жовтий
o	зелений	блакитний
o	фіолетовий	яскраво-білий
o	білий	сірий
o	чорний	салатовий
o	синій	білий
o	червоний	рожевий
o	білий	блакитний

o	синій	червоний
o	коричневий	сірий

– повертає процесор у реальний режим.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

11. Яка базова адреса текстової відеопам’яті?
12. Скільки відеосторінок має текстова відеопам’ять?
13. Яку інформацію зберігає дескриптор сегменту?
14. Який об’єм (розмір) відеосторінки текстової відеопам’яті?
15. Що таке таблиця глобальних дескрипторів?
16. Чи можна модифікувати сегменти команд в захищеному режимі під час виконання програми?
17. Яке призначення тіньових регістрів? Чи доступні вони програмісту?
18. Що таке регістр таблиці глобальних дескрипторів?
19. Які таблиці треба підготувати для виходу в захищений режим?
20. У який спосіб можна повернутися з захищеного режиму до реального?

Лабораторна робота № 9

ТЕМА: Програмування режимів EGA та VGA відеоадаптера

МЕТА: Набути навичок роботи з відеоадаптером у режимах EGA та VGA, а також використання засобів BIOS для роботи з відеопам'яттю.

Короткі теоретичні відомості

Для роботи з відеоадаптером в BIOS відведено три функції переривання 10h. *Функція 00h* призначена для встановлення режима дисплея (для виклику: AH = 00h, AL = номер режима); *функція 05h* – вибір активної дисплейної сторінки (для виклику: AH = 05h, AL = номер відеосторінки); *функція 0Fh* – визначення стану відеосистеми (для виклику: AH = 0Fh; повертає: AH = кількість символьних столбців, AL = номер режима, BH = номер активної відеосторінки).

ЗАВДАННЯ

Розробити та відлагодити програму, яка:

- переводить дисплей в заданий користувачем відеорежим;
- забезпечує можливість перемикання відеосторінок;
- виводить на екран інформацію про стан відеосистеми.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

21. Яка базова адреса текстової відеопам'яті?
22. Скілький відеосторінок має текстова відеопам'ять?
23. Яку інформацію зберігає дескриптор сегменту?
24. Який об'єм (розмір) відеосторінки текстової відеопам'яті?
25. Що таке таблиця глобальних дескрипторів?
26. Чи можна модифікувати сегменти команд в захищеному режимі під час виконання програми?
27. Яке призначення тіньових регістрів? Чи доступні вони програмісту?
28. Що таке регістр таблиці глобальних дескрипторів?
29. Які таблиці треба підготувати для виходу в захищений режим?
30. У який спосіб можна повернутися з захищеного режиму до реального?

Лабораторна робота № 10

ТЕМА: **Програмування відеоадаптера та текстової відеопам'яті комп'ютера у реальному режимі адресації**

МЕТА: Набути навичок програмування відеоадаптера у текстовому режимі, текстової відеопам'яті та керування курсором у реальному режимі адресації.

Короткі теоретичні відомості

Для роботи з відеоадаптером в BIOS відведено три функції переривання 10h. *Функція 00h* призначена для встановлення режима дисплея (для виклику: AH = 00h, AL = номер режима); *функція 05h* – вибір активної дисплейної сторінки (для виклику: AH = 05h, AL = номер відеосторінки); *функція 0Fh* – визначення стану відеосистеми (для виклику: AH = 0Fh; повертає: AH = кількість символьних стовбців, AL = номер режима, BH = номер активної відеосторінки).

Для керування курсором в BIOS відведено наступні функції переривання 10h: *01h* – встановлення конфігурації курсора; *02h* – встановлення позиції курсора; *03h* – отримання інформації про позицію та розмір курсора.

Для читання символа та його атрибута у поточній позиції курсора відведено функцію *08h* переривання BIOS 10h.

ЗАВДАННЯ

Розробити та відлагодити програму, яка:

- змінює форму курсора в текстовому режимі роботи відеоадаптера;
- виводить інформацію про стан курсора для кожної відеосторінки;
- шляхом програмування текстової відеопам'яті у реальному режимі адресації виводить на активну відеосторінку рядок символів (прізвище та ініціали) з атрибутами, відповідно до варіанта (таблиця 4.1);

Таблиця 4.1

Номер варіанта	Атрибут	
	Колір символа	Колір фону
o	жовтий	синій
o	фіолетовий	жовтий
o	жовтий	блакитний
o	зелений	жовтий

○	фіолетовий	сірий
○	синій	блакитний
○	коричневий	білий
○	зелений	рожевий
○	фіолетовий	синій
○	білий	жовтий
○	синій	зелений
○	коричневий	жовтий
○	зелений	блакитний
○	фіолетовий	яскраво-білий
○	білий	сірий
○	чорний	салатовий
○	синій	білий
○	чорний	білий
○	синій	рожевий
○	червоний	синій
○	фіолетовий	жовтий
○	червоний	рожевий
○	білий	блакитний
○	синій	червоний
○	коричневий	сірий

- виводить на екран інформацію про вказане користувачем знакомісце: символ, його код (hex), атрибут символа: колір, фон (прописом). Наприклад, `2 10 → Символ "1", код 31h, колір білий, фон чорний.`

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

31. Яка базова адреса текстової відеопам'яті?
32. Скільки відеосторінок має текстова відеопам'ять?
33. Яку інформацію зберігає дескриптор сегменту?
34. Який об'єм (розмір) відеосторінки текстової відеопам'яті?
35. Що таке таблиця глобальних дескрипторів?

36. Чи можна модифікувати сегменти команд в захищеному режимі під час виконання програми?
37. Яке призначення тіньових регістрів? Чи доступні вони програмісту?
38. Що таке регістр таблиці глобальних дескрипторів?
39. Які таблиці треба підготувати для виходу в захищений режим?
40. У який спосіб можна повернутися з захищеного режиму до реального?

Лабораторна робота № 11

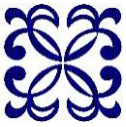
ТЕМА: Розробка та ініціалізація власних шрифтів і нестандартних символів за допомогою програмування знакогенератора відеоадаптера ПК

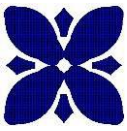
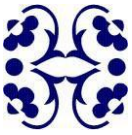
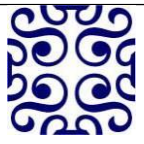
МЕТА: Набути навичок розробки власних шрифтів тексту і нестандартних символів, завантаження їх матриць до знакогенератора відеоадаптера; засвоїти отримані знання та вміння з програмування тектової відеопам'яті комп'ютера.

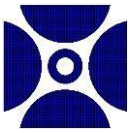
ЗАВДАННЯ

1. Розробити матриці символів, які забезпечують виведення у текстовому режимі псевдосимвол відповідно до таблиці 5.1, обравши та обґрунтувавши перед цим доцільний розмір шрифта – висоту символа у графічних точках (пікселях).
2. Розробити та відлагодити програмне забезпечення, яке:
   2.1. Перепрограмовує відеоадаптер на відповідний розмір шрифта, завантажує у блок знакогенератора таблицю символів, матриці яких розроблено у завданні 1;
   2.2. Програмуванням текстової відеопам'яті комп'ютера встановлює атрибут всіх символів відеосторінки 0 відповідно до таблиці 5.1;
   2.3. Забезпечує виведення на монітор розробленого псевдосимвола.

Таблиця 5.1

Номер варіанта (видається викладачем)	Псевдо-символ	Атрибут	
		Колір символа	Колір фону
○		синій	зелений
○		зелений	рожевий
○		червоний	синій
○		фіолетовий	жовтий

○		червоний	рожевий
○		білий	блакитний
○		синій	червоний
○		фіолетовий	синій
○		білий	жовтий
○		синій	жовтий
○		коричневий	жовтий
○		зелений	блакитний
○		фіолетовий	яскраво-білий
○		білий	сірий
○		чорний	салатовий
○		синій	білий
○		чорний	білий

○		синій	рожевий
○		фіолетовий	жовтий
○		жовтий	блакитний
○		зелений	жовтий
○		фіолетовий	сірий
○		синій	блакитний
○		червоний	синій
○		коричневий	білий

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Яке переривання BIOS забезпечує роботу з відеоадаптером?
2. Яка базова адреса текстової відеопам’яті?
3. Скілький відеосторінок має текстова відеопам’ять комп’ютера?
4. Який об’єм (розмір) відеосторінки текстової відеопам’яті?
5. Яким чином формується атрибут символа?
6. Де зберігаються атрибути символів та яким чином їх моливо встановити/ отримати?
7. Яка ширина символа у текстовому режимі? Чому вона є незмінною?
8. Чому не можливо встановити яскраво-білий фон символа?
9. Які функції переривання 10h забезпечують роботу зі шрифтами?
10. Які бувають розмірності матриць символів?

Лабораторна робота № 12

ТЕМА: Програмування принтера

МЕТА: Набути навичок програмування принтера Epson, розробки й виведення на друк власних шрифтів і нестандартних символів, друку графічної інформації.

ЗАВДАННЯ

Розробити та відлагодити програмне забезпечення принтера, яке:

3. Скидає принтер у початковий стан, про що виводить відповідне повідомлення на екран.
4. Перевіряє готовність принтера (за допомогою байту стану) та виводить на екран відповідне повідомлення.
5. Якщо принтер готовий:
   - *2.4.* Видає (N mod 10) звукових сигналів, де N – номер варіанта.
   - *2.5.* Друкує прізвище та ім’я студента-розробника ПЗ.
   - *2.6.* Переводить сторінку паперу.
   - *2.7.* Друкує рядок “КНТУ” нестандартним шрифтом (як сукупність символів у звичайному режимі, або як зображення у графічному).

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Архітектура комп’ютерних систем: конспект лекцій для студентів усіх форм навчання з курсу «Архітектура комп’ютерних систем» / Укладачі: Голотенко О.С. – .Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 120 с.

2. Тарарака В.Д. Т19 Архітектура комп'ютерних систем: навчальний посібник. – Житомир : ЖДТУ, 2018. – 383 с

3. Сучасні напрямки комп’ютерної та мікропроцесорної техніки Розділ 1. Основні тенденції розвитку комп'ютерної і мікропроцесорної техніки. Розділ 2 Характеристики ARM і Cortex процесорів: конспект лекцій. [Електронний ресурс]: для студ. спеціальності 171 Електроніка, спеціалізації «Електронні компоненти та системи» /Т. О. Терещенко, Ю.С. Ямненко; КПІ ім. Ігоря Сікорського; уклад,– Електронні текстові данні 1 файл: 5,248 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2020. – 68 с.

4. Архітектура комп’ютерів. Особливості використання комп’ютерів в ІС : навчальний посібник / С. В. Кавун, І. В. Сорбат. – Харків : Вид. ХНЕУ, 2010. – 256 с.

5. Мартін Р. Чиста архітектура. – Фабула, 2019. – 368 с

6. Тарарака В.Д. Архітектура комп'ютерних систем: навчальний посібник. – Житомир: ЖДТУ, 2018. –383с

7. Матвієнко М.П. Архітектура комп’ютерів: Навчальний посібник / Матвієнко М.П., Розен В.П., Закладний О.М. – К.: Ліра-К, 2019. – 264 с.

8. Минайленко Р.М. Архітектура комп’ютерів: Навчальний посібник / Минайленко Р.М., Конопліцька-Слободенюк О.К., Гермак В.С. – М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2022. – 105с
http://dspace.kntu.kr.ua/jspui/handle/123456789/12260

9. Bartlett J. Programming from the Ground Up. — http:// www.freebookcentre.net/Compu terScience-BooksDownload/Programming -fromthe-Ground-Up-(J.-Bartlett).html

10. Світ електроних схем. [Електронний ресурс] – Режим доступу: ttp://ua.nauchebe.net

11. Дистанційна освіта ЦНТУ. – URL: https://moodle.kntu.kr.ua/?lang=en