МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
Кафедра “Електронні обчислювальні машини**”**

Методичні вказівки до виконання лабораторних робіт

з дисципліни

“Системне програмне забезпечення”

для студентів спеціальностей
7.091501: "Комп'ютерні системи та мережі"
7.091502: ”Системне програмування”

укладачі ст. викл. Іванов О.Ю., ст. викл. Шевченко О.Г.

Затверджено
на засіданні кафедри

Протокол №6, від
21.01.10р.

Донецьк – 2010р.

Навчальне видання

Методичні вказівки
До виконання лабораторних робіт з дисципліни
"Системне програмне забезпечення"

>

Укладачі Шевченко Ольга Георгіївна,
Іванов Олександр Юрієвич

УДК 681.3

Методичні вказівки до виконання лабораторних робіт з дисципліни "Системне програмне забезпечення" (для студентів спеціальностей 7.091501: "Комп'ютерні системи і мережі",7.091502*:* "Системне програмування")/ Автори: О.Ю.Іванов , О.Г.Шевченко –Донецьк :ДонНТУ, 2010.

Приводиться порядок виконання лабораторних робіт з дослідження програмно-аппаратних ресурсів реального та захищеного режимів роботи персональних ЕОМ з використанням алгоритмічної мови С. Методичні вказівки містять програмні реалізації рішення принципових питань по кожній темі, що покликані допомогти при виконанні індивідуального завдання по кожній лабораторній роботі.

ЛАБОРАТОРНА РОБОТА №1

ОРГАНІЗАЦІЯ ОБРОБКИ ПЕРЕРИВАНЬ

***Мета* роботи** - придбання практичних навичок щодо написання власних оброблювачів апаратних та програмних переривань

Апаратні переривання це фізичні сигнали, якими контролер пристрою повідомляє процесор про необхідність обробити запит від будь-якого пристрою ПК. Коли виникає переривання, процесор переключається з поточної програми на модуль (у подальшому підпрограму) обробки переривання (у загальному випадку драйвер), а після його завершення повертається до перерваної програми. При переході на підпрограму обробки переривання в стек заноситься вміст регістра прапорів мікропроцесора, а також адреса повернення у форматі сегмент: зсув. Початкові адреси підпрограм обробки переривань прийнято називати векторами. Кожен вектор у реальному режимі роботи процесора має довжину чотири байти і представляється у формі сегмент: зсув. Молодші 1024 байта оперативної пам'яті містять таблицю векторів переривань. Вектор для переривання 0 починається з адреси 0000:0000, переривання 1 – з адреси 0000:0004, переривання 2 - з адреси 0000:0008 і т.д.

Програмні переривання генеруються наявно або командою процесора INT.

1. Виклик програмного переривання з прикладної програми

Мова програмування **C** дозволяє використовувати ряд функцій для роботи з програмними перериваннями.

1.1.Функція **int86()** викликає будь-яке системне або BIOS переривання

int int86(int intno, union REGS *inr, union REGS *outr),

де **intno** - номер переривання;
inr - покажчик на об'єднання вхідних регістрів;
outr - покажчик на об'єднання вихідних регістрів.

union REGS - визначене у файлі dos.h і є перекриттям двох типів структур: **struct WORDREGS x**; - забезпечує доступ до 16-бітових регістрів AX,BX,CX,DX, SI,DI, і до регістрів прапорів, а **struct BYTEREGS h**; - для доступу до відповідних 8-бітним половинам регістрів: AH,AL, BH,BL, CH,DH і DH,DL.

```
struct WORDREGS {unsigned int ax,bx,cx,dx, si,di, cflag,flags;};
struct BYTEREGS  {igned char al,ah, bl,bh, cl,ch, dl,dh;};
union REGS       {struct  WORDREGS x;
                    struct  BYTEREGS h;
                  };
```

Функція **int86()** спочатку копіює значення регістрів з структури ***inr** у відповідні регістри мікропроцесора, а потім генерує програмне переривання з номером **intno.** Після повернення з переривання функція копіює вміст регістрів мікропроцесора у відповідні елементи структури ***outr**.

Функція **int86()** повертає значення регістра **AX**, що той має після виходу з переривання. У випадку помилки встановлюється ненульове значення перемінної **outr.x.cflag**, і в глобальній змінній **_doserrno** запам'ятовується код помилки.

Приклад . Визначити значення поточного часу.

```
#include<stdio.h>
#include<dos.h>
void main(void)
{     union REGS inr, outr;
   inr.h.ah = 0x2C; /*  Номер функції  */
   int86(0x21, &inr, &outr);
   printf("Поточний час %u:%u:%u.%u\n", outr.h.ch, outr.h.cl,
          outr.h.dh, outr.h.dl);
}
```

Для генруванії програмного переривання з передачею параметрів через сегментні регістри використовується функція **int86x()**:

int int86x(int intno, union REGS *inr, union REGS *outr, struct SREGS *segr),

де **intno** - номер переривання;
inr - покажчик на об'єднання вхідних регістрів;
outr - покажчик на об'єднання вихідних регістрів;
segr - покажчик на структуру сегментних регістрів.

Значення сегментних регістрів ES, CS, SS і DS передаються через структуру SREGS, що визначена у файлі dos.h:

struct SREGS { unsigned int es, cs, ss, ds;};

Функція **int86x()** спочатку копіює значення регістрів з структури ***inr** у відповідні регістри мікропроцесора, потім зберігає вміст регістра **DS** і записує нові значення з структури ***segr** у регістри **DS** і **ES**. Після цього вона генерує програмне переривання з номером **intno**. При поверненні з переривання функція **int86x()** копіює вміст регістрів мікропроцесора у відповідні елементи структури ***outr** і відновлює значення регістра **DS**. Функція **int86x** повертає значення так само як і **int86()**.

Приклад. Призначити заданий каталог поточним.

```
#include <stdio.h>
#include <dos.h>
int main(void)
{  union REGS reg;
   struct SREGS sreg;
   char  *buffer = "C:\\WINDOWS";
   reg.h.ah = 0x3B;
   sreg.ds = FP_SEG(buffer);
   reg.x.dx = FP_OFF(buffer);
   int86x(0x21, &reg, &reg, &sreg);
      if(reg.x.cflag) { perror("Помилка intdosx"); return 1; }
   printf("Тепер поточний каталог - C:\\WINDOWS\n");
   return 0;
}
```

Для виклику функцій з набору переривання 21h використовується функція **intdos()**.

int intdos(union REGS *inr, union REGS *outr),

де **inr** - покажчик на об'єднання вхідних регістрів;

outr - покажчик на об'єднання вихідних регістрів.

Її варто використовувати, коли системна функція не вимагає передачі параметрів через сегментні регістри **ES** і **DS**. Будь-яке програмне переривання можна викликати за допомогою функції int86(), а функцію intdos() завжди можна замінити на int86() з номером переривання 21h.

```
#include <dos.h>
int main(void)
{    union REGS rr;
   ir.h.ah = 0x2C;
   intdos(&rr, &rr);
   printf("Поточний час %u:%u:%u.%u\n", rr.h.ch, rr.h.cl,
          rr.h.dh, rr.h.dl);
}
```

Для генерування програмного переривання 21h, з передачею параметрів через сегментні регістри використовується функція **intdosx()**, що має наступний синтаксис:

int intdosx(union REGS *inr, union REGS *outr,struct SREGS *segr)

Робота цієї функції цілком відповідає раніше розглянутому виклику функції:
int86x(0x21, &inr, &outr, &segr);

Альтернативний спосіб виклику програмних переривань надає функція **intr()**, що має наступний синтаксис:

void intr(int intno, struct REGPACK *preg),

де **intno** - номер переривання;
preg - покажчик на структуру регістрів і прапорів.

Функція **intr()** генерує програмне переривання так, як це робить функція **int86x().** При цьому вміст регістрів мікропроцесора передається через **структуру REGPACK**, визначену в файлі **dos.h** у такий спосіб:

```
struct  REGPACK {
     unsigned      r_ax, r_bx, r_cx, r_dx;
     unsigned      r_bp, r_si, r_di, r_ds, r_es, r_flags;
};
```

Функція **intr()** копіює значення з ***preg** у відповідні регістри, а потім використовує ассемблерну команду **INT** для генерування програмного переривання з номером **intno**. Після повернення з переривання функція **intr()** копіює вміст регістрів мікропроцесора у відповідні елементи структури ***preg.**

Розглянемо приклад. Функція 3Bh переривання 21h виконує установку нового поточного каталогу. Ім'я каталогу задається рядком символів у форматі [дисковод:]шлях. Функція повинна одержати в якості аргументу покажчик на цей рядок символів, переданий через пару регістрів DS:DX. У випадку виникнення помилки при зміні каталогу встановлюється прапор переносу CF у регістрі прапорів мікропроцесора.

Для виклику підмножини функцій переривання **21h**, що не вимагають аргументів чи потребують передачі параметрів тільки через регістри DX і AL, можна використовувати функцію **bdos().** Її синтаксис:

int bdos(int dosfun, unsigned dosdx, unsugned dosal),

де **dosfun** - номер функції DOS;
dosdx - значення регістра DX;
dosal - значення регістра Al.

Функція **bdos()** спочатку копіює значення своїх аргументів **dosdx** і **dosal** у відповідні регістри мікропроцесора, а потім викликає функцію **dosfun** переривання 21h. Після виходу з переривання функція **bdos()** повертає значення регістра AX.

Для виклику підмножини функцій переривання 21h, що вимагають передачі адреси (покажчика) через регістри DS:DX і аргумента через регістр AL, можна використовувати функцію **bdosptr().** Її синтаксис:

int bdosptr(int dosfun, void *argument, unsugned dosal),

де **dosfun** - номер функції DOS;
argument - адреса аргумента, що поміщається в DS:DX;
dosal - значення регістра Al.

Функція **bdosptr()** спочатку заносить адресу аргументу argument у регістри DS:DX, копіює значення **dosal** у регістр мікропроцесора AL, а потім викликає функцію **dosfun** переривання 21h. Після виходу з переривання функція **bdosptr()** повертає значення регістра AX. У випадку помилки значення, що повертається, дорівнює -1, а глобальні змінні errno та _doserrno містять код помилки.

Нарешті, макрос **void geninterrupt(int intno)** викликає асемблерну команду INT, що генерує програмне переривання з номером **intno**. При цьому для передачі аргументів і одержання результатів можна використовувати псевдозмінні

_AX _AL _AH _SI _ES
_BX _BL _BH _DI _SS
_CX _CL _CH _BP _CS
_DX _DL _DH _SP _DS
_FLAGS,

які відповідають регістрам загального призначення, спеціальним і сегментним регістрам мікропроцесора. Наприклад, у наступної програми після повернення з переривання 12h через псевдоперемінну _AX зчитується вміст регістра AX, що показує обсяг звичайної пам'яті комп'ютера в кілобайтах:

```
#include<stdio.h>
#include<dos.h>
int main(void)
{
   int size_memory;
   geninterrupt(0x12);
   size_memory = _AX;
   printf("Обсяг ОЗП %d Кбайт\n", size_memory);
   return 0;
}
```

3. Заборона/дозвіл апаратних переривань

Для керування апаратними перериваннями в IBM PC використовується мікросхема програмувального контролера переривань Intel 8259. Оскільки в кожен момент часу може надійти не один запит, мікросхема має схему пріоритетів (16 рівнів). Рівні позначаються скороченнями від IRQ0 до IRQ15.

Максимальний пріоритет відповідає рівню 0. Запити на переривання 0-7 відповідають векторам переривань від 8h до Fh; запити на переривання 8-15 обслуговуються векторами від 70h до 77h. У табл. 1 наведені призначення цих переривань.

Таблиця 1

Номер запиту	Апаратні переривання в порядку пріоритету
IRQ 0	таймер (номер переривання завжди зайнятий)
1	клавіатура, -//-
2	канал вводу/виводу=другий контролер
8	годинник реального часу з автономним живленням(RTC),-
9	програмно переводиться в IRQ2 (тільки AT)
10	вільний
11	вільний
12	контролер миші PC-2
13	співпроцесор (тільки AT), завжди зайнятий
14	контролер IDE HDD (перший канал)
15	контролер IDE HDD (другий канал)
3	COM1 (COM2 для AT),можна відключити. і номер забрати
4	COM2 (модем, COM1 для AT)
5	фіксований диск НДД, LPT2
6	контролер НГМД
7	LPT1

Переривання від таймеру відбувається 18,2 рази в секунду. Йому наданий максимальний пріоритет, оскільки, якщо він буде постійно губитися, то будуть невірними показання системних годин. Переривання від клавіатури викликається при натисканні чи відпусканні клавіші; воно викликає ланцюг подій, що звичайно закінчується тим, що код клавіші міститься в буфер клавіатури, звідки він потім може бути отриманий через програмне переривання.

Програмно можна заборонити апаратні переривання, перераховані в табл. 1. Ці переривання та інші апаратні переривання, що виникають при деяких помилках (таких, як ділення на нуль), не можуть бути масковані. Існують дві причини для заборони апаратних переривань. У першому випадку блокуються всі переривання для того, щоб критична частина коду була виконана цілком, перш ніж машина зробить яку-небудь іншу дію. Наприклад, переривання забороняють при зміни вектора апаратного переривання, щоб уникнути виконання переривання, коли вектор змінений не повністю. В другому випадку маскуються тільки визначені апаратні переривання. Це робиться, коли деякі переривання можуть взаємодіяти з операціями, критичними до часових інтервалів. Наприклад, точно розрахована за часом процедура вводу/виводу не може собі дозволити бути перерваною тривалим дисковим перериванням.

Обслуговування переривання процесором залежить від значення прапора переривання IF у регістрі прапорів. Коли цей прапор дорівнює 0, дозволені усі переривання, що дозволяє регістр маски контролера переривання. Коли прапор дорівнює 1, усі апаратні переривання заборонені. Для роботи з прапором переривання використовуються наступні функції:

void disable()
void enable()

Треба уникати відключення переривань на тривалий період. При виклику обробників переривань прапор IF встановлюється в 1, тому при написанні власного обробника треба викликати функцію enable(), якщо можна допустити апаратні переривання.

Для маскування визначених апаратних переривань потрібно послати необхідний ланцюжок бітів у порт з адресою 21h, що відповідає регістру маски переривань (IMR). Регістр маски на другій мікросхемі 8259 для AT (IRQ8-15) має адресу порту A1h.

У фрагменті, що нижче приводиться, спочатку блокується дискове переривання IRQ5, а потім наприкінці програми воно відновлюється:

```
#include<dos.h>
int main(void)
{
     outportb(0x21, 0x40); /* Пересилання в порт 21h байта 40h */
     outportb(0x21, 0);    /* Пересилання в порт 21h байта  0  */
return 0;
}
```

Після реалізації ACPI и IRQ Sharing на всіх материнських платах, починаючи з плат для Pentium 1 (чіпсети Intel), склалась розкладка переривань, яка існує до цього часу:

Лінія IRQ	Пристрій
0	Системний таймер
1	Клавіатура
2	Cascad (вивід на другу мікросхему контролеру ліній перериваннь)
8	Годинник реального часу
9	ACPI Controller
10	Вільний
11	USB
12	PS2
13	Сопроцессор
14	IDE Primary (Контроллер жорстких дисків)
15	IDE Secondary (Контроллер жорстких дисків)
3	Com Port 1 (Миша)
4	Com Port 2 (Модем)
5	Вільний
6	Floppy
7	LPT (Принтер)

Довідатися, як розподілені номери переривань, можна декількома способами:

- На початку завантаження комп'ютера з'являється текстова таблиця конфігурації. Після її йде перелік PCI-пристроїв та з призначених їм номерів IRQ.
- Для Windows 9x. У "панелі керування" є опція "Система", у ній - "Пристрої". Вибираємо властивості пристрою "Комп'ютер", і в ньому будуть перераховано всі пристрої і призначених їм IRQ.
- Для Windows 2000. Для перегляду списку IRQ потрібно скористатися стандартною утилітою (Панель керування/Адміністрування/Керування комп'ютером/Відомості про систему/Ресурси апаратури).
- Для Windows XP - /Пуск/Програм./Стандарт./Служб./ Відомості про систему/Ресурси апаратури

4. Власний оброблювач переривання

Може існувати кілька причин для написання власної підпрограми обробки переривання – доповнити або змінити стандартний обробник, обслуговувати новий пристрій тощо.

На мові С підпрограма обробки переривання повинна бути описана як функція з ключовим словом **interrupt**, наприклад:

```
void interrupt get_out(void)
{
        // Алгоритмічна частина
}
```

Для установлення нового вектора переривання використовується функція **setvect**, що має синтаксис:

void setvect(int intno, void interrupt(*handler)()),

де **intno** - номер переривання;

handler- покажчик на новий оброблювач переривання.

Дана функція встановлює адресу оброблювача переривання, задану аргументом **handler**, як новий вектор для переривання **intno**. Програма користувача перед заміною вектора переривання повинна зберегти старе значення вектора, а перед закінченням своєї роботи відновити його. Адресу обробника переривання повертає функція getvect().

void interrupt (*getvect(int intno))(),

де **intno** - номер переривання.

Змінна, якій буде привласнене значення, що повертається, повинна бути описана як покажчик на функцію типу interrupt,

```
void interrupt (*oldfunc)(void);
oldfunc = getvect(5);
```

Для відновлення вектора переривання варто викликати описану вище функцію setvect():

```
setvect(5, oldfunc);
```

Наступний приклад ілюструє установку на початку програми нового оброблювача переривання 5h і відновлення системного оброблювача наприкінці її:

```
/*  Файл програми має розширення .C*/
#include <stdio.h>
#include <dos.h>
void interrupt get_out(void);  /*Прототип функції типу interrupt*/
void interrupt (*oldfunc)(void); /*Покажчик на функцію interrupt*/
int looping = 1;
int main(void)
{
  oldfunc  = getvect(5); /* Збереження старого вектора int 5*/
  setvect(5, get_out);   /* Установка нового оброблювача int 5*/
  printf("Зациклення...\n");
  printf("Натисніть <Shift><PrtSc> для виходу з циклу\n");
  while (looping);
  setvect(5, oldfunc);   /* Відновлення старого вектора */
  printf("Тепер <Shift><PrtSc> викликає печатку копії екрана\n");
  return 0;
}
void interrupt get_out(void) // Новий оброблювач переривання 5
{
  looping = 0; /*зміна глобальної перемінної для виходу з циклу */
}
```

ПРИМІТКА: повинні бути встановлені опції компілятора:

Compiler/CodeGeneration - Test stack overflow =off;
Compiler/CodeGeneration/Optimization - Use register variables =off;
Linker - Stack Warning =off.

Користувальницький оброблювач переривання може не тільки цілком замінити системний оброблювач, але і доповнити його. Загальна структура програми в цьому випадку має вигляд:

```
#include <dos.h>
.................
void interrupt handler(void); /* Прототип функції типу interrupt*/
void interrupt (*oldfunc)(void);/* Покажчик на функцію interrupt*/
int main(void)
{
  oldfunc  = getvect(номер_переривання);// Збереження старого
                                         //вектора переривання
  setvect(номер_переривання, handler); // Установка нового
                                       // оброблювача переривання
setvect(номер_переривання, oldfunc);   // Відновлення старого
                                       // вектора
  return 0;
}
void interrupt handler(void)   /* Новий оброблювач переривання */
{
/* Перша частина користувальницького оброблювача */
  oldfunc();  // Виклик системного оброблювача з наступним
              //поверненням  у це місце
/* Друга частина користувальницького оброблювача */
}
```

При цьому нова процедура обробки переривання handler() може містити будь-який код як до, так і після виклику системного оброблювача.

4.1. Створення процедури обробки Ctrl-Break

При одночасному натисканні клавіш <Ctrl> і <Break> встановлюється в 1 старший біт байта BIOS_BREAK з адресою 0040:0071 в області даних BIOS. Будь-яка програма може перевірити значення цього біта для визначення стану Ctrl-Break. Потім викликається переривання 1Bh. Звичайно вектор переривання 1Bh указує на системний оброблювач DOS, але будь-яка програма може перехопити цей вектор і тим самим самостійно обробляти Ctrl-Break. Якщо переривання 1Bh викликає системний оброблювач, то останній установлює внутрішній прапор Clrl-Break системи. Прапор указує на те, що повинна бути виконана процедура обробки Ctrl-Break. Керування передається цій процедурі тільки у той момент, коли програма звертається до системної функції, здатної розпізнати цей прапор. Звичайно тільки стандартні функції вводу/виводу розпізнають цей прапор (функції від 1 до С переривання 21h, за винятком функцій 6 і 7). Однак, якщо в файл AUTOEXEC.BAT чи CONFIG.SYS помістити рядок BREAK=ON, тоді звертання до будь-якої системної функції приведе до виклику процедури обробки Ctrl-Break.

Процедура обробки Ctrl-Break дає можливість завершити програму в будь-який момент часу. Коли системна функція розпізнає статус Ctrl-Break, то керування передається процедурі, на яку указує вектор переривання 23h. Система використовує цю процедуру для завершення працюючої програми.

Встановлення нового оброблювача для Ctrl-Break з модифікацією вектора 23h виконує функція:

void ctrlbrk(int (*handler)(void)),

де **handler** - покажчик на функцію, що стане новим оброблювачем Ctrl_Break.

Для забезпечення виходу в систему по закінченні своєї роботи оброблювач повинний завершуватися оператором

return 0;

При цьому перед поверненням у систему автоматично відновлюється первісний вектор переривання 23h.

Приклад установки програмного оброблювача Ctrl-Break:

```
#include<stdio.h>
#include<dos.h>
#define ABORT 0
int c_break(void)
  {
   printf("\nControl-Break включений. Програма перервана ...\n");
   return(ABORT);
  }
int main(void)
  {
    ctrlbrk(c_break);
    for(;;)
      printf("Зациклення ... Для виходу"
             " натисніть <Ctrl><Break>\n");
    return 0;
  }
```

Якщо код повернення в операторі **return**, що завершує оброблювач, відмінний від нуля, то після виконання процедури обробки переривання відбудеться повернення до перерваної програми. Переривання 1Bh іноді називають апаратним перериванням Ctrl-Break на відміну від програмного переривання Ctrl-Break з номером 23h

Приклад. /* Оброблювач апаратного переривання по натисканню Cntr-Break. */

```
#include<dos.h>
#include<stdio.h>
#include<conio.h>
#include<time.h>
void interrupt myprint(void)
  {
   disable();
   cprintf("ВИВІД  З  ОБРОБЛЮВАЧА  ПЕРЕРИВАННЯ... ");
   enable();
  }
int  main(void)
  {
   int i;
   void interrupt (*oldvect)(void);
   oldvect = getvect(0x1B);
   setvect(0x1B, myprint);
   for(i = 0; i <= 30; i++)
```

```
    {
     cprintf("\n\r для виклику оброблювача переривання"
             " натисніть <Ctrl><Break>\n\r");
     sleep(1);
    }
   setvect(0x1B, oldvect);
   cprintf("\n\rпрограма з а в е р ш е н а\n\r");
   return 0;
  }
```

4.2. Особливості користувальницьких оброблювачів апаратних переривань

Програма обробки апаратних переривань, перерахованих у табл. 1, перед закінченням своєї роботи повинна посилати код 20h у порт 20h програмувального контролера переривань Intel 8259, попередньо заборонивши всі переривання:

disable();
outportb(0x20, 0x20);

Якщо оброблювач апаратного переривання не закінчується цими операторами, то мікросхема 8259 не очистить інформацію регістра обслуговування для того, щоб була дозволена обробка переривань з більш низькими рівнями, чим тільки що оброблене. Ці оператори не потрібні якщо власний оброблювач доповнює системний.

5. Створення резидентного оброблювача переривання

Програма, що знаходиться у пам’яті ПК у межах всього сеансу роботи, називається резидентною. Для того, щоб залишити програму у пам’яті, використовується функція

keep(статус, розмір)

Параметр **"розмір"** задає число параграфів оперативної пам'яті, що залишаються програмі. Функція **keep()** передає код завершення процесу через параметр **"статус".**
Розглянемо як приклад програму, що для переривання 71h встановлює резидентний оброблювач mybeep() :

```
/*     Резидентний оброблювач ПРОГРАМНОГО переривання 0x71
#include<dos.h>
 void interrupt mybeep()
{ int register i;
    for(i=0;i<=9900;i++) sound(1000 - i/20);
    nosound();
}

 void install(void interrupt (*faddr)(),  int inum)
  {  setvect(inum, faddr);
  }
  void resident(void)
   {  int status;
    keep(status,0x2000);
   }
 main()
  { install(mybeep, 0x71);
    resident();      /* Залишає в пам'яті ВСЮ ПРОГРАМУ  main */
```

```
}
```

У даній програмі зміна вектора переривання здійснюється функцією install(), а залишення програми резидентною-функцією resident(), що у свою чергу викликає функцію keep() за допомогою оператора

```
keep(status, 0x2000);
```

При цьому другий параметр 0x2000 задає розмір пам'яті (у параграфах), що залишається програмі. Це число, що відповідає 128Кб, обрано з запасом як максимальний можливий розмір програми на С при роботі з малою (small) моделлю пам'яті. При роботі з маленькою (tiny) моделлю пам'яті можна задавати 0x1000, що відповідає 64К байтів. Тип моделі пам'яті встановлюється шляхом вибору в головному меню С пункту Options, а потім підпунктів Compiler і Model. У реальних задачах на відміну від даного навчального приклада розмір пам'яті, що залишається програмі, повинний ретельно розраховуватися і, імовірно, буде значно менше.

Для виклику резидентного оброблювача програмного переривання, установленого за допомогою приведеної програми, варто запустити іншу програму з функцією geninterrupt().

Наприклад:

```
#include<dos.h>
main()
{
 geninterrupt(0x71);
}
```

При цьому буде генеруватися звуковий сигнал з перемінною частотою, що змінюється в діапазоні від 1000 Гц до 505 Гц.

6. Завдання

Розробити і виконати на ПЕОМ програму, у якій:

- встановити заданий оброблювач переривання, функція і режим виклику якого визначено в табл. 2.
- перед викликом оброблювача переривання видати об’єм вільної основної пам'яті в кілобайтах і вміст основних регістрів мікропроцесора.
- викликати переривання.

Таблиця 2

Варіант	Переривання	Пристикування і режим виклику оброблювача	Функція оброблювача переривання
1	Апаратне <PrtSc>	Заміна, нерезидентний	Генерування звуку 1000 Гц і вивід часу дня в тиках
2	Програмне 80h	Заміна, резидентний	Генерування звуку 1160 Гц
3	Апаратне від таймера	Доповнення, нерезидентний	Генерування звуку 2000 Гц кожні 4сек
4	Програмне 73h	Заміна, резидентний	Генерування звуку 1200 Гц

5	Програмне Ctrl-Break	Заміна, нерезидентний	Генерування звуку 1600 Гц і вивід поточної дати
6	Апаратне Ctrl-Break	Заміна, нерезидентний	Генерування звуку 1900 Гц і вивід поточного часу
7	Програмне 75h	Заміна, резидентний	Вивід поточної дати
8	Програмне Ctrl-Break	Заміна, нерезидентний	Вивід поточних дати і часу з виходом у MS DOS
9	Програмне 76h	Заміна, резидентний	Вивід часу дня в сек.
10	Програмне 74h	Заміна, нерезидентний	Вивід часу дня в "тиках"
11	Програмне Ctrl-Break	Заміна, резидентний	Генерування звуку 1800 Гц
12	Апаратне Ctrl-Break	Доповнення, нерезидентний	Генерування звуку 1700 Гц і вивід поточної дати
13	Програмне Ctrl-Break	Заміна, нерезидентний	Вивід поточних дати і часу
14	Апаратне від таймера	Заміна, нерезидентний	Генерування звуку від 700 до 1200 Гц кожні 3 сек.
15	Програмне Ctrl-Break	Заміна, нерезидентний	Вивід поточних дати і часу і генерування звуку 900 Гц
16	Апаратне від таймера	Доповнення, нерезидентний	Генерування звуку від 2000 до 1100 Гц кожні 4 сек.
17	Програмне 77h	Заміна, нерезидентний	Генерування звуку від 1100 до 500 Гц
18	Програмне Ctrl-Break	Заміна, резидентний	Генерування звуку від 400 до 800 Гц
19	Програмне Ctrl-Break	Заміна, резидентний	Генерування звуку від 800 до 400 Гц
20	Апаратне Ctrl-Break	Заміна, нерезидентний	Генерування звуку від 1300 до 1900 Гц
21	Апаратне Ctrl-Break	Заміна, нерезидентний	Генерування звуку від 1000 до 1800 Гц
22	Програмне	Заміна,	Вивід. поточних дати і часу і

	79h	нерезидентний	генерування звуку 750 Гц
23	Апаратне <PrtSc>	Заміна, нерезидентний	Генерування звуку від 500 до 1000 Гц
24	Програмне Ctrl-Break	Заміна, нерезидентний	Генерування звуку 500 Гц і вивід поточних дати і часу
25	Програмне 78h	Заміна, нерезидентний	Вивід часу дня в сек. і генерування звуку 766 Гц

7. Зміст звіту

1. Умова завдання.
2. Текст програми.
3. Висновки і результати програми .

8.Контрольні питання.

1. Що заноситься в стек при переході на програму обробки переривань ?
2. Що таке вектор переривання, його довжина, таблиця векторів переривання, її розташування в пам'яті ?
3. Функції для роботи з програмними перериваннями, якщо не потрібна передача параметрів через сегментні регістри мікропроцесора
4. Функції для роботи з програмними перериваннями, якщо потрібна передача параметрів через сегментні регістри мікропроцесора.
5. У чому розходження між функціями int 86 і intdos ?
6. У чому розходження між функціями intdos і intdosx ?
7. Особливості використання функції bdos.
8. Відмінність між функціями bdos і bdosptr ?
9. Керування апаратними перериваннями (пріоритети).
10. Які функції дозволяють/забороняють апаратні переривання ?
11. Як повинна бути описана програма обробки переривання ?
12. Як працює функція установки нового вектора переривання ?
13. Використання geninterrupt для створення оброблювача програмного переривання.
14. Що відбувається при одночасному натисканні клавіші CTRL+BREAK ?

ЛАБОРАТОРНА РОБОТА №2

ВИЗНАЧЕННЯ СИСТЕМНИХ РЕСУРСІВ КОМП'ЮТЕРА

Мета роботи – дослідження системних ресурсів ПК.

1. Визначення типу IBM PC і типу мікропроцесора.

Програма може визначити на якому типі IBM PC вона завантажена, прочитавши вміст байта в ПЗП BIOS за адресою F000:FFFE. Деякі типи комп'ютерів, частково сумісні з IBM PC, мають нуль у цьому байті чи інші коди. Більш повну інформацію про тип комп'ютера можна одержати з функції C0h переривання 15h, яка через пару регістрів ES:BX повертає покажчик на початок таблиці системного середовища.

Приклад. Визначити тип IBM PC, код підмоделі і рівень ревізії BIOS.

```
#include<stdio.h>
#include<dos.h>
void main(void)
 { union REGS r;
   struct SREGS s;
   unsigned char tip;
   r.h.ah = 0xC0;
   int86x(0x15, &r, &r, &s);
   tip = peekb(s.es, r.x.bx + 2);
   printf("МОДЕЛЬ  КОМП'ЮТЕРА:\n");
   switch(tip)
     { case 0xFF: printf("Оригінальний IBM PC\n"); break;
       case 0xFE: printf("IBM PC XT чи портативний PC\n"); break;
       case 0xFD: printf("IBM PC jr\n"); break;
       case 0xFC: printf("IBM PC AT (чи XT моделі 286"
                         " чи PS/2 моделі 50/60)\n"); break;
       case 0xFB: printf("IBM PC XT з 640K пам'яті на "
                        "материнській платі\n"); break;
       case 0xFA: printf("PS/2 моделі 30\n"); break;
       case 0x9: printf("Convertible PC\n"); break;
       case 0x8: printf("PS/2 моделі 80\n"); break;
       default:   printf("Невідомий тип IBM PC\n");
      }
   printf("\n КОД  ПІДМОДЕЛІ %X\n", peekb(s.es, r.x.bx + 3));
   printf("\n РІВЕНЬ РЕВІЗІЇ BIOS %X\n", peekb(s.es, r.x.bx + 4));
 }
```

1.1. Визначення типу мікропроцесора

Комп'ютери сімейства IBM PC звичайно використовують як центральний процесор мікропроцесори Intel та AMD. Стандартні для всіх мікросхем оцінки наступні.

Технологічний параметр (товщина пластини)- це товщина пластини, на якій виконується напилювання процесора. Виміряється в мікрометрах, чи мікронах [мкм].

Споживаний струм – це струм, що споживає процесор при максимальному завантаженні. Якщо помножити струм на напругу процесора, можна одержати споживану процесором потужність. Струм виміряється в амперах [А].

Напруга ядра - напруга, що подається на процесор. Визначається при виготовленні. На відміну від струму, подану на процесор напругу можна змінити, що приведе до зміни споживаної ним потужності. Вимірюється у вольтах [В].

Частота процесора - кількість циклів, виконуваних в одну секунду. У залежності від конструкції процесора, може бути змінено користувачем. Вимірюється в мегагерцах [Мгц].

Площа ядра - площа процесорного ядра. Вимірюється в квадратних міліметрах [мм^2].

Ступінь інтеграції - кількість інтегрованих елементів (транзисторів) на одиницю площі. Виміряються в мільйонах транзисторів на квадратний міліметр [млн./мм^2].

Тепловиділення - показує потужність, що виділяється у вигляді тепла . Прямо залежить від споживаної потужності і товщини пластини. Це ключовий параметр, що визначає температуру, до якої нагрівається процесор. Вимірюється у ватах [Вт].

Прикладні програми можуть одержувати інформацію про процесор по інструкції **CPUID** при будь-якому рівні привілеїв, а привілейовані програми (PprivilegeLevel=0) можуть користатися і даними регістру CR4. Крім **CPUID**, для визначення модифікації процесора використовуються також і два ідентифікатори: **S-Spec** і **маска** процесора ("mask"). У документації Intel замість терміна "mask" використовується термін "core stepping" - ревізія ядра, що краще відбиває суть цього ідентифікатора. **Маска** (ревізія ядра) процесора позначає версію ядра процесора. Розходження в масці між двома процесорами говорить про те, що в ядрах цих процесорів "прошитий" різний мікрокод. **S-Spec** - ідентифікатор, що однозначно визначає функціональні, електричні, технологічні характеристики і тип конкретного екземпляра процесора. **S-Spec** – самий важливий із трьох розглянутих ідентифікаторів: знаючи його, можна довідатися маску, робочу частоту ядра процесора, розмір L2-кеша й інші характеристики процесора. Значення **S-Spec** зазначене у маркіруванні на картриджі чи корпусі процесора. Однак **S-Spec** ніде не "прошивається" у ядрі процесора, що дає можливість для перемаркірування процесорів.

Задача ідентифікації вимагає визначення покоління процесора. Для цього у 32-розрядних процесорів аналізується регістр прапорів EFLAG;

біт 18 доступний, починаючи з процесорів 486;

біти 19 (VIF) і **20** (VIP) у процесорів, що не підтримують розширення віртуального режиму (VME) завжди нульові;

біт 21 (ID) визначає можливість використання інструкції CPUID. Ознакою приступності інструкції є можливість програмної зміни значення цього біта.

Інструкція **CPUID**, доступна, починаючи з Pentium і деяких моделей 486, викликається з параметром, зазначеним у регістрі EAX.

CPUID(0) - у регістрі **EAX** повертається максимально припустиме значення параметра виклику; у регістрах **EBX, EDX і ECX** процесор повертає символьний рядок, специфічний для виробника. Символи рядка розміщені в регістрах у зазначеному порядку, починаючи з молодших байт.

Процесори Intel повертають рядок "GenuineIntel":

EBX=756E6547h - "Genu", символ "G" у регістрі BL,
EDX=49656E69h - "ine", символ "i" у регістрі DL,
ECX=6C65746Eh - "ntel", символ "n" у регістрі CL.

Процесори AMD повертають рядок "AuthenticAMD":

EBX=68747541h
ECX=444D4163h
EDX=69746E65h

CPUID(1) - у молодшому слові регістра **EAX** процесор повертає код ідентифікації (див. таблицю 1), який є сигнатура процесора і старший елемент 96-бітного серійного номера. Це ж значення міститься в регістрі **DX** після апаратного скиду:

EAX[3:0] - степінг;
EAX[7:4] - модель;
EAX[11:8] - сімейство;
EAX[13:12] - тип;
EAX[31:14] - зарезервовано (0);

Регістри **EBX=0, ECX=0** (резерв).

Регістр **EDX** містить список наявних розширень базової архітектури і відображає регістр властивостей (Feature Flags register). Призначення біт регістра приведено у таблиці 2.

Таблиця 2

Біт	Назва	Призначення
0	FPU	Floating Poin Unit - наявність математичного співпроцесора
1	VME	Virtual-8086 Mode Enhancements - розширення режиму V86 (віртуалізація прапора переривань)
2	DE	Debugging Extensions - розширення налагодження (можливість зупинки по звертанню до портів)
3	PSE	Page Size Extension - можливість застосування розміру сторінки в 4 Мбайт
4	TSC	Time Stamp Counter - наявність лічильника реального часу
5	MSR	Model Specific Register - підтримка модельно-специфічних регістрів у стилі Pentium (інструкції RDMSR, WRMSR)
6	PAE	Physical Address Extension - можливість розширення фізичної адреси до 36 біт
7	MCE	Machine Check Exception - підтримка виключення машинного контролю #MC
8	CX8	Підтримка інструкції CMPXCHG8B
9	APIC	Наявність убудованого програмно-доступного контролера переривань APIC
10	-	Зарезервовано
11	SEP	SYSENTER Present - підтримка інструкцій швидких системних викликів SYSENTER і SYSEXIT
12	MTRR	Memory Type Range Registers - наявність регістра керування кашируванням MTRRcap
13	PGE	Page Global Enable - підтримка біт глобальності в елементах каталогу і таблиць сторінок, а також біта PGE у регістрі CR4
14	MCA	Machine Check Architecture - підтримка архітектури машинного контролю
15	CMOV	Conditional Move - підтримка інструкцій умовного пересилання CMOVcc, а якщо є FPU, то і інструкцій FCMOVCC і FCOMI
16	PAT	Page Attribute Table - підтримка таблиць атрибутів сторінок (PAT)
17	PSE-36	36-bit Page Size Extension - можливість використання 36-бітної фізичної адресації для сторінок у 4 Мб
18	PN	Processor Number - підтримка повідомлення 96-бітного серійного номера по інструкції CPUID(3)
22	-	Зарезервовано
23	MMX	Підтримка MMX
24	FXSR	Fast floating point save and restore - підтримка інструкцій швидкого збереження і відновлення контексту FPU (інструкцій FXSAVE і FXRSTOR). Вказує і на приступність індикатора використання цих інструкцій операційною системою (CR4.OSFXSR)
25	XMM	Наявність блоку XMM (підтримка нових інструкцій розширення SSE)
24-31		Зарезервовано -- // --

CPUID(2) - у регістрах **EAX, EBX, ECX, EDX** повертаються параметри конфігурації процесора. **Молодші 8 біт EAX** повідомляють, скільки разів потрібно підряд викликати інструкцію (з EAX=2) для одержання повної інформації про процесор. Інші байти регістра **EAX** і інших регістрів містять дескриптори окремих вузлів, що розшифровуються по спеціальних таблицях. Ознакою використання кожного з регістрів **EAX, EBX, ECX, EDX** є нуль у його біті 31. **Виклик CPUID(2)** з'явився з процесорами 6-го покоління. Призначення прапорів розширення архітектури представлено в табл. 3.

Таблиця 3

00h	Нульовий дескриптор (у не використовуваних байтах)
01h	TLB інструкцій: сторінки 4ДО, 4WSA, 32 входження
02h	TLB інструкцій: сторінки 4M, FA, 2 входження
03h	TLB даних: сторінки 4ДО, 4WSA, 64 входження
04h	TLB даних: сторінки 4M, 4WSA, 8 входжень
06h	Кеш інструкцій (LI): 8ДО, 4WSA, розмір рядка 32 байта
08h	Кеш інструкцій (LI): 16ДО, 4WSA, розмір рядка 32 байта
0Ah	Кеш даних (LI): 8ДО, 2WSA, розмір рядка 32 байта
0Ch	Кеш даних (LI): 16ДО, 2WSA, розмір рядка 32 байта
40h	Немає вторинного кеша
41h	Вторинний кеш 128ДО, 4WSA, розмір рядка 32 байта
42h	Вторинний кеш 256ДО, 4WSA, розмір рядка 32 байта
43h	Вторинний кеш 512ДО, 4WSA, розмір рядка 32 байта
44h	Вторинний кеш 1M, 4WSA, розмір рядка 32 байта
45h	Вторинний кеш 2M, 4WSA, розмір рядка 32 байта

CPUID(3) - одержання молодших 64 біт серійного номера процесора (Intel Processor Serial Number), доступно починаючи з Pentium III (сімейство 6, модель 7 і старше). **ED**X - середні 32 біта ідентифікатора; **ECX** - молодші 32 біта ідентифікатора. Повний ідентифікатор має довжину 96 біт. Старші 32 біта - код ідентифікації процесора, що повертається у EAX по CPUID(1). Можливість виклику визначається по біту PN регістра властивостей (після CPUID(1) біт EDX.18=1). Після апаратного скиду процесорів, що підтримують повідомлення ідентифікатора, цей виклик дозволений. Заборонити повідомлення ідентифікатора до наступного апаратного скидання можна установкою в одиницю біта 21 регістра.

Фірма AMD розширила виклики CPUID. Для перевірки наявності розширень викликається **CPUID з EAX=8000_0000h**. При наявності розширень у EAX результатом буде число, більше 8000_0000h, рівне максимальному параметру розширеного виклику. Викликом **EAX=8000_0001h** можна визначити специфічні розширення архітектури від AMD. Наприклад, підтримка 3DNow визначається по установленому біту 31 регістра EDX. Приклад програми:

```
/*Compile by BC++version 3.1 */
#include <stdio.h>
#include <string.h>
#include <conio.h>

void main() {
  char VendorSign[13];   /* for to store our vendorstring*/
  unsigned long MaxEAX;/*This will be used to store the maximum
EAX*/
```

```
char* Comp1[32]; /*This is the array that will hold the short
names for our features bitmap.*/
  unsigned long REGEAX, REGEBX, REGECX, REGEDX;
  int dFamily, dModel, dStepping, dFamilyEx, dModelEx;
  char dType[10];
  int dComp1Supported[32];
  int dBrand, dCacheLineSize, dLogicalProcessorCount,
dLocalAPICID;
  asm {
    XOR EAX, EAX
    /*An efficient alternatvie to MOV EAX, 0x0*/
    db 0fh,0a2h
    /*=cpuID,This instruction will load our registers.*/
    MOV dword ptr [VendorSign], EBX
    /*Copy the first 4 bytes in the VendorString from EBX.*/
    MOV dword ptr [VendorSign+4], EDX
    /*Copy the next 4 bytes.*/
    MOV  dword ptr [VendorSign+8],   ECX
    /*Copy the next 4 bytes.*/
    MOV  dword ptr MaxEAX,           EAX
    /*EAX contains the maximum value to call CPUID with. Copy
     it to the MaxEAX variable.*/
  }
  VendorSign[12]=0;
//The last character in the VendorSign can be anything.
//To make sure that it stops at the last character we add
//a zero character at the end
  printf("Vendor string: %s\n", VendorSign);
  printf("Maximum EAX value: %i\n", MaxEAX);
  if(strcmp(VendorSign, "GenuineIntel")==0) {
    Comp1[0]="FPU";   /*Floating Point Unit*/
    Comp1[1]="VME";   /*Virtual Mode Extension*/
    Comp1[2]="DE";    /*Debugging Extension*/
    Comp1[3]="PSE";   /*Page Size Extension*/
    Comp1[4]="TSC";   /*Time Stamp Counter*/
    Comp1[5]="MSR";   /*Model Specific Registers*/
    Comp1[6]="PAE";   /*Physical Address Extesnion*/
    Comp1[7]="MCE";   /*Machine Check Extension*/
    Comp1[8]="CX8";   /*CMPXCHG8 Instruction*/
    Comp1[9]="APIC";  /*On-chip APIC Hardware*/
    Comp1[10]="";     /*Reserved*/
    Comp1[11]="SEP";  /*SYSENTER SYSEXIT*/
    Comp1[12]="MTRR"; /*Machine Type Range Registers*/
    Comp1[13]="PGE";  /*Global Paging Extension*/
    Comp1[14]="MCA";  /*Machine Check Architecture*/
    Comp1[15]="CMOV"; /*Conditional Move Instrction*/
    Comp1[16]="PAT";  /*Page Attribute Table*/
    Comp1[17]="PSE-36"; /*36-bit Page Size Extension*/
    Comp1[18]="PSN";  /*96-bit Processor Serial Number*/
    Comp1[19]="CLFSH"; /*CLFLUSH Instruction*/
    Comp1[20]="";     /*Reserved*/
    Comp1[21]="DS";   /*Debug Trace Store*/
    Comp1[22]="ACPI"; /*ACPI Support*/
```

```
  Comp1[23]="MMX";  /*MMX Technology*/
  Comp1[24]="FXSR"; /*FXSAVE FXRSTOR (Fast save and restore) */
  Comp1[25]="SSE";  /*Streaming SIMD Extensions*/
  Comp1[26]="SSE2"; /*Streaming SIMD Extensions 2*/
  Comp1[27]="SS";   /*Self-Snoop*/
  Comp1[28]="HTT";  /*Hyper-Threading Technology*/
  Comp1[29]="TM";   /*Thermal Monitor Supported*/
  Comp1[30]="IA-64"; /*IA-64 capable*/
  Comp1[31]="";     /*Reserved*/
}
else if(strcmp(VendorSign, "AuthenticAMD")==0) {
  Comp1[0]="FPU";   /*Floating Point Unit*/
  Comp1[1]="VME";   /*Virtual Mode Extension*/
  Comp1[2]="DE";    /*Debugging Extension*/
  Comp1[3]="PSE";   /*Page Size Extension*/
  Comp1[4]="TSC";   /*Time Stamp Counter*/
  Comp1[5]="MSR";   /*Model Specific Registers*/
  Comp1[6]="PAE";   /*Physical Address Extesnion*/
  Comp1[7]="MCE";   /*Machine Check Extension*/
  Comp1[8]="CX8";   /*CMPXCHG8 Instruction*/
  Comp1[9]="APIC";  /*On-chip APIC Hardware*/
  Comp1[10]="";     /*Reserved*/
  Comp1[11]="SEP";  /*SYSENTER SYSEXIT*/
  Comp1[12]="MTRR"; /*Machine Type Range Registers*/
  Comp1[13]="PGE";  /*Global Paging Extension*/
  Comp1[14]="MCA";  /*Machine Check Architecture*/
  Comp1[15]="CMOV"; /*Conditional Move Instrction*/
  Comp1[16]="PAT";  /*Page Attribute Table*/
  Comp1[17]="PSE-36"; /*36-bit Page Size Extension*/
  Comp1[18]="";     /*?*/
  Comp1[19]="MPC";  /*MultiProcessing Capable*/
  Comp1[20]="";     /*Reserved*/
  Comp1[21]="";     /*?*/
  Comp1[22]="MIE";  /*AMD Multimedia Instruction Extensions*/
  Comp1[23]="MMX";  /*MMX Technology*/
  Comp1[24]="FXSR"; /*FXSAVE FXRSTOR (Fast save and restore) */
  Comp1[25]="SSE";  /*Streaming SIMD Extensions*/
  Comp1[26]="";     /*?*/
  Comp1[27]="";     /*?*/
  Comp1[28]="";     /*?*/
  Comp1[29]="";     /*?*/
  Comp1[30]="3DNowExt"; /*3DNow Instruction Extensions*/
  Comp1[31]="3DNow"; /*3DNow Instructions */
}
if(MaxEAX>=1) {
  asm {
    MOV EAX,1
    db 0fh,0a2h
    MOV [REGEAX],EAX
    MOV [REGEBX],EBX
    MOV [REGECX],ECX
    MOV [REGEDX],EDX
  }
```

```
    dFamily=((REGEAX>>8)&0xF);
    dModel=((REGEAX>>4)&0xF);
    dStepping=(REGEAX&0xF);
    dFamilyEx=((REGEAX>>20)&0xFF);
    dModelEx=((REGEAX>>16)&0xF);
    switch(((REGEAX>>12)&0x7)) {
      case 0:
        strcpy(dType, "Original");
        break;
      case 1:
        strcpy(dType, "OverDrive");
        break;
      case 2:
        strcpy(dType, "Dual");
        break;
    }

    for(unsigned long C=1, Q=0;Q<32;C*=2, Q++) {
      dComp1Supported[Q]=(REGEDX&C)!=0?1:0;
    }


    dBrand=REGEBX&0xFF;
    dCacheLineSize=((strcmp(Comp1[19], "CLFSH")==0)
     &&(dComp1Supported[19]==1))?((REGEBX>>8)&0xFF)*8:-1;
    dLogicalProcessorCount=((strcmp(Comp1[28], "HTT")==0)
     &&(dComp1Supported[28]==1))?((REGEBX>>16)&0xFF):-1;
    dLocalAPICID=((REGEBX>>24)&0xFF); //This works on P4 or later
  }
  printf("%s\n", dType);
  printf("Family %i, Model %i, Stepping %i\n", dFamily, dModel,
dStepping);
  printf("Extended Family %i, Extended Model %i\n", dFamilyEx,
dModelEx);
  printf("Supported flags: ");
  for(unsigned long Q=0;Q<27;Q++)
  {
    if(dComp1Supported[Q])
     {
      printf("%s ", Comp1[Q]);
     }
  }
  printf("\n");
  printf("CacheLineSize: %i\n", dCacheLineSize);
  printf("Logical processor count: %i\n", dLogicalProcessorCount);
  printf("Local APIC ID: %i\n", dLocalAPICID);
}
```

1.2 Визначення частоти процесора

1.2.1. Визначення частоти CPU методом порівняння циклів

Для програмного визначення частоти CPU можна врахувати, що час роботи команд визначається або в тіках - адреса **0x40:0x63**, або в сотих частках секунди з переривання **21h,** функції **2Ch**. У цьому випадку алгоритм складається з етапів:

- емпірично визначити кількість циклів тесту - до 10**7;
- знайти час роботи еталонного порожнього циклу, як різницю часів початку і кінця циклу;
- визначити час роботи циклу, у тілі якого знаходиться одна асемблерна команда, наприклад dec;
- знайти чистий час роботи тестуємої команди, як різницю між результатами попередніх двох пунктів;
- знайти частоту процесора як величину зворотну часу виконання однотактної команди в секундах.

1.2.2. Визначення частоти CPU методом завдання інтервалу часу.

Інтервал часу в мікросекундах формується годинником реального часу - від переривання **int 70h**. У програмі користувача інтервал задається перериванням **int 15h** у двох видах:

- функцією 86h int 15h, після чого програма користувача переходить у режим чекання на зазначений час;
- функцією 83h int 15h - тут керування передається наступному оператору програми, а кінець заданого інтервалу часу відзначається установкою в одиницю байта з адресою 0:04A0.

Для реалізації алгоритму в регістрах CX:DX задається інтервал, наприклад, 60000мкс і викликається переривання 0x15, при _AX=0x8300. Далі організується цикл до кінця заданого інтервалу часу. У тілі циклу, крім команд перевірки на його завершення, повинний бути лічильник циклів і біля сотні асемблерних команд dec dx, заданих повторювачем DUP. Визначається час роботи однотактної команди, як частка від розподілу заданого інтервалу часу на кількість тактів у циклі. Частота процесора - це величина зворотна часу виконання однотактної команди в секундах.

Приклад.

```
{union REGS rr;
 unsigned int i;
 unsigned long max,c;
 float fq;
printf("Уведіть час визначення частоти процесора***\n");
printf("у секундах (рекомендується в межах 1..4294)\n");
printf("при завданні більш тривалого часу\n");
printf("підвищується точність визначення частоти.\n");
scanf("%u",&i);
printf("визначення частоти. Будь ласка, почекайте...\n");
max=i*1000000.0;
rr.x.ax=0x8300;
rr.x.cx=ceil(max/65536);
rr.x.dx=fmod(max,65536);
int86(0x15,&rr,&rr);
_ES=0;
_SI=0x4a0;
asm mov ecx,0
mm1:    asm{
```

```
        inc ecx
        db 120 DUP(4Ah)
        cmp al,byte ptr es:[si]
        jne mm1
        mov dword ptr c,ecx
    };
fq=max;
fq/=c;
fq/=127.0;
fq=1.0/fq;
printf(" Частота процесора %8.5f Мгц\n",fq);
}
```

2. Визначення характеристик BIOS

Basic Input/Output System – це основна система вводу/виводу, розміщена в ПЗП (звідси назва ROM BIOS). Вона являє собою набір програм перевірки й обслуговування апаратури комп'ютера, і виконує роль посередника між операційною системою і апаратурою. BIOS одержує керування при включенні і скиданні системної плати, тестує саму плату й основні блоки комп'ютера - відеоадаптер, клавіатуру, контролери дисків і портів вводу/виводу, набудовує chipset плати і завантажує зовнішню операційну систему.

Звичайно на системній платі встановлено тільки ПЗП із системним (Main, System) BIOS, що відповідає за саму плату і контролери FDD, HDD, портів і клавіатури. У системний BIOS практично завжди включений System Setup - програма настроювання системи. Відеоадаптери і контролери HDD мають власні BIOS в окремих ПЗП; їх також можуть мати й інші плати - інтелектуальні контролери дисків і портів, мережні карти і т.п. BIOS для сучасних системних плат розробляється однією з фірм, що спеціалізуються на цьому. Іноді для однієї і тієї ж плати маються версії BIOS від різних виробників - у цьому випадку допускається копіювати прошивання чи заміняти мікросхеми ПЗП; у загальному ж випадку кожна версія BIOS прив'язана до конкретної моделі плати.

Інформація, що характеризує BIOS зберігається у виді текстових рядків по відповідним адресам у ROM BIOS. Основні дані починаються з таких базових адрес:

Виробник 0x000:0x091.
Версія 0x000:0x061.
Дата видання 0x000:0xFFF5.
Дані відеосистеми:
Тип VideoBIOS 0x000:0x001E.
Тип відеокарти і версія VideoBIOS 0x000:0x004B.

Крім того, характеристики інтерфейсу:

Типи шини даних 0x000:0xFFD9.
Тип чіпсету 0x000:0xEC71.

Приклад. Визначити основні характеристики BIOS

```
. . . . . .
char far *ptr  = MK_FP(0xF000, 0xE091);
 char far *ptr1 = MK_FP(0xF000, 0xE061);
 char far *ptr2 = MK_FP(0xF000, 0xFFF5);
 char far *ptr3 = MK_FP(0xC000, 0x001E);
 char far *ptr4 = MK_FP(0xC000, 0x004B);
 char far *ptr5 = MK_FP(0xF000, 0xFFD9);
 char far *ptr6 = MK_FP(0xF000, 0xEC71);
```

```
        ...
   printf("Виробник: ");
   for(; *ptr == 0; ptr++) ;
   while(*ptr) printf("%c", *ptr++);
   printf("\n");
   printf("Версія: ");
   for(; *ptr1 == 0; ptr1++) ;
   while(*ptr1) printf("%c", *ptr1++);
   printf("\n");
   printf("Дата видання: ");
   for(; *ptr2 == 0; ptr2++) ;
   while(*ptr2) printf("%c", *ptr2++);
   printf("\n");
   printf("Тип VideoBIOS: ");
   for(; *ptr3 == 0; ptr3++) ;
   while(*ptr3) printf("%c", *ptr3++);
   printf("\n");
   printf("Тип відеокарти і версія VideoBIOS: ");
   for(; *ptr4 == 0; ptr4++) ;
   while(*ptr4) printf("%c", *ptr4++);
   printf("\n");
   printf("Тип шини даних: ");
   for(; *ptr5 == 0; ptr5++) ;
   while(*ptr5) printf("%c", *ptr5++);
   printf("\n");
   printf("Тип чіпсета: ");
   for(; *ptr6 == 0; ptr6++) ;
   while(*ptr6) printf("%c", *ptr6++);
   printf("\n");
}
```

3. Визначення обсягу основної пам'яті

Для визначення розміру оперативної пам'яті можна використовувати переривання **12h BIOS.** Це переривання повертає в регістр AX мікропроцесора кількість кілобайт звичайної пам'яті в системі (без обліку розширеної і додаткової відображуваної пам'яті). Переривання **12h** використовується функцією C biosmemory(), яка повертає обсяг ОЗП в кілобайтах. Прототип функції знаходиться у файлі bios.h.
Приклад. Визначити обсяг ОЗП IBM PC

```
#include<stdio.h>
#include<bios.h>
int main(void)
 {
  printf("Обсяг звичайної пам'яті %d Кбайт\n", biosmemory());
  return 0;
 }
```

4. Визначення числа і типу периферійних пристроїв

При включенні ПЕОМ BIOS перевіряє приєднане устаткування і повідомляє про результати в двохбайтну змінну з адресою 0x0040:0x0010. Переривання **11h BIOS** повертає в

регістр AX значення цієї змінної. Значення інтерпретується як набір бітових полів, що характеризують устаткування:

біт 0 якщо "1", то присутній накопичувач на гнучкому магнітному диску (НГМД);

біт 1 якщо "1", то є співпроцесор;

біти 2-3 розмір базової пам'яті на системній платі (11 - 64 К и більш);

біти 4-5 активний відеоадаптер:

00 - не використовується,
01 - кольоровий (CGA) 40 символів * 25 рядків,
10 - кольоровий (CGA) 80 символів * 25 рядків,
11 - монохромний 80 символів * 25 рядків;

біти 6-7 кількість НГМД (тільки, якщо біт 0 дорівнює "1"):

00 - один,
01 - два,
10 - три,
11 - чотири;

біт 8 використовується тільки для IBM PCjr: якщо "0", то є контролер прямого доступу до пам'яті (ПДП);

біти 9-11 кількість послідовних портів RS232;

біт 12 якщо "1", те є присутнім ігровий адаптер;

біт 13 використовується тільки для IBM PCjr: якщо "1", то приєднано послідовний принтер;

біти 14-15 кількість портів для принтерів.

Приклад .Перевірити наявності устаткування (переривання BIOS 11h - функція biosequip() в мові С)

```
#include<stdio.h>
#include<bios.h>
#include<dos.h>
int main(void)
{
 union
   {
     int status;
     struct
       {
         unsigned drives_present:1;
         unsigned coprocessor   :1;
         unsigned unused1       :4;
         unsigned num_drives    :2;
         unsigned unused2       :1;
         unsigned RS232_ports   :3;
         unsigned unused3       :2;
         unsigned num_printers  :2;
       }fields;
    }ob;
    ob.status = biosequip();
    if (!ob.fields.drives_present) printf("Немає НГМД\n");
     else printf("Усього %u НГМД\n",ob.fields.num_drives + 1);
    if (ob.fields.coprocessor)
        printf("Є співпроцесор із ПЗ\n");
    else printf("Немає співпроцесора з ПЗ\n");
```

```
    printf("Кількість послідовних портів RS232 %u\n",
            ob.fields.RS232_ports);
    printf ("Кількість портів для принтерів  %u\n",
             ob.fields.num_printers);
    return 0;
 }
```

Приклад. Визначити адреси портів (перший варіант)

```
{void far *p;
 long com;
 int flag;
 p=MK_FP(0x40,0);
 com=*(long far *)p;
 printf("\n\nAdress COM1 : %ph",com);
 if(flag<2)return;
 p=MK_FP(0x40,2);
 com=*(long far *)p;
 printf("\nAdress COM2 : %ph",com);

}
```

Приклад . Визначити адреси портів (другий варіант)

```
printf("* Послідовні порти :\n");
int86(0x11, &regs, &regs);
 regs.h.ah=regs.h.ah>>1;
 regs.h.ah&=7;
printf("  - Кількість :  %u\n", regs.h.ah);
int k=regs.h.ah;

printf("  - Адреси:\n    N        Адреса\n");
int addr=0x400;
for (int i=0;i<k;i++)
 {
  unsigned long far *comadr= (unsigned long far* )MK_FP(0,addr);
  printf ("    %d         %x\n",i,*comadr);
  addr+=2;
 }
```

5. Визначення типу магнітних накопичувачів

Для визначення виду і конфігурації магнітних накопичувачів існує більш точний спосіб за допомогою функцій переривання **21h** (компілятор Borland):

- функція **4408h**, повертає регістр AL=0, якщо це FD Drv;
- функція **4409h**, повертає в регістрі DX атрибути накопичувача;
  біт 15 дорівнює 1, якщо це Substituted Drv,
  біти 12 і **9** рівні 1, якщо це Network Drv;
- функція **440Dh,** під функція 0660h визначає тип накопичувача;

Число і тип FDD і HDD можна довідатися з байтів 10h, 12h енергонезалежної пам'яті. Розмір дисків визначає функція **08h**, переривання **13h** - у регістрі DH вона повертає кількість голівок, у регістрі CH повертає молодші цифри кількості циліндрів, у перших 2 бітах регістра CL - старші цифри кількість циліндрів, останній 6 біт регістра CL – кількість секторів на доріжці.

5.1. Визначення числа і розмірів дисків

Наявність HDD у комп'ютері визначається шляхом посилки в порт **70h** коду байту - визначника **12h.** Ненульовий результат приймається з порту **71h**. Функція **8** переривання **13h** повертає в регістри мікропроцесора характеристики HDD (це максимальні номери, починаючи з 0):

CH - молодша частина max номера циліндра;

CL - у перших двох бітах - старша частина max номера циліндрів, а в інших бітах - max номер сектора на доріжці;

DH - max номер голівки (сторони).

Добуток цих величин (з врахуванням того, що сектор=512 байт) дає розмір HDD у байтах. Наявність FDD у комп'ютері визначається шляхом посилки в порт **70h** коду байту-визначника **10h.** Ненульовий результат приймається з порту **71h** і визначає дисководи:

1- 360 Кб; 2- 1.2 Мб; 0x30- 720 Кб; 0x40- 1.44 Мб;

Якщо дисководів два:

0x42- 1.2Мб і 1.44Мб; 0x24- 1.44Мб і 1.2Мб;

0x34- 1.44Мб і 720Кб; 0x44- 1.44Мб і 1.44Мб і т.д.

6. Визначення типу клавіатури

При натисканні клавіші викликається переривання клавіатури 9h, і код символу міститься в буфер клавіатури, що є областю пам'яті здатної запам'ятати до 15 символів. Взаємодію з клавіатурою забезпечує переривання BIOS **16h**. Функція **10h** даного переривання вибирає символ з буферу клавіатури; код символу повертається в регістрі AX. Функція **11h** перевіряє, чи мається в буфері клавіатури символ для зчитування. Якщо є, то прапор нуля мікропроцесора (ZF) встановлюється в 0, і в регістрі AX передається код символу, хоча сам символ залишається в буфері клавіатури. Якщо символу для зчитування немає, то прапор ZF встановлюється в 1. Функція **05h** переривання 16h дозволяє програмі занести визначений код символу в буфер клавіатури. Код передається через регістр CX. Якщо після повернення з переривання регістр AL містить 0, то занесення символу завершилося успішно. У противному випадку регістр AL містить 1, що говорить про те, що буфер клавіатури повний.

За допомогою описаних функцій переривання **16h** можна виконати перевірку типу клавіатури за методикою, рекомендованою в технічному описі BIOS. Для цього спочатку робиться спроба з допомогою функції **05h** занести в буфер клавіатури код **FFFFh** (цьому коду не відповідає ніяка клавіша), потім за допомогою функції 11h перевіряється наявність символу для зчитування, а за допомогою функції **10h** витягається символ з буфера. Якщо код витягнутого символу дорівнює **FFFFh**, то функції **05h, 10h** і **11h** функціонують коректно, що свідчить про наявність 101-клавішної клавіатури. Якщо ж хоча б одна з функцій працює некоректно, то на комп'ютері встановлена 83-клавішна клавіатура. Описана методика реалізована в наступній програмі:

```
#include<dos.h>
#include<process.h>
#define SPEC_CODE 0xFFFF
void retrieve(void);
union REGS r;
int main(void)
 {
   int i;
   for(i = 0; i <= 1; i++)
    { /*  Виконати дві спроби запису коду 0xFFFF у буфер  */
```

```
      r.x.cx = SPEC_CODE;
      r.h.ah = 0x5;
      int86(0x16, &r, &r);
      if(r.h.al)
        { /*   Буфер повний чи функція 5h не підтримується */
          r.h.ah = 0x10;
          int86(0x16, &r, &r);   /*  Витяг символу  */
        }else
          {
            retrieve();
            return 0;
          }
     }
   printf("83-кл. клавіатура\n ");
   return 0;
}

void retrieve(void)
 {
   int j;
   for(j = 0; j < 15; j++)  /* Цикл по всій довжині буфера  */
     {
       r.h.ah = 0x11;
       int86(0x16, &r, &r); /* Перевірка наявності
                               символу для зчитування      */
       if(r.x.flags & 0x40) /* Перевірка прапора ZF (у регістрі
                               прапорів це 6-й розряд)      */
         {
           printf("83-кл. клавіатура\n ");
           exit(0);
         }
         r.h.ah = 0x10;
         int86(0x16, &r, &r);  /*  Зчитування символу   */
         if(r.x.ax == SPEC_CODE)
          {
            printf("Розширена 101-кл. клавіатура\n");
            exit(0);
          }
      }
   printf("83-кл. клавіатура\n ");  }
```

7. Варіанти завдань

Розробити програму, що визначає :

1. Модель і назву фірми виготовлювача мікропроцесора, дані про BIOS і тип клавіатури.
2. Сімейство і частоту мікропроцесора, тип співпроцесора, розмір HDD,FDD і тип PC.
3. Частоту і назву фірми виготовлювача мікропроцесора, конфігурацію накопичувачів, об’єм ОЗУ.
4. Модель і частоту мікропроцесора, тип співпроцесора і розмір FDD,HDD.

5. Сімейство і назву фірми виготовлювача для МП; об’єм ОЗП, конфігурацію накопичувачів і тип клавіатури.
6. Сімейство, модель і частоту мікропроцесора, дані про BIOS, типи накопичувачів і розмір HDD.
7. Степінг, сімейство і назву фірми виготовлювача для МП, тип співпроцесора, дані про BIOS.
8. Модель і назву фірми виготовлювача мікропроцесора, тип FDD, об’єм ОЗП.
9. Назву фірми виготовлювача і частоту мікропроцесора, тип клавіатури, тип співпроцесора і тип накопичувачів.
10. Модель і частоту мікропроцесора, об’єм HDD, дані про BIOS .
11. Сімейство і частоту мікропроцесора, тип FDD, обсяг ОЗП .
12. Модель і частоту мікропроцесора, дані про BIOS, тип PC, і розмір HDD і FDD.
13. Сімейство, модель і частоту мікропроцесора, об’єм ОЗП і тип FDD.
14. Степінг і назву фірми виготовлювача мікропроцесора, тип співпроцесора, тип усіх накопичувачів.
15. Модель, назву фірми виготовлювача і частоту МП, конфігурацію накопичувачів, об’єм ОЗП і дані про BIOS.
16. Сімейство і частоту мікропроцесора, об’єм ОЗП і тип FDD.
17. Частоту і назву фірми виготовлювача мікропроцесора, тип співпроцесора, дані про BIOS і розмір FDD, HDD.
18. Модель і частоту мікропроцесора, тип FDD, об’єм ОЗП.
19. Модель, назву фірми виготовлювача і частоту мікропроцесора, дані про BIOS, розмір HDD.
20. Сімейство і частоту мікропроцесора, тип клавіатури, кількість і тип FDD.
21. Сімейство, модель і назву фірми виготовлювача мікропроцесора, тип співпроцесора, дані про BIOS і об’єм ОЗП.
22. Сімейство, модель і частоту мікропроцесора, тип клавіатури, об’єм ОЗП і версію BIOS.
23. Сімейство і назву фірми виготовлювача МП, тип PC і розміри FDD, HDD.
24. Модель і частоту мікропроцесора, тип співпроцесора, і дані про BIOS.
25. Сімейство, модель і назву фірми виготовлювача МП, конфігурацію накопичувачів, об’єм ОЗП і тип FDD.

8. Зміст звіту

1. Умова завдання.
2. Текст програми.
3. Результати роботи програми.

9. Контрольні питання

1. Які особливості визначення типу мікропроцесору різних платформ ?
2. Записати фрагменти програм визначення типу мікропроцесору різних платформ ?
3. Який алгоритм визначення частоти мікропроцесору?
4. Особливості виконання команди CPUID.
5. Фрагмент програми визначення характеристик BIOS.
6. Засоби визначення обсягу різних типів пам'яті.
7. Функції визначення типу периферійних пристроїв.
8. Алгоритм визначення числа і типу накопичувачів.
9. Робота переривань для визначення типу клавіатури.
10. Алгоритм визначення накопичувачів на магнітних дисках.

11. Визначити тип співпроцесора і дані про BIOS.
12. Визначити назву фірми виготовлювача МП.
13. Визначити розмір FDD,HDD.

ЛАБОРАТОРНА РОБОТА №4

ДОСЛІДЖЕННЯ ОРГАНІЗАЦІЇ ОПЕРАТИВНОЇ ПАМ'ЯТІ

Мета роботи - дослідження організації пам'яті реального та захищеного режиму роботи процесору

1. Організація пам'яті реального режиму

Для сучасних процесорів максимальний обсяг пам'яті, що адресується, дорівнює 64 Гбайт. Адресний простір комп'ютерів IBM PC при роботі їх у реальному режимі являє собою одномірний масив байт, кожний з яких має 20-розрядну фізичну адресу. Зручно вважати, що кожна комірка пам'яті має дві адреси: фізичну та логічну. Фізична адреса являє собою 20-розрядне значення в діапазоні від 0 до FFFFF, що визначає положення кожного байта в просторі пам'яті 1 Мб. Логічний адрес складається з двох 16-розрядних без знакових значень: базового (початкового) адресу сегмента і зсуву усередині сегмента. Сегмент являє собою логічну одиницю пам'яті розміром 64 Кбайт. Усі сегменти починаються на 16-байтних границях пам'яті, які називаються межами параграфів. Для будь-якої комірки пам'яті база ідентифікує перший байт її сегмента, тобто початок сегмента, а зсув визначає відстань у байтах від початку сегмента до цього осередку.

Адресний простір комп'ютерів IBM PC у межах 1 Мбайт розподіляється відповідно до табл. 1.

2. Область даних BIOS

Область основної пам'яті розміром 256 байт, розташована безпосередньо за таблицею векторів переривань, - починаючи з адреси 0040:0000 і закінчуючи 004F:0000 - призначена для використання програмами BIOS. У табл. 2 приведений опис частини інформації, що поміщається системою BIOS у зазначену область пам'яті. Звертаючись до неї, прикладні програми одержують важливу інформацію про стан системи. Наприклад, прочитавши байт за адресою 0040:0013, програма може визначити обсяг оперативної пам'яті в кілобайтах.

У моделях комп'ютерів IBM PC використовується операція "затінення" (Shadow) для системного ПЗП BIOS і BIOS відеоадаптерів VGA. Shadow Memory - це так звана "тіньова" пам'ять. В адресах пам'яті від 640 Кб до 1 Мб (A0000h - FFFFFh) знаходяться "вікна", через які "видно" зміст різних системних ПЗП. Наприклад, адреси F0000h- FFFFFh займають системне ПЗП, що містить BIOS системи, вікно C0000h - C7FFFh - ПЗП відеоадаптера (відео BIOS) і т.п. При включенні режиму Shadow для яких-небудь адресних діапазонів, що відповідають системним ПЗП або картам розширення, зміст їх ПЗП копіюється у ділянки основної пам'яті, що потім підключаються до цих же адрес замість ROM, "затінюючи" їх. Для реалізації операції "затінення" в оперативній пам'яті, крім основної ділянки області даних BIOS, що починається по адресі 0040:0000, виділяється додаткова ділянка, що має назву розширеної області даних BIOS (табл.1).

Характеристики розширеної області даних BIOS можна визначити шляхом виклику функції **C0h** переривання **15h.** Якщо після повернення з переривання прапор переносу містить 0, то розширена область BIOS даних підтримується. Через пару регістрів ES:BX повертається покажчик на початок таблиці системного середовища, яке містить характеристики розширеної області даних BIOS.

Таблиця 1

Адреса (сегмент зсув)	Довжина (у байтах)	Залежність від	Найменування і опис
0000:0000	1024		Таблиця векторів переривань 256 4-байтових адрес
0040:0000	256		Область даних BIOS
0050:0000	256		Область даних DOS
0060:0000	256		Область даних DOS, використована тільки при завантаженні
XXXX:0000	8928	версії MS DOS	Код BIOS (прочитаний з IO.SYS завантажувального диску)
XXXX:0000	29920	версії MS DOS	Оброблювачі переривань DOS, включаючи INT 21h (MSDOS.SYS)
XXXX:0000		конфігу-рації	DOS: буфера, область даних і встановлювані драйвери
XXXX:0000			Резидентна частина COMMAND.COM, включаючи оброблювач INT 22h,23h, 24h
XXXX:0000			Резид. програми і дані
XXXX:0000			Поточна виконувана програма (типу .COM чи .EXE).
XXXX:0000		конфігу-рації	Транзитна частина COMMAND.COM. перезавантажена якщо була чимось перекрита
XXXX:0000			Розширена область даних BIOS для комп'ютера PS/2
			Границя 640 К зони
A000:0000			EGA-, VGA-пам'ять для деяких відеорежимів
B000:0000			Відеопам'ять монохромного адаптера й адаптера Hercules
B800:0000			Відеопам'ять CGA (Hercules)
C000:0000 до E000:0000			Зовнішній код ПЗП. ПЗП BIOS шукає тут (у 2 К-блоках) код, виконує мий при завантажен.
також			Сторінковий блок EMS
E000:0000 до E000:FFFF			Модулі ПЗП материнської плати у блоках по 64К (тільки системної плати IBM PC AT).
F600:0000			ПЗП-резидентний інтерпретатор BASIC (для PC фірми IBM)
FE00:0000			ПЗП - BIOS: програма автоматичного тестування POST і ін.
F000:FFF0			Команда JMP на програму, виконувану при вмик/скиданні
F000:FFF5:8	8		Дата видання BIOS
F000:FFFE	1		Ідентифікаційний код IBM PC

Таблиця 2.

Адреса (сегмент зсув)	Довжина (у байтах)	Найменування і опис
		ОБЛАСТЬ ДАНИХ ПОРТІВ
0040:0000	8	Базова адреса вводу-виводу для COM1-COM4
0040:0008	8	Базова адреса вводу-виводу для LPT1-LPT4
- - - - - - - -	- - - - - - - - - - - - - -	- ЗМІШАНА ОБЛАСТЬ ДАНИХ- - - - - - - - - - - - - -
0040:0010	2	Прапори устаткування
0040:0013	2	Обсяг пам'яті в кілобайтах
- - - - - - - -	- - - - - - - - - - - - - -	- ОБЛАСТЬ ДАНИХ КЛАВІАТУРИ- - - - -
0040:0017	1	Прапори 1 стану регістрів клавіатури
0040:0018	1	Прапори 2 стану регістрів клавіатури
0040:0019	1	Уведення з додаткового клавіатурного поля
0040:001A	2	Адреса початку буфера клавіатури
0040:001C	2	Адреса кінця буфера клавіатури
0040:001E	32	Буфер клавіатури
- - - - - - - -	- - - - - - - - - - - - - -	- ОБЛАСТЬ НАКОПИЧУВАЧА FDD- - - - -
0040:003E	1	Стан повторного калібрування
0040:003F	1	Стан двигуна
0040:0040	1	Лічильник числа відключень двигуна
0040:0041	1	Стан останньої операції
0040:0042	7	Байти стану контролера
- - - - - - - -	- - - - - - - - - - - - - -	- ОБЛАСТЬ ДАНИХ ВІДЕОАДАПТЕРА- - - - -
0040:0049	1	Поточний відеорежим
0040:004A	2	Кількість стовбців у відображуваному тексті
0040:004C	2	Довжина буфера регенерації в байтах
0040:004E	2	Адреса зсуву активної відеосторінки
0040:0050	16	Положення курсору (відеосторінки 0-7)
0040:0060	2	Тип курсору (початок. і кінець рядка розгорнення)
0040:0062	1	Активна відеосторінка
0040:0063	2	Базова адреса відеоконтролера
0040:0065	1	Поточна установка регістра 3x8
==========	==========	==========
0040:0066	1	Поточна установка регістру
- - - - - - - -	- - - - - - - - - - - - - -	- ОБЛАСТЬ ДАНИХ СИСТЕМНОГО ТАЙМЕРА- -
0040:006C	2	Молодше слово вмісту таймера
0040:006E	2	Старше слово вмісту таймера
0040:0070	1	Індикатор заповнення таймера
- - - - - - - -	- - - - - - - - - - - - - -	-СИСТЕМНА ОБЛАСТЬ ДАНИХ- - - - - -
0040:0071	1	Байт BIOS_BREAK
0040:0072	2	Прапор скидання
- - - - - - - -	- - - - - - - - - - - - - -	-ОБЛАСТЬ ДАНИХ ТВЕРДОГО ДИСКА- -
0040:0074	1	Стан останньої операції
0040:0075	1	Число підключених накопичувачів HDD
- - - - - - - -	- - - - - - - - - - - - - -	- ЧАСИ ЧЕКАННЯ РЕАКЦІЇ (ТАЙМАУТИ)- - -
0040:0078	4	Значення часів чекання для LPT1-LPT4
0040:007C	4	Значення часів чекання для COM1-COM4
- - - - - - - -	- - - - - - - - - - - - - -	- ОБЛАСТЬ ДАНІ КЛАВІАТУРИ- - - - - -
0040:0080	2	Адреса початку буфера клавіатури
00040:0082	2	Адреса кінця буфера клавіатури
- - - - - - - -	- - - - - - - - - - - - - -	- ОБЛАСТЬ ДАНИХ ВІДЕОАДАПТЕРА- - - -
0040:0084	1	Число відображуваних рядків тексту мінус 1
0040:0085	2	Висота символу в рядках розгорнення
0040:0087	1	Параметри відеорежиму
0040:0088	1	Параметри відеорежиму
- - - - - - - -	- - - - - - - - - - - - - -	- ОБЛАСТЬ ЗВ'ЯЗКУ МІЖ ПРОГРАМАМИ- - -
0040:00F0	16	Область, де програма може чи записати введені дані (наприклад, статус).

3. Енергонезалежна пам'ять

Персональні комп'ютери класів IBM PC містять годинник реального часу і енергонезалежну пам'ять (CMOS) ємкістю 64 байта, що одержують живлення від батарейки чи акумулятора. CMOS - база даних, призначена для збереження інформації про конфігурацію ПК. CMOS зберігає свої дані на мікросхемі багаторазового запису (write many-read many). Програма установки BIOS SETUP при записі зберігає в ній свою системну інформацію, яку згодом сама ж і зчитує при завантаженні ПК. Кожна частка має розмір у 1 байт. У даній пам'яті зберігається різна інформація, що включає поточну дату і поточний час, конфігурацію устаткування і т. і. (табл. 3).

Таблиця 3

Адреса	Опис
00H-0d	використовується годинник реального часу (RTC)
0e	байт стану після автоматичного тестування POST
0f	байт стану при поверненні в реальний режим
10H	тип накопичувача на гнучких дисках
11H	(зарезервовано)
12H	тип накопичувачів на твердих дисках (якщо < 15)
13H	(зарезервовано)
14H	байт конфігурації устаткування
15H-16H	розмір базової пам'яті
17H-18H	обсяг вище 1M
19H	тип твердого диска C (якщо > 15)
1a	тип твердого диска D (якщо > 15)
1bH-20H	(зарезервовано)
21H-2d	(зарезервовано)
2eH-2f	пам'ять для контрольної суми вмісту адрес від 10h до 20h
30H-31H	обсяг розширеної пам'яті вище 1M
32H	поточний, в двоїчно-десятковому форматі (наприклад, 20) BCD
33H	допоміжна інформація
34H-3f	(зарезервовано)

Контрольна сума являє собою 16-бітну суму всіх значень, записаних в осередки CMOS з 10H по 20H. В осередок 2EH пишеться старший байт суми, а в 2FH - молодший. Більш докладно RTC:

Адреса (HEX)	Опис
00H	Поточна секунда
01H	Сигнальна секунда
02H	Поточна хвилина
03H	Сигнальна хвилина
04H	Поточна година
05H	Сигнальна година
06H	Поточний день тижня (1 - Неділя)
07H	Поточний день місяця
08H	Поточний місяць
09H	Поточний рік (тільки 2 останні цифри, напр. 98)

Усі значення RTC змережуються в BCD форматі як 2 напівбайти але в десятковому форматі. Наприклад, 31 (dec) зберігається як 31 (hex). Докладніше окремі байти CMOS:

Адреса Опис

- -

0EH Байт діагностики завантаження (POST Byte):
Біти 0 і 1 завжди рівні 0.
Біт 2 - Час вірний (1=вірно, що сьогодні не 30 лютого)
Біт 3 - Невірний завантажувальний твердий диск (1=не можна завантажитися з вінчестеру)
Біт 4 - Помилка розміру RAM (1=POST знайшла невірний розмір RAM)
Біт 5 - Невірний запис про устаткування (1=невірне устаткування)
Біт 6 - Невірна контрольна сума (1=невірна сума CMOS)
Біт 7 - Утрата живлення батареї CMOS (1=утрата живлення)

0FH Байт статусу завершення роботи ПК. Застосовується найчастіше після перезавантаження ПК процедурою SETUP. Значення можуть бути наступні:
0 - якщо було перезавантаження по натисканні Ctrl-Alt-Del чи несподіваний перезапуск. У будь-якому випадку процедура POST не виконується
1 - перезапуск після визначення розміру пам'яті
2 - перезапуск після тесту пам'яті
3 - перезапуск після виявлення помилки пам'яті
4 - перезапуск по запиті завантажника ОС
5 - перезапуск унаслідок далекого переходу (FAR JMP) на адресу 0:0467H
6,7,8 - перезапуск після перевірки захищеного режиму 80286
9 - перезапуск після перепризначення блоку пам'яті (функція 0x87 переривання 0x15)

10H Байт типу дисковода:
Біти 0-3: перший дисковод
Біти 4-7: другий дисковод
0100 = 4 = 1,44 Мб

12H Тип вінчестера (для дисків C: і D:, коли байт знаходиться у проміжку від 1 до 14). Біти 0-3: перший вінчестер. Біти 4-7: другий вінчестер У будь-якому випадку, значення бітів можуть бути наступними: 0000 = 0 - диск не встановлений, інше_значення = тип диска 1111 - дивись адреси 19H и 1AH.

14H Байт устаткування:
Біт 0 = 1, якщо є присутнім дисковод(и)
Біт 1 = 1, якщо є присутнім математичний співпроцесор
Біти 2, 3 не використовуються і рівні 0
Біти 5, 4 - основний відеоадаптер: · 00 - немає чи EGA ·
01 - 40*25 EGA, CGA, VGA ·
10 - 80*25 EGA, CGA, VGA ·
11 - монохромний (ч/б)
Біти 6, 7 - кількість дисководів - 1 (00=1, 01=2, 10=3, 11=4)

15H, 16H Базова пам'ять: 15H - молодший байт 16H - старший байт.

17H, 18H Додаткова пам'ять понад 1 Мб (17H - молодший байт, 18H - старший байт). Розмір записаний у Кб.

19H Тип диска № 0 (3:), якщо значення адреси (12H & 0FH) = 0FH

20H Тип диска № 1 (D:), якщо значення адреси (12H & F0H) = F0H

2EH, 2FH Контрольна сума значень адрес від 10H по 20H 2EH - старший байт · 2FH - молодший байт

34H-3FH РЕЗЕРВ.

Для того, щоб прочитати байт із енергонезалежної пам'яті, необхідно спочатку послати необхідну адресу байта в порт **70h**, а потім виконати команду читання байта з порту **71h.**

Приклад: Визначити розмір розширеної пам'яті

```
/*  Визначення наявності і розміру розширеної   пам'яті */
#include<stdio.h>
#include<dos.h>
#define AT          0xFC
#define PS_2__30    0xFA
#define PS_2__80    0xF8
int main(void)
{
   unsigned char type_ibm, high_byte, low_byte;
   type_ibm = peekb(0xF000, 0xFFFE);
   if(type_ibm == AT || type_ibm == PS_2__30 ||
      type_ibm == PS_2__80)
     {
       outportb(0x70, 0x17);
       low_byte=inportb(0x71);
       outportb(0x70, 0x18);
       high_byte=inportb(0x71);
       printf("Обсяг розширеної пам'яті %u Кбайт\n",
               high_byte * 256 + low_byte);
     }
    else printf("Розширена пам'ять відсутня\n");
   return 0;
 }
```

Програмувати CMOS бажано з реального режиму ОС. Для ОС Windows це робиться за допомогою спеціальних драйверів VxД чи SYS.

4. Визначення наявності і розміру додаткової відображаємої пам'яті

Для задоволення специфікації LIM EMS 4.0 менеджер розширеної пам'яті реалізує 28 різних функцій, у багатьох з яких є багато підфункцій. При програмуванні необхідно:

- помістити код функції для функції специфікації, що вимагається, розширеної пам'яті в регістр AH;
- помістити інші аргументи, необхідні для обраної функції, і/чи структури даних у пам'ять;
- передати керування менеджеру розширеної пам'яті шляхом видачі програмного переривання 67h;
- менеджер розширеної пам'яті повертає керування програмі, що видала запит, перезаписуючи код функції, поміщений у регістр AH, кодом для запитаної операції. Код стану 00h сигналізує про успішне завершення функції. Будь-яке інше значення показує, що менеджер розширеної пам'яті наштовхнувся на які-небудь проблеми. Значення кодів помилок і їхній зміст перераховані далі.

Для маніпулювання додатковою відображуваною пам'яттю використовується переривання **67h.** Функція **42h** даного переривання повідомляє:

- у регістрі DX сумарна кількість сторінок EMS по 16Кб кожна;
- у регістрі BX число наявних у даний момент вільних сторінок;

- у регістрі AH статус після виконання функції (якщо 0, операція завершилася успішно, інакше - помилка).

Однак використовувати переривання **67h** можна тільки за умови, що в комп'ютері забезпечена апаратна і програмна підтримка додаткової відображуваної пам'яті. Тому перед викликом переривання **67h** необхідно переконатися в тім, що в ланцюжку драйверів є драйвер додаткової відображуваної пам'яті (у поле ім'я_пристрою заголовка драйвера ДОП повинне бути записане ім'я EMMXXXX0).

Приклад. Визначення наявності і розміру додаткової відображуваний пам'яті (встановити значення Alignment рівне byte):

```
#include<stdio.h>
#include<dos.h>
int main(void)
{
  union REGS regs;
  struct SREGS segregs;
  typedef struct _DRIVER
   {
     struct _DRIVER far *pNext;
     int DevAttr;
     unsigned int Strategy;
     unsigned int Interrupt;
     char sName[8];
   };
  struct _DRIVER far *ptr;
  unsigned int offset;
  regs.h.ah = 0x52;
  intdosx(&regs, &regs, &segregs);
  ptr = MK_FP(segregs.es, regs.x.bx + 0x22);
  offset = FP_OFF(ptr);
  while(offset != 0xFFFF)
    ( if(ptr->DevAttr < 0)
      if(!strcmp(ptr->sName, "EMMXXXX0"))
       {
         regs.x.ax = 0x4200;
         int86(0x67, &regs, &regs);
         if(regs.h.ah)
               printf("Помилка при звертанні до адміністратора
               EMS\n");
          else  printf("Обсяг додаткової пам'яті %lu Кбайт\n",
                        (unsigned long)regs.x.dx * 16);
         return 0;
       }
     ptr=ptr->pNext;
     offset = FP_OFF(ptr);
    }
  printf("Додаткова пам'ять не підтримується\n");
  return 0;
 }
```

Якщо менеджер розширеної пам'яті існує і готовий обслуговувати запити, можна видати функцію 7 "Одержати версію", для того щоб упевнитися, що версія менеджера розширеної пам'яті, з яким програма спілкується, підтримує версію специфікації розширеної пам'я-

ті, що їй потрібна. Ця функція повертає число в двоічно-десятковому коді з двох цифр у регістрі AL. Старші чотири біти числа показують основний номер версії. Молодші чотири біти чи дробова частина цього числа можуть використовуватися постачальниками для позначення специфіки виробників менеджерів розширеної пам'яті. Отже, програмі краще виконати порівняння на "більше чи дорівнює".

5. Визначення наявності й адрес зовнішніх ПЗП

У комп'ютері IBM PC існує єдиний адресний простір для оперативної і постійної пам'яті. Між адресами C000:0000 і E000:0000 можуть бути встановлені зовнішні ПЗП на платах відеоадаптерів EGA, VGA, контролера твердого диска, адаптера мережі і т.п. Адресний простір ПЗП можна вважати розбитим на дві частини: область від C000H до C7FFFH сканується раніш виконання більшості тестів, інший адресний простір C8000H - EFFFFH сканується після виконання більшості тестів.

Перша область може містити ПЗП адаптерів використовуваних процедурами ініціалізації (наприклад, ПЗП дисплея). Обидві області скануються з кроком рівним 2Кбайта в пошуках коректних модулів ПЗП адаптерів.

Коректний модуль ПЗП містить наступну інформацію:

байт 0 - 55H,

байт 1 - 0AAH,

байт 2 - довжина модуля в 512-байтових одиницях (максимально 64К);

байт 3 - крапка входу в модуль ініціалізації, що викликається із BIOS при виконанні тестів включення живлення.

Модуль ініціалізації викликається інструкцією між сегментного виклику і повинний повернути керування BIOS інструкцією міжсегментного повернення.

Таким чином необхідно виконати сканування даної ділянки пам'яті з метою виявлення зовнішніх ПЗП. Кожен блок розміром 2 Кбайт у зазначеному діапазоні перевіряється на сигнатуру AA55h, з якої починаються зовнішні ПЗП . При встановленні сигнатури AA55h перевіряється контрольна сума модуля ПЗП: усі байти підсумовуються по модулю 100h, у результаті чого повинне бути отримане значення 0.

Приклад. Визначити наявності і початкові адреси додаткових ПЗП

```
#include<stdio.h>
#include<dos.h>
unsigned char check_sum(unsigned);
int main(void)
 {
   unsigned segment, signature;
   for(segment = 0x000; segment < 0x000; segment += 128)
    {   /* 128 параграфів складає 2 Кбайт */
     signature = (unsigned)peek(segment, 0);
     if(signature == 0xAA55)
      if(!check_sum(segment))
        printf("Мається ПЗП за адресою %04X:0000\n", segment);
    }
   return 0;
 }

unsigned char check_sum(unsigned segment)
 {
  int i;
  unsigned char sum = 0;
```

```
  long length = (long)peekb(segment, 2) * 512L;
  for (i = 0; i < length; i++)
    sum += (unsigned char)peekb(segment, i);
  return sum;
 }
```

6. Робота з XMS - пам'яттю

Робота з XMS реалізується при наявності в ОЗП драйвера himem.sys.- шляхом визначення адреси функції - диспетчера XMS. Режими для XMS-manager є int **2fh**, funcnion **4300h, 4310h.**

Приклад . Використання функцій по роботі зXMS пам’яттю

```
#include <stdio.h>
#include <dos.h>
union REGS rr;
  /*Структура для обміну блоками пам'яті*/
struct XMS_MOVE
 { unsigned long length;
   unsigned int  source_handle;
   unsigned long source_off;
   unsigned int  dest_handle;
   unsigned long dest_off;
 };
struct XMS_MOVE xmov;
/*     Покажчик на функцію - менеджер XMS пам'яті:*/
void far (*xms_entry_point)(void);
/*     Функція, що організує доступ до XMS:*/
 int xms_install(void)
 { _AX=0x4300;
  geninterrupt(0x2F);
  if (_AL==0)  return 1;
  _AX=0x4310;
  geninterrupt(0x2F);
  xms_entry_point=(void far (*)(void))MK_FP(_ES,_BX);
  return 0;
 }
/*      Функція обробки входу в XMS - пам'ять:*/
 int xms_go(char a, int d)
 { _AH = a;
   _DX = d;
  xms_entry_point();
  rr.x.ax = _AX;
  rr.x.bx = _BX;
  rr.x.dx = _DX;
  if (rr.x.ax==0)  return 1;
  return 0;
 }
/*      Функція маніпулювання з блоками XMS і ConvertMem:*/
 int xms_movblk(long len, int hnd1, long off1,
                          int hnd2, long off2)
 { xmov.length = len;            /*  Exactly byte*/
```

```
   xmov.source_handle = hnd1;     /*from   (0)*/
   xmov.source_off = off1;        /*offset (abs adr)*/
   xmov.dest_handle = hnd2;       /*to     (0)*/
   xmov.dest_off = off2;          /*offset (abs adr)*/
   _SI = FP_OFF(&xmov);
   if (xms_go(0x0B,0))  return 1;
 return 0;
 }
```

Приклад організації роботи з XMS пам’яттю.

```
 int main(void)
 {     unsigned int  handle;
       unsigned long r;
       unsigned char far *txt="0123456789";
/*Інсталяція*/
    if (xms_install()) { printf("\nError XMS"); return 1; }
/*Захоплення блоку  XMS - 3kb*/
    if (!xms_go(0x09,3))  printf("\nAllocXMS= %X", rr.x.dx);
    handle=rr.x.dx;
/*Перенести блок 10 байт  і XMS у ConvertMem*/
    r = (unsigned long)txt;     //abs address
    xms_movblk(10, handle,0, 0,r);
/*Захоплення ще одного блоку XMS  - 12kb*/
    if (!xms_go(0x09,12))  printf("\nAllocXMS= %X", rr.x.dx);
    handle=rr.x.dx;
/*Звільнити останній блок XMS -12Kb*/
    if (!xms_go(0x0B,handle)) printf("\n Free");
}
```

7. Тестування областей пам'яті

Приклади тестування.

Приклад тестування відеопам'яті.

```
void V_MemWin( void )
{  char res,i;
   union REGS rr;
   int size, j;
/* Визначення розміру відеопам'яті int10 Fun12*/
  rr.h.ah = 0x12;
  rr.h.bl = 0x10;
  int86(0x10, &rr, &rr);
  res = rr.h.bl;    //Розмір відеопам'яті
  switch(res)
    { case 0:   size=64;   break;
      case 1:   size=128;  break;
      case 2:   size=192;  break;
      case 3:   size=256;  break;
    }
  rr.h.ah = 0x0f;
  int86(0x10, &rr, &rr);
```

```
/* Запам'ятати номер активної відео сторінки*/
  res = rr.h.bh;
  i=0;
  for(j=0;j<size/64;j++)
/* Читати усі відео сторінки*/
    { rr.h.ah = 0x05;
      rr.h.al = i;     // це її номер
      int86(0x10, &rr, &rr);
      aindow(1,1,24,77);
      textbackground(i);  clrscr();
      printf("\n VIDEO_PAGE N %d ",i);
      getch();
      i++;
    }

/* Відновити номер активної відео сторінки*/
  rr.h.ah = 0x05;
  rr.h.al = res;
  int86(0x10, &rr, &rr);
}
```

Тестування DMA і контролера переривань

```
 char Test1,Test2, Flag,i,j;
  Flag=0;          //*** Тест ПДП
  for(i=0;i<3;i++)
    { for(j=0;j<8;j++)
        { Test1=inportb(j);
          outportb(j,Test1);
          Test2=inportb(j);
          if (Test1!=Test2)  {  Flag=1;  break; }
        }
  ...

  Flag=0;  /* Тест КОНТРОЛЕРА  ПЕРЕРИВАНЬ*/
  disable();
  for(i=0;i<3;i++)
  for(j=0x20;j<0x22;j++)
    { Test1=inportb(j);
      outportb(j,Test1);
      Test2=inportb(j);
      if (Test1!=Test2)  {  Flag=1;  break; }
    }
  enable();
  ...
```

Перевірка роботи DMA і контролера переривань. Обсяг XMS,

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

#define AT          0xFC
```

```
#define PS_2__30    0xFA
#define PS_2__80    0x8

/* Покажчик на функцію - менеджер XMS пам'яті:*/
void far (*xms_entry_point)(void);
union REGS rr;

void test_DMA();
int XMS_size();
void A20_check();
int xms_install(void);
int xms_go(char a, int d);

void main()
{
 clrscr();
 test_DMA();
 XMS_size();
 A20_check();
 getch();
}

/* Test DMA and interrupt controller*/
void test_DMA()
{
  char Test1,Test2, Flag,i,j;
  Flag=0;          // Test DMA
  for(j=0;j<8;j++)
      {
          Test1=inportb(j);
          outportb(j,Test1);
          Test2=inportb(j);
          if (Test1!=Test2)
          {
          Flag=1;  break;
          }
      }
  if(Flag) printf("DMA is absent\n");
  else     printf("DMA present\n");

  Flag=0;  // Test interrupt controller
  disable();
  for(j=0x20;j<0x22;j++)
      { Test1=inportb(j);
          outportb(j,Test1);
          Test2=inportb(j);
          if (Test1!=Test2)  {  Flag=1;  break; }
      }
  enable();
  if(Flag) printf("Interrupt controller is absent\n");
  else     printf("Interrupt controller is present\n");
}
```

```
/* Check for XMS size*/
int XMS_size()
{
  union REGS reg;
  unsigned char type_ibm, high_byte, low_byte;

  /* Check for XMS present*/
  printf("\nTESTING XMS MEMORY:");
  reg.x.ax=0x4300;
  int86(0x2f,&reg,&reg);
  if(reg.h.al==0x80)  printf(" OK.\n");
  else
      {
       puts(" Memory error, or HIMEM is absent.");
       return 0;
      }

  /* Check for XMS size*/
  type_ibm = peekb(0xF000, 0xFFFE);
  if(type_ibm == AT || type_ibm == PS_2__30 ||
       type_ibm == PS_2__80)
      {
          outportb(0x70, 0x17);
          low_byte=inportb(0x71);
          outportb(0x70, 0x18);
          high_byte=inportb(0x71);
          printf("Size of XMS %u Кбайт\n",
                      high_byte * 256 + low_byte);
      }
     else printf("XMS is absent\n");
}

/* Check for enabled line A20*/
void A20_check()
{
  _AX=0x4300;
  geninterrupt(0x2F);
  if(_AX) printf("\nHIMEM is enabled\n");
  else    printf("\nHIMEM  is disabled\n");

     if(xms_install()) printf("Cannot install XMS\n");
     else
          if(xms_go(0x07,0x002f))
               printf("A20 is disabled\n");
          else
               printf("A20 is phisically enabled\n");
}

/*Функція, що організує доступ до XMS:*/
 int xms_install(void)
 {
     _AX=0x4300;
     geninterrupt(0x2F);
```

```
    if (_AL==0)  return 1;
  _AX=0x4310;
  geninterrupt(0x2F);
  xms_entry_point=(void far (*)(void))MK_FP(_ES,_BX);
  return 0;
 }

/*Функція обробки входу в XMS - пам'ять:*/
 int xms_go(char a, int d)
 {
     _AH = a;
     _DX = d;
  xms_entry_point();
  rr.x.ax = _AX;
  rr.x.bx = _BX;
  rr.x.dx = _DX;
  if (rr.x.ax==0)  return 1;
  return 0;
 }
```

7.1. Методи тестування ОЗУ

Тест ОЗП методом послідовного запису/читання. У кожний блок пам'яті послідовно записуються нулі. Потім байти блоку зчитуються і порівнюються з нулем. Далі, у той же блок, пишуться одиниці (FFh) і ті ж дії повторюються.

Тест пам'яті методом шахового коду. З початку блоку по два байти послідовно пишуться коди AAh і 55h, потім перевіряється правильність їхнього зчитування.

Тест пам'яті методом зчитування/запису. У блок пам’яті послідовно пишуться нулі, потім кожен байт блоку зчитується, порівнюється з нулем і туди пишуться одиниці (FFh). Коли досягнутий кінець блоку, від кінця до початку зчитуються раніше записані одиниці і пишуться нулі.

Тест пам'яті методом парної/непарної адреси**.** З початку блоку послідовно у байти з парною адресою записуються нулі, а з непарною адресою - одиниці (FFh). Потім байти з початку блоку зчитуються і парні порівнюються з нулем, а непарні з кодом FFh.

Тест пам'яті методом «бігучої» одиниці**.** У блок спочатку записуються нулі, потім у перший байт блоку записується одиниця**.** Відбувається читання всіх байтів від кінця блоку до його початку і їхнє порівняння з кодами 00h і FFh. Коли досягнутий початок блоку, у другий байт пишеться одиниця, і ті ж дії повторюються.

Алгоритм попарного зчитування забезпечує будь-які адресні переходи з різною зміною інформації при зчитуванні. У початкову адресу записується одиниця на фоні всіх інших нулів, а потім зчитуються адреси перший із другим, перший із третім і так далі до останнього, потім друга адреса з першим, другий з третім і так далі.

Приклад .Визначити обсяг та протестувати пам'ять**.**

```
#include <dos.h>
#include <stdio.h>
#include <string.h>
union REGS rr;
struct SREGS srr;
unsigned char far *ll;
unsigned int fmcb,dos_seg;
struct mcb
```

```
{
 char type;
 unsigned int owner,size;
 char unnved[3];
 char name[9];
}mcb;

char zbl=0,tot=0;
void main (void)
{
 int p,i,n;
 unsigned int temp,d_s;
 rr.x.ax=0x5200;
 intdosx(&rr,&rr,&srr);
 ll=MK_FP(srr.es,rr.x.bx);
 dos_seg=srr.es;
 fmcb=ll[-1]<<8 | ll[-2];
 temp=fmcb;d_s=dos_seg;
 while (zbl<2)
   {
    movedata(temp,0,_DS,(unsigned)&mcb.type,0x10);
    mcb.name[8]=0;
    if((mcb.type!='M')&&(mcb.type!='Z')&&(zbl>0))
      break;//not UMB
    printf("\n %c",mcb.type);
    printf("   %0.4x",temp+1);
    printf("   %0.4x",mcb.owner);
    printf("   %0.4x",mcb.size);
    if(mcb.owner==0)
      {printf("     Free Block");
       n=mem_test(temp,temp+mcb.size+1);
       if(n)
      printf(" Test Ok");
       else
      printf(" Trouble");
       }
    else
      if(mcb.owner==temp+1)
     printf("  %s",mcb.name);
      else
     if(mcb.owner<=dos_seg)
       printf("     System\n");
     else
       printf("     Enviroment\n");
    temp+=mcb.size+1;
    tot++;
    if(mcb.type=='Z')
      zbl++;
   }
}
int mem_test (unsigned int beg,unsigned int end)
{
 int n=1;
```

```
 unsigned long i,j,k,z;
 unsigned char test,test1;
 for(i=beg+1;i<end;i++)
   for(j=0;j<16;j++)
     {
      if(!(z=j%2))
     pokeb(i,j,0);
      else
     pokeb(i,j,0xff);
      }
 for(i=beg+1;i<end & n ;i++)
   for(j=0;j<16;j++)
     {
      if(!(z=j%2))
     {
      test=peekb(i,j);
      if(test!=0)
         {n=0;return;}
     }
      else
     {
      test=peekb(i,j);
      if(test!=0xff)
        {n=0;return;}
      }
     }
 return n;
}
```

9. Варіанти завдань

Розробити програму, що визначає:

1. Список блоків основної пам'яті з адресами і розмірами. Загальний і доступний обсяг XMS.
2. Список драйверів, з адресами заголовків і пристроїв. Обсяг і версію XMS-пам'яті.
3. Обсяги основної і всієї пам'яті - з енергонезалежної. Наявність і адреси зовнішніх ПЗП.
4. Перевірити роботу сторінок відеопам'яті. Обсяг XMS, перевірити наявність включеної лінії A20.
5. Перевірити роботу DMA і контролера переривань. Доступний обсяг XMS, версію XMS - пам'яті.
6. Список драйверів, з адресами заголовків і пристроїв. Загальний і доступний обсяг XMS.
7. Список блоків основної пам'яті з адресами і розмірами. Обсяг і версію XMS-пам'яті.
8. Перевірити роботу сторінок відеопам'яті. Наявність і адреси зовнішніх ПЗП.
9. Обсяг XMS, перевірити наявність включеної лінії A20. Наявність і адреси зовнішніх ПЗП.
10. Список блоків основної пам'яті з адресами і розмірами. Доступний обсяг XMS, версію XMS - пам'яті.
11. Обсяги основної і всієї пам'яті - з енергонезалежної. Обсяг і версію XMS-пам'яті.
12. Перевірити роботу DMA і контролера переривань. Обсяг XMS, перевірити наявність включеної лінії A20.

13. Обсяги основної і всієї пам'яті - з енергонезалежної. Загальний і доступний обсяг XMS.
14. Перевірити роботу сторінок відеопам'яті. Обсяг і версію XMS-пам'яті.
15. Перевірити роботу DMA і контролера переривань. Наявність і адреси зовнішніх ПЗП.
16. Список блоків основної пам'яті з адресами і розмірами. Доступний обсяг XMS, версію XMS - пам'яті.
17. Список драйверів, з адресами заголовків і пристроїв. Обсяг XMS, перевірити наявність включеної лінії A20.
18. Список блоків основної пам'яті з адресами і розмірами. Загальний і доступний обсяг XMS.
19. Список драйверів, з адресами заголовків і пристроїв. Обсяг і версію XMS-пам'яті.
20. Перевірити роботу сторінок відеопам'яті. Обсяг XMS, перевірити наявність включеної лінії A20.
21. Перевірити роботу DMA і контролера переривань. Доступний обсяг XMS, версію XMS - пам'яті.
22. Список драйверів, з адресами заголовків і пристроїв. Загальний і доступний обсяг XMS.
23. Список блоків основної пам'яті з адресами і розмірами. Обсяг і версію XMS-пам'яті.
24. Перевірити роботу сторінок відеопам'яті. Наявність і адреси зовнішніх ПЗП.
25. Обсяги основної і всієї пам'яті - з енергонезалежної. Загальний і доступний обсяг XMS.

10. Зміст звіту

1. Умова завдання.
2. Текст програми.
3. Результати.

Контрольні питання

1. Що є адресою слова? Апаратне формування адреси.
2. Привести приклади адрес рівних границям сегментів і номерам параметрів.
3. Механізм відображення додаткової пам'яті.
4. Що таке область верхньої пам'яті і блоки верхньої пам'яті ?
5. Намалювати схему розподілу пам'яті ОЗП 2МБ.
6. Реальний режим роботи мікропроцесора і особливості його реалізації.
7. Як завантажувати резидентні програми і драйвери у UMB ?
8. Місце основної і розширеної області даних BIOS.
9. Для чого використовується операція "затінення" BIOS ?
10. Як прочитати байт енергонезалежної пам'яті ?
11. Структура заголовка пристрою при установці драйвера.
12. Як вказати в заголовку що драйвер відноситься до блокового / символьного пристрою.
13. Як визначити початкову адресу заголовка 2-го драйвера.
14. Як визначити розмір 2-го блоку пам'яті.
15. Як визначити ім'я програми 2-го блоку пам'яті ?
16. Початок зовнішніх модулів ПЗП.
17. Скількома способами можна одержати лінійний адрес 0FFEF із сегментного?