Ionel Daniel MORARIU

INTERFEŢE ŞI
PROTOCOALE DE COMUNICAŢIE

Îndrumar de laborator

Editura Universităţii „Lucian Blaga” din SIBIU

2007

Referenţi ştiinţifici:

Prof. univ. dr. ing. Nicolae ŢĂPUŞ, Universitatea „Politehnica” Bucureşti

Prof. univ. dr. ing. Lucian VINŢAN, Universitatea „Lucian Blaga” din Sibiu

Tehnoredactare: Daniel MORARIU

Descrierea CIP a Bibliotecii Naţionale a României

MORARIU, IONEL DANIEL

Interfeţe şi protocoale de comunicaţie: îndrumar de laborator / Morariu Ionel Daniel - Sibiu

Editura Universităţii “Lucian Blaga” din Sibiu, 2007

Bibliogr.

ISBN 978-973-739-447-7

004.35

CUPRINS

Introducere

Alături de procesor, sistemele de intrare ieşire reprezintă o parte importantă a unui calculator. Interacţiunea omului cu calculatorul are loc prin intermediul acestor sisteme de intrare ieşire, astfel că performanţele sistemului de calcul depind de relaţia dintre procesor şi memorie, pe de o parte şi dispozitivele de intrare ieşire, pe de altă parte. Deşi în ultimul timp viteaza de lucru a procesoarelor a crescut extraordinar de mult instrucţiunile vor fi executate tot la rata la care acestea sunt furnizate din memorie sau citite de pe unităţile de disc. Pe lângă faptul că unele periferice au un rol important asupra vitezei de lucru a unui calculator, altele permit conectarea calculatorului la diferite procese tehnologice industriale. Sunt tot mai dese cazurile în care calculatorul este folosit la supravegherea sau controlul proceselor tehnologice iar comunicaţia între calculator şi proces se realizează prin intermediul interfeţelor standard de care acesta dispune. De aceea această carte îşi propune să prezinte posibilităţi de a dezvolta aplicaţii şi a comunica cu astfel de procese prin descrierea resurselor şi a protocoalelor interfeţelor disponibile pe un PC. Această carte se doreşte a fi un ghid util pentru cei care vor să dezvolte aplicaţii care să comande diferite dispozitive sau procese folosind calculatorul.

Cartea de faţă este destinată în principal a fi un îndrumar de laborator la disciplina „Interfeţe şi protocoale de comunicaţie” existentă în planul de învăţământ al specializării Calculatore şi Automatizări. Nu se încearcă prezentarea în această carte a tuturor perifericelor existente care pot fi conectate la un calculator ci se prezintă doar câteva care sunt foarte utilizate, cartea propunându-şi prezentarea interfeţelor standard şi a protocoalelor folosit de acestea pentru a fi posibilă dezvoltarea de noi periferice şi comunicarea cu acestea. La finele fiecărui capitol sunt propuse spre rezolvare câteva probleme care necesită utilizarea interfeţei prezentate.

Cartea este organizată pe 9 capitole. În primul capitol se prezintă facilităţile oferite de mediile de programare de nivel înalt (Visual C, C++ Builder, Borland C++), în dezvoltarea de aplicaţii pentru comanda perifericelor. În acest capitol sunt abordate mediile de programare uzuale care pot fi utilizate pentru programarea perifericelor. Nu vor fi prezentate aici mediile de programare dedicate, existente, care maschează înţelegerea la nivel hardware a modului de funcţionare a acestor interfeţe şi care vor face obiectul unei alte cărţi. Capitolul 2 este dedicat prezentării sistemului video (şi mai ales a plăcii video) precum şi

modulului low-level de programare a acestuia. În capitolul 3 se prezintă tastatura cu modul ei de funcţionare ca şi dispozitiv de intrare şi de ieşire precum şi resursele hardware disponibile. În capitolul 4 se ia în vizor unităţile de disc ca şi unele dintre cele mai utilizate periferice şi se prezintă modul de organizare a informaţilor şi modul de acces la acestea folosind serviciile BIOS urmând ca în capitolul 5 să se prezinte modul fizic de acces la aceste informaţii şi comenzile interfeţei ATA. În capitolul 6 se prezintă caracteristicile comunicaţiei seriale, în special standardul RS232 existent pe calculator, precum şi protocolul de comunicaţie şi resursele hardware puse la dispoziţie pentru a comunica pe interfaţa serială. Capitolul 7 este dedicat prezentării structurilor cartelelor electronice, a modului de citire al acestora şi a posibilităţii de conectare a acestora la un calculator. Caracteristicile de comunicaţie ale portului paralel precum şi regiştrii de comunicaţie sunt prezentaţi în capitolul 8. De asemenea se prezintă toate tipurile de porturi paralele existente şi modul de dezvoltare de aplicaţii pentru acestea. În capitolul 9 se prezintă magistralele existente în interiorul calculatorului precum şi modul de dezvoltare de aplicaţii pentru a lucra cu plăcile de extensie conectate prin aceste magistrale. În anexe sunt prezentate câteva aplicaţii simple dezvoltate în limbajul C++ care exemplifică posibilităţi de programare şi accesare a câtorva perifericelor.

Sperând că această carte va fi utilă cititorilor, autorul aşteaptă comentarii şi sugestii privind îmbunătăţirea conţinutului cărţii pe adresa:

daniel.morariu@ulbsibiu.ro

Daniel MORARIU

1 Medii de programare

1.1 Scopul lucrării

Lucrarea îşi propune familiarizarea studenţilor cu funcţiile puse la dispoziţie de diferitele medii de programare pentru dezvoltarea de aplicaţii care să permită lucrul cu diferite dispozitive conectate la calculator. Se încearcă familiarizarea studenţilor cu dezvoltarea acestor tipuri de aplicaţii.

1.2 Consideraţii generale

O dată cu apariţia sistemelor de operare moderne au fost restricţionate o mare parte din metodele clasice de accesare şi de lucru cu perifericele. În acest capitol sunt prezentate atât o parte din metodele vechi pentru lucrul cu perifericele disponibile numai sub sistemele de operare MS-DOS, Windows 95/98/Me care nu sunt aşa de restrictive, din punct de vedere al accesului la porturi, precum şi metodele puse la dispoziţie în sistemele de operare bazate pe tehnologia NT.

1.3 Comanda perifericelor sub sistemul de operare MS-DOS

1.3.1 Facilităţi oferite de limbajul Borland C++ 3.1 pentru lucrul cu perifericele

1.3.1.1 Folosirea asamblării direct din programul C

Limbajul C oferă posibilitatea programării mixte atât în limbajul C cât şi în limbajul de asamblare, astfel beneficiind atât de structurile de date puternice ale limbajului de nivel înalt cât şi de viteza de execuţie a programelor scrise în limbajul de asamblare.

1.3.1.2 Accesul fizic la regiştrii microprocesorului

Se poate face folosind variabilele puse la dispoziţie de mediul C:

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

Exemplu:

```
_AX = 0x1122;
```

1.3.1.3 Inserarea de instrucţiuni scrise în asamblare direct în C

Se face folosind prefixul asm.

Exemplu:

```
asm mov ax, 1122h
```

Pentru scrierea mai multor instrucţiuni succesive în limbajul de asamblare se pot folosi acoladele, la fel ca pentru orice instrucţiuni compusă.

```
asm {
     xor eax,eax
     mov dx,378h
     in al,dx
    }
```

Programele care conţin linii asm trebuie să anunţe acest lucru compilatorului prin directiva

```
#pragma inline
```

1.3.1.4 Utilizarea sistemului de întreruperi al microprocesorului 8086

Oferă posibilitatea unui apel indirect la o subrutină a cărei adresă este înscrisă într-un câmp din tabloul vectorilor de întreruperi al microprocesorului.

Funcţiile pentru utilizarea întreruperilor software sunt:

```
void int86(int nr_intr, union REGS *inregs, union REGS
           *outregs);
void int86x(int nr_intr, union REGS *inregs, union REGS
           *outregs, struct SREGS *segregs);
```

Cele două funcţii permit apelul întreruperilor software sub limbajul C oferind accesul la regiştrii procesorului imediat înaintea apelului întreruperii cât şi imediat după execuţia acesteia. Astfel parametrul *nr_intr* reprezintă numărul întreruperii care se doreşte a fi apelată, ***inregs* reprezintă adresa unei structuri care conţine valorile ce vor fi scrise în regiştrii procesorului înainte de a se apela întreruperea specificată iar ***outregs* reprezintă adresa unei structuri în care se vor scrie valorile aflate în regiştrii procesorului după execuţia întreruperii specificate. Diferenţa între **int86(...)** şi **int86x(...)** constă în faptul că int86x() permite modificarea registrelor de segment. Funcţiile încarcă registrele

microprocesorului cu valorile din *inregs*, execută întreruperea software specificată, şi întorc în *outregs* valorile registrelor după execuţie.

Transferul de parametrii prin registrele microprocesorului către subrutina de servire a întreruperii se poate face prin următoarele structuri de date care sunt declarate în biblioteca **dos.h.**

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

Structura **BYTEREGS** defineşte variabilele care fac legătura cu regiştrii procesorului cu acces pe octet iar structura **WORDREGS** defineşte variabilele care fac legătura cu regiştrii cu acces pe cuvânt ai procesorului. Structura **REGS** folosită în cadrul funcţiilor int86(...) şi int86x(...) ca fiind *union* defineşte variabile care fac legătura atât cu regiştrii de acces pe cuvânt cât şi pe octet ai procesorului. Specificarea structurii REGS ca şi *union* oferă posibilitatea ca cele două variabile definite de tip BYTEREGS şi WORDREGS să folosească acelaşi spaţiu de memorie atât pentru definirea variabilelor pe cuvânt cât şi a celor pe octet.

Un exemplu simplu care foloseşte structurile definite şi funcţia **int86** pentru apelul întreruperii INT 21h subfuncţia 01h (citirea unui caracter de la tastatură) este:

```
#include <dos.h>
union REGS r_in,r_out;
void main(){
   r_in.h.ah=0x01;
   int86(0x21,&r_in, &r_out);
}
```

1.3.1.5 Instalarea handler-elor utilizator

În ceea ce priveşte implementarea de noi rutine de tratare a întreruperilor hardware şi software şi instalarea acestora în locul celor existente (instalarea handler-elor utilizator), limbajul C pune la dispoziţie următoarele funcţii, echivalente celor din limbajul de asamblare:

```
#ifdef __cplusplus
   #define __CPPARGS ...
#else
   #define __CPPARGS
#endif

void interrupt(*old_intr)(__CPPARGS);
void interrupt newIntr(__CPPARGS)
{...
}
```

Această funcţie reprezintă echivalentul C al funcţiei 35h a întreruperii 21h. Prima instrucţiune defineşte o variabilă de tip interrupt numită *old_intr* în care se poate salva adresa vechii rutine de tratare a întreruperii pentru a permite restaurarea ulterioară a acestei. Ce-a de-a doua instrucţiune reprezintă declararea unei nou rutine de tratare a întreruperii. Definirea constantei „__CPPARGS” care este folosită ca şi parametru la cele două instrucţiuni este necesară pentru a putea face diferenţa (la nivel de compilator) între definirea unei întreruperi în C şi a unei întreruperi în C++.

Adresa unei rutine de tratare a întreruperii poate fi preluată folosind funcţia *getvect* cu prototipul:

```
old_intr = getvect(nr_intr);
```

Funcţia va fi apelată cu un întreg *nr_intr* reprezentând indexul întreruperii la care se doreşte preluarea vectorului de întrerupere şi va întoarce adresa rutinei de tratare a întreruperii respective, văzută ca o funcţie de tip interrupt care va fi salvată în *old_intr*.

Un exemplu care citeşte adresa rutinei de tratare a întreruperii 21h şi o salvează într-o variabilă:

```
void interrupt(*old_21h)(...);
      //declaraţie – pointer la o funcţie de tip întrerupt
……
old_21h = getvect(0x21);
      // citirea vectorului de întrerupea
```

Această funcţie este folosită de obicei pentru salvarea adresei rutinei de tratare a unei întreruperi înainte de a indirecta întreruperea respectivă către noua rutină proprie. Salvarea permite apelul vechii rutine direct din noua rutină proprie şi restaurarea vechiului vector dacă este cazul şi dacă restaurarea mai poate fi posibilă.

Pentru scrierea în tabela vectorilor de întreruperi a noii adrese a rutinei de tratare întreruperii se poate folosi funcţia *setvect* cu prototipul:

```
void setvect(int nr_intr, void newIntr);
```

Această funcţie este echivalentă funcţiei 35h a întreruperii 21h. Funcţia este apelată cu un parametru întreg *nr_intr* ce reprezintă indexul întreruperii ce se doreşte indirectată şi cu un al doilea parametru *newIntr* reprezentând adresa noii rutine de tratare a întreruperii respective, văzută ca o funcţie de tip interrupt.

Exemplu: pentru a indirecta întreruperea 21h se pot folosi următoarele linii de program:

```
void interrupt new_21h(…)
     {
     //corp funcţie noua întrerupere
     }
void main(void)
{
 ...
    setvect(0x21,new_21h);
}
```

1.3.1.6 Funcţii de tip „interrupt”

Definirea unei funcţii ca fiind de tip interrupt are ca efect următoarele:

1. La intrare în funcţie se vor executa automat următoarele instrucţiuni din limbajul de asamblare:

```
push ax, bx, cx, dx, es, ds, si, di, bp
mov bp,DGROUP
mov ds,bp     ;poziţionează de pe zona de date a
              ;programului
mov bp,sp     ;permite accesarea ulterioară a
              ;regiştrilor salvaţi
```

Obs. Ultima linie apare doar dacă programul este compilat cu opţiunea “Standard Stake Frame” activă.

2. La ieşirea din funcţie se vor executa următoarele instrucţiuni:

```
pop bp, di, si, ds, es, dx, cx, bx, ax
iret
```

Obs: Datorită restaurării din stivă a tuturor regiştrilor implicaţi în întoarcerea de parametrii de către anumite întreruperi, în forma prezentată nu se pot întoarce valori valide folosind funcţii de tip **interrupt**. Ca urmare, se poate deduce că folosirea funcţiilor de tip întterrupt este utilă la indirectarea întreruperilor hardware sau a întreruperilor software care nu întorc parametri. Într-o astfel de situaţie, relativ des întâlnită, conţinutul unei funcţii de tip interrupt poate arăta la fel ca în exemplul următor:

```
void interrupt new_21(…)
{
      old_21h();   //apelul vechii rutine de tratare
                   //a întreruperii
                   //operaţii de salvare a datelor în
                   //variabile program
}
```

1.3.1.7 Funcţii pentru scrierea/citirea în/din porturi

Pentru scrierea în port limbajul C pune la dispoziţie următoarele funcţii definite în biblioteca **dos.h** şi în biblioteca **conio.h:**

```
//în biblioteca dos.h
void outportb ( unsigned portID, unsigned char val);
void outport  ( unsigned portID, unsigned      val);

//în biblioteca conio.h
unsigned outpw( unsigned portID, unsigned int  val);
int      outp ( unsigned portID, int           val);
```

Funcţia **outportb** permite scrierea unui octet (specificat în parametrul „val”) la adresa de port specificată prin parametrul „portID”. Funcţia **outport** permite scrierea un cuvânt specificat în parametrul „val” la adresa de port specificată în parametrul „portID”. Instrucţiunile **outpw** şi **outp** sunt macrouri care permit scrierea la portul specificat prin „portID” a unei valori specificate în „val”.

Pentru citirea dintr-un port în limbajul C se pot folosi următoarele instrucţiuni:

```
//în biblioteca dos.h
unsigned char inportb( unsigned  portID);
unsigned      inport ( unsigned  portID);

//în biblioteca conio.h
unsigned inpw (unsigned portID);
int      inp  (unsigned portID);
```

Funcţia **inportb** permite citirea unui octet de la adresa de port specificată ca şi parametru al funcţiei „portID”. Funcţia **inport** permite citirea uni cuvânt de la adresa de port specificată.

Un exemplu simplu care permite trimiterea de semnale direct la difuzorul (speaker-ul) calculatorului prin scrierea în portul 61h (bitul 0 şi 1) activând sau dezactivând producerea de sunete cu o anumită frecventă pe speaker. Frecventa semnalului este scrisă în portul 42h după ce s-a specificat în portul 43h valoarea b6h.

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#define ESC 27

void main()
{ int gata = 1,frecventa = 150, divizor;
  unsigned short value;
     while(gata){
       switch(getch()){
          case ESC: gata = 0;break;
          case '+': frecventa++; break;
          case '-': frecventa--; break;
       }
       if (frecventa == 0)
          frecventa = 1;
          //specificare frecvenţă
       divizor =  1193180/ frecventa;
       outport(0x43, 0xb6);  // programare timer 2
       outport(0x42, (short)( divizor % 256));
       outport(0x42, (short)( divizor / 256));
          //validare speaker
       value = inport(0x61);
       value = value | 0x03; // activare speaker
       outport(0x61, value);
     }
     value = inport(0x61);
     value = value & (0xFF - 0x03);  // dezactivare -
     outport(0x61, value);           //speaker
}
```

1.3.1.8 Funcţii pentru accesul direct la memorie

Limbajul C prin intermediul bibliotecii **dos.h** pune la dispoziţie funcţii care permit scrierea şi citirea unei anumite zone de memorie specificată prin adresa de segment şi adresa de offset. Astfel pentru citirea unui octet de la o adresă

specificată de memorie există funcţia **peekb** iar pentru citirea unui cuvânt de la o adresă de memorie există funcţia **peek**. Prototipurile celor două funcţii sunt:

```
int  peek (unsigned adrSegment, unsigned adrOffset);
char peekb(unsigned adrSegment, unsigned adrOffset);
```

Adresa de la care se doreşte citirea este specificată prin adresă de segment (parametrul adrSegment) şi prin adresă de offset (parametrul adrOffset).

Pentru scrierea la o adresă specificată de memorie a unui cuvânt sau a unui octet sunt puse la dispoziţie funcţiile **poke** şi **pokeb** definite în biblioteca **dos.h**.

```
void poke (unsigned adrSegment, unsigned adrOffset, int
           value);
void pokeb(unsigned adrSegment, unsigned adrOffset, char
           value);
```

Funcţia **poke** permite scrierea unui cuvânt iar funcţia **pokeb** permite scrierea unui octet specificat prin parametrul „value” la adresa de segment „adrSegment” şi la adresa de offset „adrOffser”.

De asemenea limbajul C mai pune la dispoziţie macrouri care permit definirea de variabile de tip pointer spre o anumită zonă de memorie specificată. De altfel pune la dispoziţie macrouri care permit citirea adresei de segment şi a adresei de offset la care să găseşte o anumite variabilă.

```
void far *MK_FP(unsigned adrSeg, unsigned adrOff);
unsigned FP_SEG(void far *p);
unsigned FP_OFF(void far *p);
```

Macroul MK_FP permite crearea unei variabile care pointează spre zona de memorie specificată prin adresa de segment „adrSeg” şi adresa de offset „adrOff”. Folosind această variabilă de tip pointer se poate scrie şi citi zona respectivă de memorie. Macroul FP_SEG permite citirea adresei de segment în care se găseşte definită variabila specificată ca şi parametru. Macroul **FP_OFF** permite citirea adresei de offset la care se găseşte definită variabila specificată ca şi parametru.

1.4 Accesul la interfeţe sub sistemele de operare Windows NT/2000/XP

Aceste sisteme de operare, spre deosebire de sistemele Windows 95/98/Me, utilizează nivele de privilegii ale procesorului. Nivelul la porturile de I/O este controlat de nivelul privilegiilor de I/O (IOPL –Input/Output Privilege Level) din registrul indicatorilor de condiţii şi de harta drepturilor de I/O din segmentul

de stare al procesorului TSS (Task State Segment). Astfel de câte ori se încearcă accesul la un port de I/O dintr-un program utilizator obişnuit se va genera o excepţie de instrucţiune privilegiată.

Procesorul recunoaşte patru nivele de privilegiu al programelor (de la 0 la 3). Nivelul 0, cel mai privilegiat reprezintă nivelul nucleu. Nivelul 3, nivelul cel mai puţin privilegiat, reprezintă nivelul programelor utilizator. Cei doi biţi IOPL din registrul indicatorilor de condiţii indică nivelul de privilegiu pe care trebuie să îl aibă un proces pentru a putea executa instrucţiunile privilegiate, asemenea instrucţiuni fiind şi cele de scriere / citire din porturi.

Harta drepturilor de I/O din segmentul de stare al procesorului conţine câte un bit pentru fiecare adresă de I/O şi poate fi utilizată pentru a permite programelor care nu au un nivel de privilegiu suficient de mare să poată accesa porturile de I/O. Dacă bitul corespunzător unei adrese de I/O este setat, o instrucţiune de I/O cu acea adresă va genera o excepţie, iar în caz contrar operaţia de I/O va fi permisă. Procesorul testează drepturile de I/O atunci când se execută o instrucţiune de I/O într-un proces, iar procesul respectiv nu are un nivel de privilegiu suficient pentru execuţia instrucţiunii.

O altă problemă care apare în limbajele de nivel înalt pentru accesul la porturi este aceea că aceste limbaje nu mai pun la dispoziţie funcţii pentru accesul direct la porturile de I/O, funcţii care în variantele mai vechi ale acestor limbaje erau disponibile (funcţiile *inport* şi *outport* prezentate mai sus).

Există două soluţii pentru accesul la porturile de I/O sub sistemele de operare Windows NT/2000/XP. Prima este de a se utiliza un driver de dispozitiv care rulează cu nivelul de privilegiu 0, driver care să permită accesul la porturi. Datele sunt transferate între un program utilizator şi driver prin intermediul apelurilor funcţiei sistem DeviceIOControl, operaţia care trebuie executată de driver fiind indicată printr-un cod de control IOCTL (I/O Control). Pentru simplificarea programelor utilizator, driverul poate pune la dispoziţie şi unele funcţii de I/O similare cu funcţiile *inport* şi *outport* puse la dispoziţie în trecut de mediile de dezvoltare în limbajul C. De exemplu, aceste funcţii pot fi furnizate sub forma unor biblioteci DLL (Dynamic Link Library). Astfel, nu va mai fi necesar apelul direct al funcţiei sistem DeviceIoControl.

Această soluţie este cea recomandată pentru accesul la porturi. Totuşi, dezavantajul utilizării unui asemenea driver de dispozitiv este că eficienţa transferurilor de date va fi redusă. La fiecare apel al unei funcţii pentru citirea sau scrierea unui octet, procesorul trebuie să comute de la execuţia cu nivelul de

privilegiu 3 la cea cu nivelul de privilegiu 0, iar după execuţia operaţiei să se realizeze comutarea inversă. Însă, driverul poate pune la dispoziţie şi funcţii pentru citirea şi scrierea unui bloc de date, în locul citirii şi scrierii unui singur octet. A doua soluţie pentru accesul la porturile de I/O constă în modificarea hărţii drepturilor de I/O pentru a permite unui anumit proces accesul la unele porturi. Deşi această metodă nu este recomandată, ea permite rularea unor aplicaţii existente sub sistemele de operare Windows NT/2000/XP.

1.4.1 Driverul „NTPort Library”

Există mai multe drivere disponibile pentru accesul la porturile I/O sub sistemele de operare bazate pe tehnologia NT. În cadrul acestui laborator se vor prezenta două dintre acesta. Driver-ul NTPortLibrary pus la dispoziţie de firma „Yeal Soft Studo” care oferă librăriile şi fişierele de interfaţă pentru mai multe limbaje de nivel înalt cum ar fi Visual C++, C++ Builder, Visual Basic şi Delphy. Pentru aceste limbaje de programare de nivel înalt se pune la dispoziţie funcţii pentru activarea sau dezactivarea anumitor porturi precum şi funcţii pentru citirea unui octet / cuvânt / dublu-cuvânt de la un port de intrare şi scrierea unui octet / cuvânt la un port de ieşire.

Acest driver necesită o instalare în prealabil deoarece are nevoie de înregistrarea unor anumite dll-uri în sistemul de operare.

În continuare vom prezenta doar modul de utilizare al acestui driver pentru limbajul Borland C++ Builder. Pentru utilizarea funcţiilor de acces la porturi (Inport, Outport) in aplicaţii create în acest limbaj sunt necesare următori paşi:

1. Se creează un proiect pentru aplicaţia care se doreşte realizată.
2. Se copiază în folderul proiectului fişierele „ntport.h” şi „bcbport.lib” din folderul unde este instalat driverer-ul (de obicei „c:\Program Files\NTPortLibrary”).
3. În proiectul astfel creat se include fişierul „bcbport.lib”. Includerea acestui fişier se face accesând meniul „Project -> Add To project...”. Pentru a putea include acest fişier în fereastra de dialog care apare se selectează tipul de fişiere cu extensia „lib”. Fişierul „bcbport.lib” va fi copiat în prealabil în directorul în care se găseşte proiectul.
4. În fişierul cpp în care se doreşte utilizarea funcţiilor de acces la porturi se include fişierul „ntport.h” pus la dispoziţie de driverul de acces la porturi.

Acest fişier conţine prototipurile funcţiilor care vor putea fi utilizate pentru accesul la porturile de intrare ieşire. Şi acest fişier va fi copiat în prealabil în directorul în care se găseşte proiectul.

În fişierul „ntport.h” sunt definite mai multe prototipuri de funcţii de intrare ieşire astfel:

```
//pentru operaţii de citire din port
WORD  Inp     (WORD PortAddress);   //citeşte un octet
WORD  Inport  (WORD PortAddress);   //citeşte un octet
WORD  Inpw    (WORD PortAddress);   //citeşte un cuvânt
WORD  InportW (WORD PortAddress);   //citeşte un cuvânt
DWORD Inpd    (WORD PortAddress);   //citeşte un dublu-
                                    //cuvânt
DWORD InportD (WORD PortAddress);   //citeşte un dublu-
                                    //cuvânt

// pentru operaţii de scriere în port
void Outp     (WORD PortAddress, WORD Data);  //octet
void Outport  (WORD PortAddress, WORD Data);  //octet
void Outpw    (WORD PortAddress, WORD Data);  //cuvânt
void OutportW (WORD PortAddress, WORD Data);  //cuvînt
void Outpd    (WORD PortAddress, DWORD Data); //dublu-
                                              //cuvânt
void OutportD (WORD PortAddress, DWORD Data); //dublu-
                                              //cuvânt
```

Funcţiile sunt asemănătoare cu cele definite în vechia versiune pentru MS-DOS a limbajului C. Aceste instrucţiuni permit citirea unui octet, cuvânt sau a unui dublu-cuvânt de la adresa de port specificată ca şi parametru prin „PortAddress” precum şi scrierea a unui octet, cuvânt sau dublu cuvânt specificat în parametrul „Data” la adresa de port specificată prin „PortAddress”.

Un exemplu simplu care permite trimiterea de semnale direct la difuzorul (speaker-ul) calculatorului prin scrierea în portul 61h (bitul 1 şi 2) activând sau dezactivând producerea de sunete cu o anumită frecventă pe speaker. Frecventa semnalului este scrisă în portul 42h după ce s-a specificat în portul 43h valoarea b6h.

Un exemplu simplu care scriind în portul 61h (bitul 1 şi 2) activează sau dezactivează producerea de semnale direct pe difuzorul (speaker-ul) calculatorului folosind funcţiile definite de biblioteca NTPort pentru scrierea şi citire din porturi. Frecventa semnalului este scrisă în portul 42h după ce s-a specificat în portul 43h valoarea 0b6h. Se prezintă interfaţa pentru această aplicaţii şi codul sursă din fişierul „Speacer_unit.cpp” din acest proiect.


```
//                    Speaker_unit.cpp
//---------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include „Speaker_unit.h”
#include „ntport.h”

//---------------------------------------------------
#pragma package(smart_init)
#pragma resource „*.dfm”
TForm1 *Form1;

//---------------------------------------------------
__fastcall Tform1::Tform1(Tcomponent* Owner) :
           Tform(Owner)
{
}

//---------------------------------------------------
void __fastcall Tform1::btnSoundClick(Tobject *Sender)
{
    int divizor =  1193180 / StrToInt(valFrecventa->
                                      Caption);
        // timer 2, square wave
    Outport(0x43, 0xb6);
    Outport(0x42, (short)( divizor % 256));
    Outport(0x42, (short)( divizor / 256));
        //validare speaker
    short value = Inport(0x61);
    value = value | 0x03;       // activare speaker
    Outport(0x61, value);
}

//---------------------------------------------------
void __fastcall TForm1::btnNoSoundClick(TObject *Sender){
    short value = Inport(0x61);
    value = value & (0xFF - 0x03);  // dezactivare speaker
    Outport(0x61, value);
```

```
}

//---------------------------------------------------
void __fastcall TForm1::btnIncreaseFreqClick(TObject
                *Sender)
{
     int value = StrToInt(valFrecventa->Caption);
     value ++;
     valFrecventa->Caption = IntToStr(value);
}

//---------------------------------------------------
void __fastcall TForm1::btnDescFreqClick(TObject *Sender)
{
        int value = StrToInt(valFrecventa->Caption);
        value--;
        if (value == 0) value = 1;
        valFrecventa->Caption = IntToStr(value);
}
```

1.4.2 Driverul PortTalk

Este un alt driver care permite accesul la porturile de I/O. Acesta nu necesită instalare şi pune la dispoziţie doar fişierele de interfaţă pentru mediul de dezvoltare Microsoft Visual C++ 6.0. Acest driver pune la dispoziţie doar funcţii pentru citirea / scrierea în port pe octet şi pe cuvânt. Pe lângă aceste funcţii, driverul mai dispune de funcţii pentru modificare hărţii drepturilor de I/O ale unui proces. Pachetul conţine, în plus un program „AllowIo.exe" care permite unui program existent (Dos sau Windows) accesul la porturile ale căror adrese sunt specificate ca parametrii în linia de comandă.

Pentru utilizarea funcţiilor de acces la porturi (inport, outport) in Visual C++ sunt necesare următoarele operaţii:

1. Se copiază fişierele PortTalk_IOCTL.c şi PortTalk_IOCTL.h în directorul în care se găseşte proiectul aplicaţiei.

2. Se include în fişierul sursă al aplicaţie fişierul „PortTalk_IOCTL.h" folosind linia:

```
#include „PortTalk_IOCTL.h"
```

3. Înainte de utilizarea funcţiilor de I/O se apelează funcţia OpenPortTalk();

4. La sfârşitul aplicaţiei se apelează funcţia ClosePortTalk();

În fişierul PortTalk_IOCTL.c funcţiile inportb şi outportb sunt definite astfel:

```
void outportb(unsigned short PortAddress, unsigned char
              Data);
unsigned char inportb(unsigned short PortAddress);
```

Fişierele PortTalk_IOCTL.c şi PortTalk_IOCTL.h nu trebuiesc incluse în proiectul aplicaţiei, ci doar copiate în directorul proiectului.

Programul AllowIo.exe, care permite modificarea hărţii drepturilor de I/O pentru programele existente, poate fi lansat în execuţie din linie de comandă. Ca şi parametri se introduc numele programului executabil pentru care se solicită accesul la porturi, urmat de lista porturilor de I/O la care se doreşte accesul. Prin specificarea unei adrese de I/O se va acorda accesul la 8 porturi consecutive începând de la adresa respectivă. De exemplu, pentru accesul programului Port.exe la toate registrele portului paralel LPT1 se va introduce comanda:

allowio.exe Port.exe 0x378

Pentru accesul la toate porturile de I/E se poate specifica opţiunea /a.

2 Sistemul Video

2.1 Scopul lucrării

Lucrarea are ca scop familiarizarea studenţilor cu tipurile de adaptoare video existente şi cu modul de funcţionare al acestora. Se prezintă atât funcţiile puse la dispoziţie pentru utilizarea acestor adaptoare în modul text cât şi funcţiile pentru utilizarea acestora în modul grafic.

2.2 Consideraţii generale

Sistemul video este compus în principal din două părţi:

- un monitor (display), folosit pentru reprezentarea informaţiilor de tip text şi grafic
- un adaptor video – reprezentat printr-un dispozitiv pentru conversia semnalelor digitale transmise de procesor în semnale pentru controlul monitorului.

2.3 Tipuri de adaptoare video

MDA - Monochrome Display Adapter – cel mai vechi adaptor, permite numai modul text 80*25 (80 coloane şi 25 linii);

CGA – Color Graphics Adapter – permite lucrul cu 4 culori principale, modul text 80*25 şi 40*25 precum şi modul grafic cu rezoluţia 640*200;

HGC – Hercules Graphics Card – permite lucru cu înaltă rezoluţie în mod text şi în mod grafic o rezolzţie de 640*350 şi 16 culori;

VGA – Video Graphics Array – permite folosirea a maximum 256 culori şi o rezoluţie de 640*480 în modul grafic;

Super VGA – permite o rezoluţie de până la 1024*768 şi a 16 biţi pe culoare;

XGA – eXtended Graphics Adapter.

2.4 Serviciile BIOS referitoare la seturile de caractere din modurile alfanumerice

Un caracter este definit în memoria video printr-o matrice de pixeli, ale cărui dimensiuni depind de modul alfanumeric curent. Dimensiunile standard de caractere sunt 8*8, 8*14, 8*16, 9*14, 9*16.

Serviciile BIOS au scopul de a încărca seturi de caractere standard sau definite de utilizator. Programatorul poate încărca un set de caractere în două moduri:

- folosind una din funcţiile 0h..4h, funcţie care execută automat şi un apel al funcţiei de setare a modului alfanumeric corespunzător, fără a şterge conţinutul memoriei video.
- Folosind una din funcţiile 10h..14h, după care programatorul trebuie să apeleze funcţia de setare a modului alfanumeric, iar pagina 0 de memorie video trebuie să fie activă.

Serviciile referitoare la seturile de caractere sunt subfuncţii ale funcţiei 11h a întreruperii **10h**.

Subfuncţia 00h:- încarcă în memorie, în mod text un set de caractere definit de utilizator.

AH=11h (funcţia)

AL=00h (subfuncţia)

BL-numărul blocului în care se va încărca setul de caractere (0..4)

BH – numărul de octeţi pe caracter (max. 32)

CX – numărul de caractere de încărcat

DX – ofsetul primului caracter în blocul de memorie selectat

ES:BP – pointer la setul de caractere ce se doreşte încărcat

Întoarce – nimic.

Subfuncţia 01h:- încărcarea setului monocrom 9*14 din ROM.

AH=11h

AL=01h

BL-numărul blocului în care se va încărca setul de caractere (0..4)

Întoarce – nimic.

Subfuncţia 02h:- încărcarea setului 8*8 din ROM.

AH=11h

AL=02h

BL-numărul blocului în care se va încărca setul de caractere (0..4)

Întoarce – nimic.

Subfuncţia 03h:- setarea blocului activ (setului de caractere).

AH=11h

AL=03h

BL – valoarea ce va fi încărcată în registrul de selectare a setului de caractere activ.

Întoarce – nimic.

Subfuncţia 04h:- încărcarea setului monocrom 8*16 din ROM.

AH=11h

AL=04h

BL-numărul blocului în care se va încărca setul de caractere (0..4)

Întoarce – nimic.

Subfuncţia 10h:- încărcarea în memorie a unui set de caractere definit de utilizator.

AH=11h

AL=10h

BL – numărul blocului în care se va încărca setul de caractere (0..4)

BH – numărul de octeţi pe caracter (maxim 32)

CX – numărul de caractere de încărcat

DX – ofsetul primului caracter în blocul de memorie selectat

ES:BP – pointer la setul de caractere ce se doreşte încărcat

Întoarce – nimic.

Subfuncţia 11h:- încărcarea setului monocrom 9*14 din ROM.

AH=11h

AL=11h

BL – numărul blocului în care se va încărca setul de caractere (0..4)

Întoarce – nimic.

Subfuncţia 12h:- încărcarea setului monocrom 8*8 din ROM.

AH=11h

AL=12h

BL – numărul blocului în care se va încărca setul de caractere (0..4)

Întoarce – nimic.

Subfuncţia 14h:- încărcarea setului monocrom 8*16 din ROM.

AH=11h

AL=14h

BL – numărul blocului în care se va încărca setul de caractere (0..4)

Întoarce – nimic.

Subfuncţia 30h:- citirea informaţiilor despre seturile de caractere existente în ROM.

AH=11h

AL=30h

BL - numărul setului de caractere

0 – partea superioară a setului de caractere 8*8 definit de utilizator (mod grafic)

1 – partea inferioară a setului de caractere 8*8 definit de utilizator (mod grafic)

2 – setul de caractere 8*14 din ROM

3 – partea inferioară a setului de caractere 8*8 din ROM

4 – partea superioară a setului de caractere 8*8 din ROM

5 – setul de caractere 9*14

6 – setul de caractere 8*16

7 – setul de caractere 9* 16

Întoarce:

CX – numărul de octeţi /caracter din setul de caractere selectat

DL – numărul de rânduri de caractere pe ecran

ES:BP – adresa din memorie a setului de caractere specificat.

2.5 Sistemul video – modul grafic

Modurile grafice standard recunoscute de adaptorul superVGA sunt:

Adaptor	**Mod grafic**	**Număr culori**	**Număr pagină**	**Adresa de bază**	**Rezoluţie**
04h	CGA	4	1	B8000h	320*200
05h	CGA	4	1	B8000h	320*200
06h	CGA	2	2	B80000h	640*200
0Dh	EGA	16	8	A0000h	320*200
0Eh	EGA	16	4	A0000h	640*350
0Fh	EGA	4	1	A0000h	640*350
10h	EGA	16	2	A0000h	640*350
11h	VGA	2	1	A0000h	640*480
12h	VGA	16	1	A0000h	640*480
13h	VGA	256	1	A0000h	320*200
	SVGA	256	1	A0000h	640*480

Memoria video poate fi scrisă sau citită atât cu ajutorul întreruperilor (rutinelor) BIOS cât şi prin scrierea direct în memoria video, prin instrucţiuni de scriere directă (MOV). Folosirea rutinelor BIOS este simplă însă foarte înceată în comparaţie cu accesarea directă.

2.6 Serviciile BIOS pentru lucrul cu modurile grafice – INT 10h

AH=00h – selectarea modului video. Selectează modul video şi şterge ecranul, schimbând şi dimensiunea cursorului. Pentru modul grafic cursorul nu este afişat.

AL = codul modului video.

Întoarce: nimic.

AH=0Ch – scrierea unui element de imagine. Scrie un punct de o culoare specificată la o coordonată ecran specificată.

AL = valoarea pentru culoarea elementului de imagine.

CX = încărcat cu coordonata orizontală a elementului de imagine.

DX = încărcat cu coordonata verticală a elementului de imagine.

BH = numărul paginii video pentru moduri grafice cu mai mult de o pagină.

Întoarce: nimic.

AH=0Dh – citirea unui element de imagine. Întoarce culoarea unui element de imagine aflat la o poziţie ecran specificată.

CX = încărcat cu coordonata orizontală a elementului de imagine.

DX = încărcat cu coordonata verticală a elementului de imagine.

BH = numărul paginii video pentru moduri grafice cu mai mult de o pagină.

Întoarce:

AL = valoarea pentru culoarea elementului de imagine.

AH=0Eh – scrierea unui caracter în mod TTY (TeleType). Scrie un caracter la poziţia curentă a cursorului şi avansează cursorul.

AL = caracterul de scris.

BL = culoarea caracterului pentru modul grafic.

BH = numărul paginii video pentru moduri grafice cu mai mult de o pagină.

Întoarce: nimic.

AH=0Fh – obţinerea modului video curent. Întoarce numărul modului video curent şi anumite informaţii despre acesta.

Întoarce:

AL = numărul modului video;

AH = numărul de caractere pe rând;

BH = numărul paginii video active.

2.7 Probleme propuse

Problema 1. Să se scrie un program în C care încarcă şi activează setul de caractere monocrom 8*8.

Problema 2. Să se scrie un program C care baleiază toate seturile de caractere din ROM la apăsarea tastei „+” crescător de la subfuncţia 0h la subfuncţia 14h şi descrescător la apăsarea tastei „-”. Programul va fi părăsit la apăsarea tastei „0”.

Problema 3. Să se scrie un program care la apăsarea tastei „9” baleiază toate modurile grafice din tabelul prezentat desenând un pătrat, iar la apăsarea tastei „+” sau „-” baleiază toate culorile modului grafic curent. Se va afişa pe ecran şi modul video curent.

Problema 4. Să se scrie un program care desenează un pătrat în modul 13h, baleiind toate cele 256 de culori prin scrierea direct în memorie video. (Vezi anexa 1).

3 Gestiunea tastaturii

3.1 Scopul lucrării

Lucrarea are ca scop familiarizarea cu modul de lucrul al tastaturii. Este prezentat modul de gestiune al buffer-ului de tastatură precum şi modul de funcţionare al întreruperii hardware a tastaturii INT 09h şi a portului de tastatură.

3.2 Consideraţii generale

Există patru posibilităţii de accesare a tastaturii:

- folosind serviciile DOS ale întreruperii INT 21h
- folosind serviciile BIOS ale întreruperii INT 16h
- folosind întreruperea generată de tastatură INT 09h
- prin citirea şi scrierea direct în portul de tastatură

Serviciile întreruperi INT 21h sunt cele mai simple pentru accesarea tastaturii, dar şi cele mai lente. Serviciile BIOS constituie modul de bază de accesare a tastaturii oferind posibilitatea procesării diferitelor taste speciale cum ar fi tastele funcţionale şi tastele de control al cursorului.

Întreruperea hard 09h nu este destinata folosirii de către utilizator, ea este generată de către controlorul de tastatură la fiecare apăsare sau eliberare a unei taste. Această întrerupere poate fi totuşi rescrisă de către utilizator.

Accesarea directă a portului de tastatură constituie modul cel mai rapid dar şi cel mai dificil mod.

La fiecare apăsare a unei taste, acţiunea este anunţată BIOS-ului prin intermediul întreruperii 09h. Întreruperea 09h apelează o rutină de tratare a întreruperii care citeşte portul de tastatură (60h) pentru a prelua codul tastei apăsate. Codul (scancodul) este transferat BIOS-ului care îl translatează într-un cod de doi octeţi. Octetul mai puţin semnificativ conţine valoarea ASCII a caracterului, iar octetul mai semnificativ conţine codul scan. Tastele speciale (cele numerice şi cele funcţionale) conţin 0 în octetul mai puţin semnificativ şi

codul scan al tastei în octetul mai semnificativ. Codurile astfel translatate sunt memorate la adresa dată de valoarea din locaţia 0000:041Eh (în zona tampon a tastaturii), unde sunt păstrate până când sunt citite de un anumit program.

O facilitate a tastaturii este cea de a repeta automat codul emis la menţinerea unei taste apăsate, dacă acest timp depăşeşte o jumătate de secundă se generează automat codul tastei de zece ori pe secundă.

Controlul operării cu tastatura este realizat de BIOS, prin utilizarea zonelor de memorie rezervate în acest scop. Aceste zone pot fi utilizate pentru a citi starea tastaturii sau pentru a modifica modul de operare al acesteia. Octeţii standard rezervaţi care memorează starea tastaturii se află la adresele 417h şi 418h, semnificaţia biţilor fiind următoarea:

Octetul de stare de la adresa 417h:

Bit								**Semnificaţie**
7	6	5	4	3	2	1	0	
X								Starea tastei Insert: 1- activa, 0- inactivă
	X							Starea tastei Caps Lock: 1-activa, 0- inactivă
		X						Starea tastei Num Lock: 1-activa, 0- inactivă
			X					Starea tastei Scroll Lock: 1-activa, 0- inactivă
				X				Starea tastei Alt: 1-activa, 0- inactivă
					X			Starea tastei Ctrl: 1-activa, 0- inactivă
						X		Starea tastei Shift stânga: 1-activa, 0- inactivă
							X	Starea tastei Shift dreapta: 1-activa, 0- inactivă

Octetul de stare de la adresa 418h:

Bit								**Semnificaţie**
7	6	5	4	3	2	1	0	
X								Tasta Ins apăsata=1
	X							Tasta Caps Lock apăsată=1
		X						Tasta Num Lock apăsată=1

			X					Tasta Scroll Lock apăsată=1
				X				Tastele Ctrl-Num Lock apăsate=1
					X			1=activ semnalul de la tastatura de PcJr
						0		Nefolosit
							0	Nefolosit

3.3 Portul de tastatură – Codurile de scanare

Tastatura trimite un singur cod spre calculator numit codul de scanare (scancodul) care codifică tasta şi starea acesteia (o tastă poate avea două stări – apăsată sau eliberată). BIOS-ul realizează conversia din codul de scanare în cod ASCII. Codurile de scanare reprezintă un număr de ordine al tastei respective şi anume un întreg pe 8 biţi având valoarea cuprinsă între 1 şi numărul total de taste. La apăsarea unei taste, controlul este preluat de handler-ul întreruperii 09h, aceasta citeşte codul de scanare din portul de tastatură de la adresa 60h. Dacă o tastă este apăsată un timp mai îndelungat, acelaşi cod de scanare este generat în mod repetat până la eliberarea tastei. Codul de scanare generat la eliberarea unei taste este acelaşi cu cel de la apăsarea tastei cu excepţia bitului 7 (cel mai semnificativ bit) care este pe 1. Astfel, de exemplu, tasta „ESC” (care este prima tastă de pe tastatură) produce codul „01h” la apăsare şi codul „81h” la eliberare.

3.4 Buffer-ul de tastatură - Codurile ASCII şi codurile de scanare

Buffer-ul de tastatură este o porţiune din zona de date BIOS folosită pentru memorarea temporară a informaţiei introduse de la tastatură. La apăsarea unei taste, handler-ul întreruperii hard 09h plasează codul de scanare şi codul ASCII corespunzător tastei apăsate în acest buffer.

Buffer-ul de tastatură este caracterizat de următorii parametrii:

- *buffer start* – reprezintă adresa de început a buffer-ului în zona de date BIOS
- *buffer end* – reprezintă adresa de sfârşit a buffer-ului în zona de date BIOS

- *buffer head* – reprezintă adresa primului caracter care trebuie citit de către sistem

- *buffer tail* - reprezintă adresa locaţiei la care se va insera de către handler-ul întreruperii 09h următorul caracter

Sistemul de operare setează valorile pentru buffer start şi buffer end la încărcare şi nu le modifică în timpul lucrului.

Imediat după încărcarea sistemului de operare parametrii pentru buffer start, buffer head şi buffer tail au aceeaşi valoare iniţială.

La tastarea unei comenzi caracterele care compun comanda vor fi depuse pe câte 2 octeţi (unul pentru codul ASCII şi unul pentru codul SCAN) şi în buffer, inclusiv Enter-ul (terminatorul de comandă). Dacă sistemul de operare nu este ocupat cu execuţia altor operaţii caracterele sunt preluate din buffer iar parametrii buffer head şi buffer tail sunt setaţi la aceeaşi valoare care pointează la octetul imediat următor ultimului caracter din comandă. În cazul în care un caracter nu este preluat din buffer, parametrul buffer head este păstrat nemodificat şi pointează spre cel mai vechi caracter din buffer care nu a fost preluat încă iar parametrul buffer tail va pointa spre următoarea locaţia goală din buffer.

În zona de date BIOS se găsesc adresele parametrilor buffer-ului de tastatură care sunt:

- buffer start la adresa: 0040:0080h – de obicei 01Eh

- buffer stop la adresa: 0040:0082h – de obicei 03Eh

- buffer head la adresa: 0040:001Ah- adresa “capului” zonei tampon a tastaturii (de unde începe şirul de caractere nepreluate încă)

- buffer tail la adresa: 0040:001Ch- adresa “cozii” zonei tampon a tastaturii (unde se termină şirul de caractere)

Buffer-ul funcţionează ca o structură de tip QFIFO (coadă ciclic), aceasta însemnând că după ce ultima locaţie a fost ocupată se va înscrie automat în primul octet al buffer-ului şi ca atare parametrul buffer tail va fi iniţializat cu valoarea de start a buffer-ului.

Handlerul întreruperii 09h citeşte codurile scan de la tastatură şi le converteşte în acţiuni sau în coduri de caractere după cum urmează:

Pentru tastele ASCII când este întâlnit un “scan code” codul ASCII şi codul SCAN al tastei sunt plasate în zona tampon a tastaturii care se află la adresa 0000:041Eh şi are o dimensiune de 32 octeţi. Codurile sunt plasate în zona tampon la locaţia adresată de pointerul cozii zonei tampon. În urma acestei operaţii pointerul cozii de aşteptare este incrementat cu 2.

Dacă au fost apăsate tastele ALT, CTRL, sau SHIFT se actualizează octeţii de stare de la adresa 0000:0417h şi 0000:0418h

Dacă a fost apăsată combinaţia CTRL+ALT+DEL, octeţii de la adresa 0000:0472h sunt iniţializaţi cu valoarea 1234h şi se predă controlul rutinei de iniţializare.

3.5 Serviciile BIOS ale întreruperii INT 16h

Funcţia AH=00h - Citirea (aşteptarea) codului unei taste apăsate.

Întoarce:

AL codul ASCII al caracterului (dacă AL este zero, AH este codul ASCII extins);

AH codul scan sau un cod ASCII extins.

Funcţia AH=01h - Verificarea disponibilităţii codului unei taste apăsate

Întoarce:

ZF=1 codul nu este disponibil;

ZF=0 codul este disponibil;

AX poziţionat la fel ca funcţia 00h.

Funcţia AH=02h - Citirea stării tastei Shift. Verificarea tastelor speciale apăsate

Întoarce:

AL valorile corespunzătoare ale tastei Shift şi starea lock ca în switch-urile tastaturii.

Funcţia AH=05h - Scrie un caracter în buffer-ul de tastatură

CL = codul ASCII al caracterului;

CH = codul SCAN pentru tastele speciale (0 pentru tastele normale).

Întoarce: AL = starea (0- a reuşit scrierea, 1- buffer-ul este plin)

Funcţia AH=10h - Citeşte (aşteaptă) codul unei taste apăsate – specifică pentru tastatura cu 101 taste

Întoarce:

AL = codul ASCII al caracterului (dacă AL=0, AH se găseşte codul ASCII extins);

AH = codul SCAN sau codul ASCII extins.

Funcţia AH=11h - Verifică disponibilitatea codului unei taste, doar pentru tastatura cu 101 taste

Întoarce:

ZF = 1 codul nu este disponibil;

ZF = 0 codul este disponibil;

AX este poziţionat ca la funcţia 10h.

Funcţia AH = 12h - Citirea stării tastei SHIFT. Verificarea tastelor speciale apăsate

– doar pentru tastatura cu 101 taste.

Întoarce:

AL – valorile corespunzătoare ale tastei Shift şi starea lock ca în switch-urile tastaturii.

3.6 Probleme propuse

Problema 1: Să se scrie un program care să permită introducerea în buffer-ul de tastatură a unei comenzi pentru sistemul de operare standard utilizând serviciile oferite de întreruperea BIOS INT16h.

```
//scrie comanda „CALC” in buffer-ul de tastatură
#include<dos.h>
void main (void)
{
```

```
   union REGS r;          //declarăm o variabilă de tip REGS
   r.h.ah = 0x05;         //subfucncţia 05h a lui INT 16h
   r.h.ch = 0;            //scrie un caracter in bufferul
   r.h.cl = 'C';          // de tastatură
   int86(0x16,&r,&r);     //apel INT 16h

   r.h.ah = 0x05;
   r.h.ch = 0;
   r.h.cl = 'A';
   int86(0x16,&r,&r);

   r.h.ah = 0x05;
   r.h.ch = 0;
   r.h.cl = 'L';
   int86(0x16,&r,&r);

   r.h.ah = 0x05;
   r.h.ch = 0;
   r.h.cl = 'C';
   int86(0x16,&r,&r);

   r.h.ah = 0x05;
   r.h.ch = 0;            //codul ascii a tastei Enter este
   r.h.cl = 0x0d;         //format din 2 caractere 0x0d şi
   int86(0x16,&r,&r);     //0x0a

   r.h.ah = 0x05;
   r.h.ch = 0;
   r.h.cl = 0x0a;
   int86(0x16,&r,&r);
}
```

Problema 2: Să se scrie un program care permite introducerea unui şir de maxim 16 caractere în buffer-ul de tastatură (fără să le afişeze pe ecran) şi ulterior să afişeze şirul introdus prin citirea acestuia direct din buffer-ul de tastatura în ordinea în care a fost introdus.

Problema 3. Să se scrie un program care prin indirectarea întreruperii INT 09h să afişeze pe ecran "scan codul" tastei apăsate. (Se va realiza citirea din portul de tastatură 60h).

Obs. Întreruperea 09h pe lângă citirea din portul tastaturii (60h), realizează şi comunicarea cu controlorului de întreruperi 8259A (transmite codul EOI – End of interrupt), de aceea la interceptarea lui INT 09h trebuie apelată şi vechea rutină a întreruperii INT 09h.

4 Nivelul fizic de acces la disc

4.1 Scopul lucrării

Lucrarea are ca scop cunoaşterea principiului de organizare şi de funcţionare a unităţilor de disc precum şi a serviciilor la nivel BIOS şi fizic puse la dispoziţie de interfaţă pentru comunica cu acestea.

4.2 Servicii BIOS - INT 13h, Servicii DOS - INT 25h, INT 26h

Serviciile BIOS - disc INT 13h constituie categoria de servicii low-level pentru accesarea discului. Astfel, handler-ul acestei întreruperi conţine programe care realizează operaţii de I/O la nivel fizic, manevrând datele pe disc la nivel de sector. De fapt, orice cerere high-nivel de acces la disc va conduce eventual în cele din urmă la INT 13h (excepţie fac câteva programe extrem de complicate cum ar fi Disk Manager sau Disk Repairer care operează cu discul folosind direct comenzile controller-ului de disc).

Din punct de vedere al programatorului, organizarea fizică a discurilor este similară atât pentru floppy discuri cât şi pentru hard discuri. Toate informaţiile stocate pe disc sunt înregistrate pe *piste*, sub forma unor cercuri concentrice aflate pe suprafaţa fiecărui platan. Pistele sunt numerotate de la 0, pornind din exterior spre interior. Un disc conţine mai multe mii de piste pe fiecare platan. Fiecare pistă e divizată în unităţi mai mici numite *sectoare*, care sunt numerotate de la 1. Numărul de sectoare de pe o pistă variază în funcţie de tipul unităţii de disc. La calculatoarele compatibile IBM PC, sectoarele create prin procedura de formatare standard conţin 512 octeţi de date, la care se adaugă un număr de octeţi utilizaţi pentru controlul intern al unităţii şi pentru detecţia şi corecţia erorilor ceea ce explică diferenţa dintre dimensiunea formatată şi neformatată a discului.

Întreruperea 13h foloseşte adresarea fizică a datelor pe disc. Aceasta include numărul de cilindru, numărul de cap (numărul de fată) şi numărul de sector.

Serviciul BIOS furnizează de asemenea operaţii de format /read /write pentru sectoare de lungime ne-standard şi astfel pot fi prelucrate sectoare scrise sub alte sisteme de operare în afara de MS-DOS. Astfel, deşi DOS-ul foloseşte numai

sectoare de 512 octeţi, BIOS-ul poate suporta sectoare de 128, 256, 512 sau 1024 octeţi şi deci se poate citi discuri scrise sub alte sisteme de operare.

În afară de serviciile BIOS disc, se pot obţine diverse informaţii referitoare la disc prin intermediul a 3 elemente din tabela vectorilor de întrerupere şi anume vectorii 1Eh, 41h şi 46h. Aceşti vectori pointează la tabele de parametri pentru floppy disc, primul hard disc şi al doilea hard disc din sistem. Adresele acestor vectori sunt:

- 0000:0078h pentru INT 1Eh;

- 0000:0104h pentru INT 41h;

- 0000:0118h pentru INT 46h.

De exemplu, adresa tabelei de parametri pentru floppy-disc se poate obţine astfel:

```
XOR AX,AX
MOV ES,AX          ;ES <-- segment IVT
LES BX,ES:[78h]    ;ES:BX <-- adresa tabela parametri
                   ;floppy
```

Tabela de parametri pentru floppy disc se află în ROM-BIOS şi ocupă 11 octeţi. Informaţiile din tabelă se referă la lungimea sectorului în octeţi, lungimea zonelor (GAP) dintre sectoare, informaţii de timming cap /motor unitate etc.

În mod similar, tabela de parametri pentru hard disc (vectorul 41h) poate fi folosită pentru a obţine informaţii precum numărul de cilindri şi capete. Dacă exista un al 2-lea hard disc în sistem, vectorul 46h va pointa la o tabela de parametri cu aceeaşi structura şi aceleaşi semnificaţii ca şi cea pointată de vectorul 41h.

Obs: există totuşi sisteme aşa-numite compatibile IBM-PC care nu respectă semnificaţia vectorilor 41h şi 46h, folosind aceşti vectori într-un mod nestandard, de aceea se recomanda folosirea cu atenţie a acestor vectori.

DOS-ul prezintă anumite funcţii similare cu cele BIOS dar care procesează discuri logice. Există astfel doua moduri principale de acces al discului prin intermediul acestor servicii DOS:

- citire absolută a discului logic - întreruperea 25h;

- scriere absolută a discului logic - întreruperea 26h.

În acest sens, sistemul de operare MS-DOS foloseşte adresa logică pentru accesarea datelor pe disc. Această adresă are o singură componentă şi anume numărul logic al sectorului. Cu alte cuvinte, MS-DOS-ul tratează discul ca o secvenţă continuă de sectoare numerotate de la 0 până la numărul maxim de sectoare de pe disc. Numărul 0 corespunde primului sector de pe prima pista (cea mai dinafară) pe fata 0 (prima fata).

Pentru ca procesul de citire scriere date pe disc să decurgă mai rapid se foloseşte tehnica de întreţesere (interleaving). Această tehnică presupune că sectoarele de pe o pistă sunt numerotate cu un offset constant numit factor de întreţesere (interleaving factor). De exemplu, dacă pista constă din 9 sectoare şi factorul de întreţesere este 5 atunci sectoarele sunt localizate în următoarea ordine: 1, 6, 2, 7, 3, 8, 4, 9, 5.

Obs: - alegerea valorii optime pentru factorul de întreţesere constituie o problema relativ complicată. în acest sens exista mai multe utilitare dedicate iar anumite sisteme PC prezintă rutine BIOS interne pentru determinarea şi setarea optimă a acestui factor;

- dacă un disc conţine mai multe discuri logice (hard disc partiţionat) DOS-ul va numerota sectoarele independent pentru fiecare disc logic;

- pentru a valida BIOS-ul să folosească adresele logice de sector, acestea trebuie convertite mai întâi în forma sector –faţă - pistă.

În principiu, funcţiile realizate de întreruperile 25h şi 26h sunt foarte asemănătoare cu cele ale serviciului BIOS INT 13h fiind astfel folosite pentru citire /scriere sectoare disc. Se pot deci folosi aceste întreruperi pentru editarea şi analizarea zonelor de disc ne-accesibile prin sistemul de fişiere cum ar fi de exemplu sectorul de boot sau directorul rădăcina. în versiunile de DOS mai vechi (sub 3.31) aceste întreruperi vor citi date (sectoare de pe disc) plasându-le intr-un buffer de memorie (pointat de DS:BX) şi respectiv vor scrie date pe disc preluându-le din buffer-ul de memorie. În acest caz vor putea fi procesate discuri logice cu o capacitate până la 32 Mo.

4.3 Pachetul de citire / scriere pentru INT 25h, INT 26h

În cadrul versiunilor mai recente, întreruperile 25h şi 26h folosesc conceptul de pachet de citite /scriere. Adresa pachetului de citire /scriere pentru INT 25h/

INT26h este transmisa prin registrele DS:BX. Registrul CX va conţine valoarea -1 (0FFFFh). Structura pachetului de citire /scriere este următoarea:

Offset	Mărime (octeţi)	Semnificaţie
00h	4	Adresa primului sector logic de procesat
04h	2	Nr. de sectoare de procesat
06h	4	Adresa buffer (Offset, segment) pentru scriere/citire pe/de pe disc

În cazul apariţiei unei erori la execuţia INT 25h, INT 26h, registrul AX va conţine un cod de eroare. Valoarea din AL constituie o extensie a codului de eroare transmis handler-ului întreruperii INT 24h, iar valoarea din AH descrie eroarea apăruta astfel:

01h - comanda greşita sau necunoscuta;

02h - marca de adresa greşita sau inexistenta;

03h - disc write -protected (pentru INT 26h);

04h - sector nelocalizat (negăsit);

08h - eroare DMA;

10h - eroare de transfer de date;

20h - eroare controller;

40h - eroare operaţie seek;

Daca nu exista erori, CF=0 şi AL=00h.

Obs: - întreruperile 25h şi 26h salvează în stiva registrul de fanioane => după terminarea execuţiei acestor întreruperi utilizatorul va trebui fie sa restaureze registrul de fanioane din stiva fie sa ajusteze valoarea SP prin incrementarea cu 2 pentru a alinia stiva în poziţia iniţiala.

4.4 Întreruperea INT 13h

Se prezintă câteva din funcţiile des folosite ale întreruperii **BIOS low-level INT 13h** de acces al discului la nivel fizic.

Funcţia 00h - Resetare sistem disc - funcţia iniţializează controller-u de disc recalibrează unităţile de disc ataşate la controller. Operaţia de recalibrare (RTZ - Return To Zero) presupune poziţionarea capetelor de citire /scriere la pista 0. Funcţia se foloseşte în scopul pregătirii discului şi controller-ului în vederea unor operaţii de I/O şi după o eroare de I/O cu discul înainte ca programul sa reia operaţia care a produs eroarea. Daca iniţializarea se aplica hard discului va fi de asemenea iniţializat şi controller-ul de floppy. O unitate de disc, fiind constituita şi dintr-o parte mecanică, este supusa uneori unor erori de hazard, motiv pentru care în aceste situaţii se recomanda iniţializarea şi repetarea operaţiei de cel puţin 3 ori.

- parametri de intrare: - AH = 00h

  - DL = unitatea de disc:

    - 00h-7Fh - floppy disc;

    - 80h-FFh - hard disc;

- parametri de ieşire:

  - daca funcţia s-a executat fără erori (cu succes):

    - CF=0 - execuţie fără erori;

    - AH=00h;

  - daca funcţia s-a executat cu erori (insucces):

    - CF=1 - execuţie cu erori;

    - AH = octet cod eroare;

Observaţii: octetul cod de eroare codifică diverse tipuri de erori dintre care, cele mai uzuale sunt cele legate de: tentativă de scriere pe un disc protejat la scriere (03h), operaţie de iniţializare fără succes la hard disc (05h), detecţie sector defect la hard disc (0Ah), respectiv pistă defectă la hard disc (0Bh), eroare de controller disc (20h) etc.

Funcţia 01h - Întoarce stare disc - funcţia citeşte şi întoarce starea ultimei operaţii de I/O cu discul. Introducerea parametrilor de intrare se face similar cu funcţia 00h. Starea este întoarsă sub forma

aceluiaşi octet cod eroare în registrul AH iar registrul AL va întoarce valoarea 00h.

Funcţia 02h - Citire sector - funcţia citeşte unul sau mai multe sectoare de pe un disc într-un buffer de memorie.

- parametri de intrare:

  - AH=02h

  - AL =nr. de sectoare de citit

  - CH= nr. pista (cilindru)

  - CL = biţii 0-5: nr. sector în cadrul pistei;

    biţii 6-7:cei mai semnificativi 2 biţi ai numărului pistei (hard disc);

  - DH =număr cap;

  - DL =unitate de disc:

    - 00h-7Fh - floppy disc;

    - 80h-FFh – hard disc;

  - ES:BX = segment: Offset buffer memorie;

- parametri de ieşire:

  - execuţie fără erori:

    - CF=0 - execuţie fără erori;

    - AH=00h;

    - AL =nr. de sectoare citite;

  - execuţie cu erori:

    - CF=1 - execuţie cu erori;

    - AH =octet cod eroare;

Observaţii: - în cadrul hard discurilor cei mai semnificativi 2 biţi (biţii 8 şi 9) ai numărului de cilindru (pista) sunt plasaţi în biţii 6 şi 7 ai registrului CL. Aceşti 2 biţi împreuna cu cei 8 biţi ai registrului CH formează numărul complet al cilindrului:

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

Nr. cilindru low — Nr. cilindru high — Nr. Sector

În cazul în care dorim de exemplu să citim de pe pista 731 începând cu sectorul 10 registrul CX va trebui încărcat cu valoarea totala 0DBCAh.

Dacă se execută funcţia pentru citirea a mai mult de un sector şi apar erori în timpul citirii, operaţia se va încheia după sectorul în care a apărut eroarea.

Funcţia 03h - Scriere sector - funcţia scrie unul sau mai multe sectoare pe un disc dintr-un buffer de memorie. Parametri de intrare şi de ieşire sunt transmişi în mod similar cu funcţia 02h.

Funcţia 04h - Verificare sector - funcţia verifică dacă datele dintr-un anumit nr. de sectoare pot fi citite (compara datele din sectoare folosind coduri CRC). Parametri sunt aceiaşi ca şi în cazul funcţiei 02h.

Funcţia 08h - Întoarce parametri unităţii de disc - funcţia întoarce parametri unităţii de disc specificate.

- parametri de intrare:
  - AH=08h
  - DL = unitate de disc:
    - 00h-7Fh - floppy disc;
    - 80h-FFh – hard disc;
  - parametri de ieşire:
    - daca funcţia s-a executat fără erori (cu succes):
      - CF=0 - execuţie fără erori;

- BL = tipul unităţii de disc floppy (PC-AT , PS/2);

  - 01h=>360K, 40 piste, 5.25";

  - 02h=>1.2 M, 80 piste, 5.25";

  - 03h=>720K, 80 piste, 3.5";

  - 04h=>1.44M, 80 piste, 3.5";

- CH =nr. maxim piste;

- CL = biţii 0-5, nr. maxim sector;

  biţii 6-7, cei mai semnificativi 2 biţi pentru nr. maxim piste;

- DH = nr. maxim cap;

- DL = nr. de unităţi de disc pe primul controller;

- daca funcţia s-a executat cu erori:

  - CF=1 - execuţie cu erori;

  - AH = octet cod eroare;

  - ES:DI = segment: offset al tabelei cu parametrii unităţii de disc;

Observaţii:

- în sistem pot exista mai multe controllere de disc, fiecare poate suporta doua unităţi de disc.

- în cadrul sistemelor IBM-PC-XT funcţia este suportată numai pentru hard disc.

Funcţia 0Dh - Reset sistem hard disc - funcţia este similara cu funcţia 00h numai ca acţionează asupra controller-ului de hard disc. Se va specifica în registrul DL unitatea de hard disc: 80h-FFh.

Funcţia 0Ch - Căutare (poziţionare) pistă - funcţia poziţionează capetele de citire /scriere ale hard discului pe pista specificată (cilindrul specificat).

- parametri de intrare:

  - AH=0Ch;

- CH = nr. cilindru (cei mai puţin semnificativi 8 biţi);

- CL = biţii 6-7, cei mai semnificativi 2 biţi ai nr. cilindru;

- DH = nr. cap;

- DL = unitate hard disc: 80h-FFh;

- parametri de ieşire:

- execuţie fără erori:

- CF=0 - execuţie fără erori;

- AH=00h;

- execuţie cu erori:

- CF=1 - execuţie cu erori;

- AH = octet cod eroare;

Observaţii: - funcţia nu realizează un transfer de date ci numai poziţionarea corespunzătoare a capetelor şi este suportata numai de hard discuri.

Funcţia 10h - Întoarce stare unitate hard disc - funcţia acţionează numai asupra hard discului. Ca şi în cazul celorlalte funcţii, returnează în AH valoarea 00h daca discul este pregătit pentru operaţii de I/O, altfel returnează în AH un octet de cod de eroare. în registrul DL se va specifica unitatea de hard disc: 80h-FFh.

Funcţia 11h - Recalibrare hard disc - funcţia realizează operaţia de recalibrare (aduce capetele de citire /scriere la pista 0). Parametri sunt transmişi în mod similar cu celelalte funcţii referitoare la hard disc. De fapt, această funcţie este apelată în cadrul funcţiilor 00h şi 0Dh.

Observaţii: - în general, în cazul apariţiei unei erori semnalizata de fanionul CF=1, se recomandă reluarea funcţiei respective de cel puţin 3 ori, iar în cazul menţinerii erorii se va prelucra respectiva eroare în conformitate cu codul întors în registrul AH.

4.5 Probleme propuse

Problema 1: Să se scrie un program care afişează conţinutul directorului rădăcina de pe discul curent folosind INT 25h astfel:

- se va obţine discul curent de pe care se va afişa directorul rădăcina (folosind întreruperea INT 21h subfuncţia AH =19h, întoarce în AL discul curent).

  Intrare: AH=19h

  Ieşire: AL – număr unitate curentă (0-A, 1-B, 2-C,...);

- se va citi sectorul de boot pentru a obţine numărul de intrări în directorul rădăcina şi adresa de start a directorului rădăcina folosind întreruperea INT 25h astfel:

  AL= număr unitate (0=A, 1=B, etc.);

  CX=0FFFFh;

  DS:BX =seg: offset bloc parametri.

- pentru calculul efectiv al numărului de intrări şi al adresei de start se vor lua în considerare următoarele:

  - în sectorul de boot, la offset-ul 11h, se află codificat pe un cuvânt numărul maxim de intrări din directorul rădăcina
  - directorul rădăcina ocupă în întregime un anumit număr de sectoare
  - în sectorul de boot, la offset-ul 10h, se afla codificat pe un octet numărul copiilor FAT, la ofsetul 16h se afla codificat pe un cuvânt numărul de sectoare dintr-o copie FAT, iar la ofsetul 0Eh se afla codificat pe un cuvânt numărul de sectoare care preced FAT 1.
  - Spaţiul de pe disc este distribuit în principiu astfel:
    - Sector de boot;
    - FAT #1 (File Allocation Table –tabela Alocării Fişierelor);
    - FAT #2 (copie FAT #1);
    - Director rădăcină;

- Zona de date.

- se va citi pe rând câte un sector din directorul rădăcina afişându-se conţinutul sau astfel:
  - se va citi sectorul curent folosind INT 25h.
  - se vor testa pe rând intrările de director din sector astfel:
    - daca primul octet al intrării este 0 => intrare nefolosită => restul intrărilor vor fi de asemenea nefolosite => se va termina afişarea şi se va ieşi din program;
    - daca intrarea respectiva constituie o intrare ştearsă anterior (codul E5h) => se va trece la următoarea intrare;
    - daca intrarea respectiva este utilizata => se va afişa numele şi eventuala extensie precizându-se la afişare daca este vorba de un subdirector (<SUBDIR>) sau eticheta de volum (<VOLUM>);
    - se vor afişa pe un ecran cele 16 intrări ale unui sector după care, la apăsarea unei taste, se va trece la următorul ecran (următorul sector).

Informaţia de bază relativă la un fişier este păstrată în intrarea de director a respectivului fişier. O intrare de director ocupă 32 octeţi în următorul format:

Ofset	**Mărime**	**Semnificaţie**
00h	8	numele subdirectorului /fişierului (maxim 8 caractere)
08h	3	extensia (maxim 3 caractere)
0Bh	1	atributele fişierului
0Ch	10	rezervaţi
16h	2	ora creării sau a ultimei modificări
18h	2	data creării sau a ultimei modificări
1Ah	2	Numărul primului cluster din lanţ
1Ch	4	Dimensiunea fişierului în octeţi

Numele fişierului este format din 8 caractere. Dacă primul octet este 00h atunci însemnă că intrarea în director nu a fost folosită niciodată, dacă este 0E5h însemnă că fişierul respectiv a fost şters.

Atributele fişierului au următoarea semnificaţie:

7	6	5	4	3	2	1	0
X	X	Arhiva	Director	Volum	Sistem	Ascuns	Read-only

- bit 7 – nefolosit;
- bit 6 – nefolosit;
- bit 5 – fişier de tip arhivă. Acest bit este poziţionat ori de câte ori fişierul este modificat;
- bit 4 – subdirector;
- bit 3 – identificator de volum. Un singur fişier poate avea acest atribut şi trebuie să se afle în directorul rădăcină;
- bit 2 – fişier sistem, nu poate fi găsit prin operaţiile obişnuite de căutare în director. Este folosit pentru desemnarea fişierelor ce formează sistemul de operare;
- bit 1 – fişier ascuns, nu poate fi găsit prin operaţiile obişnuite de căutare în director;
- bit 0 – fişier protejat la scriere. Fişierul poate fi doar citit.

Problema 2: Să se scrie un program de test al rutinei de tratare a erorii critice INT 24h astfel:

- în principiu programul de test va încerca o operaţie simplă cu discul care poate genera o eroare critică (de ex. Încercarea de creare director pe o unitate de dischetă – funcţia 39h, INT 21h)
- într-o primă fază se va reindirecta INT 24h (se va scrie o procedură de redirectare INT 24h) astfel:
- – obţinerea şi salvarea vectorului INT 24h (cel vechi), cu funcţia 35h, INT 21h.
- – instalarea vectorului nou (utilizator) INT 24h, cu funcţia 25h, INT 21h.
- – setarea adresei (nouă) INT 24h în zona de PSP începând la ofsetul 12h folosind funcţia 51h, INT 21h:
  - parametrii intrare: AH=51h

  - parametrii ieşire: BX = adresa segment PSP după care se va depune noua adresă începând la ofsetul 12h în cadrul PSP în ordinea Offset : Segment

- se va scrie noua rutină INT 24h astfel:
  - se va seta un semafor soft de eroarea
  - se va obţine adresa de revenire în programul întrerupt ţinându-se cont de următoarele considerente:
- la intrarea în rutina INT 24h stiva are următorul conţinut:

SP->	dword	adresa de revenire;
	Word	fanioane salvate din rutina INT 21h la intrarea în INT24h;
	Word	AX la intrarea în INT 24h;
	Word	BX la intrarea în INT 24h;
	Word	CX la intrarea în INT 24h;
	Word	DX la intrarea în INT 24h;
	Word	SI la intrarea în INT 24h;
	Word	DI la intrarea în INT 24h;
	Word	BP la intrarea în INT 24h;
	Word	DS la intrarea în INT 24h;
	Word	ES la intrarea în INT 24h;
	Dword	adresă reîntoarcere la apelul INT 24h;
	Word	fanioane salvate ale aplicaţiei DOS la intrarea în INT21h.

- se vor reface registrele AX, BX…,ES din stiva şi se vor restaura flag-urile aplicaţiei DOS

- se va seta flag-ul de Carry

- se va încărca în AL=0- codul de revenire în aplicaţie ţinându-se cont că rutina standard INT 24h întoarce sistemului de operare un cod de revenire în aplicaţie astfel:

  AL=00h – ignorare - se ignoră eroarea;

=01h – retry – reâncercarea operaţiei;

=02h – abort – terminarea aplicaţiei (CTRL BREAK – INT 23h);

=03h - fail – întoarcerea la aplicaţie, indicând insuccesul funcţiei DOS.

- se va reveni în aplicaţie cu un salt la adresa de revenire

Programul principal va decurge astfel:

- se va apela rutina de reindirectare INT 24h;
- se va apela funcţia care va încerca operaţia cu discul . dacă operaţia cu discul reuşeşte se va restaura INT 24h original şi se va încheia programul. Pentru restaurarea INT 24h original se poate apela o rutină care va folosi pentru restaurare funcţia 25h, INT 21h.

În caz de insucces al operaţiei cu discul se va testa dacă este vorba de o eroare critică prin testarea semaforului soft poziţionat pe noua rutină INT 24h (de ex. Se poate apela o rutină care va citi numai semaforul şi îl va întoarce prin intermediul unui registru). În cazul în care nu este vorba de o eroare critică se va citi numai semaforul şi îl va întoarce prin intermediul unui registru . În cazul în care nu este vorba de o eroare critică se va restaura INT 24h original şi se va încheia programul.

Dacă este vorba de o eroare critică, se va reseta semaforul soft , eventual tot printr-o rutină , se va afişa un mesaj de atenţionare de forma “Eroare disc INT 24h – apăsaţi o tastă pentru reâncercare sau ESC pentru abandonare” şi se va da posibilitatea de abandon sau reâncercare operaţie cu discul.

O posibilă implementare a acestei aplicaţii este prezentată în Anexa 2.

5 Interfaţa IDE/ATA

5.1 Scopul lucrării

Lucrarea are ca scop prezentarea regiştrilor şi a comenzilor interfeţei ATA care permit accesul la nivel fizic la informaţiile stocate pe unităţile de disc.

5.2 Registrele interfeţei ATA

IDE este cea mai răspândită interfaţă pentru unităţile de discuri fixe. Denumirea de IDE (Integrated Drive Electronics) se referă la unităţile de discuri care au un controler integrat. Această interfaţă mai este cunoscută şi sub numele de ATA (AT Attachment) care reprezintă standardul ANSI care defineşte interfaţa de conectare la calculatoarele AT.

Interfaţa de comunicaţie cu unităţile de disc pune la dispoziţie la nivel fizic o serie de registre de I/O. Datele sunt transferate în paralel pe 16 biţi între memoria calculatorului şi buffer-ul unităţii de discuri sub acţiunea comenzilor transmise în prealabil de la calculator. Datele citite de pe suport sunt memorate în buffer-ul unităţii, urmând a fi transferate calculatorului, iar datele transferate din memoria calculatorului sunt memorate în buffer-ul unităţii urmând a fi scrise pe suport.

Într-un sistem există de obicei mai multe interfeţe ATA. În continuare sunt prezentate adresele pentru patru asemenea interfeţe:

Interfaţa 1: adrese 1F0h – 1F7h, IRQ 14;

Interfaţa 2: adrese 170h – 177h, IRQ 15 sau IRQ 10;

Interfaţa 3: adrese 1E8h – 1EFh, IRQ 12 sau IRQ 11;

Interfaţa 4: adrese 168h – 16Fh, IRQ 10 sau IRQ 9.

Toţi regiştrii sunt pe 16 biţi şi sunt accesaţi atât de către disc cât şi de către program de acea scrierea acestor regiştrii trebuie să se facă respectând riguros condiţiile cerute. Aceşti regiştrii au următoarele funcţii (voi prezenta doar regiştrii pentru interfaţa 1) :

1F0h - Registru de date

1. la citire: – conţine cei 16 biţi citiţi din buffer-ul unităţii;

2. la scriere: - conţine cei 16 biţi care se doresc scrişi pe disc.

1F1h – registrul de eroare / setare caracteristici

1. **La citire** – reprezintă **registrul de eroare** şi conţine starea ultimei comenzi executate de unitatea de discuri sau conţine un cod de diagnosticare. La terminarea fiecărei comenzi, cu excepţia comenzilor *Execute Device Diagnostic* sau *Device Reset*, conţinutul acestui registru este valid dacă bitul ERR din registrul de stare este 1. La terminarea execuţiei unei comenzi *Execute Device Diagnostic* sau *Device Reset* acest registru conţine un cod de diagnosticare.

Semnificaţia biţilor registrului de eroare variază în funcţie de comanda care a fost executată. Semnificaţia biţilor pentru comenzile de citire / scriere este:

7	6	5	4	3	2	1	0
ICRC	UNC	MC	IDNF	MCR	ABRT	MN	X

- bit 7 (ICRC – Interface CRC) – indică prin valoarea 1 apariţia unei erori CRC în timpul transferului la nivelul interfeţei ATA;
- bit 6 (UNC – Uncorrectable Data Error) – dacă este 1 indică apariţia unei erori care nu a putut fi corectată;
- bit 5 (MC – Media Changed) – este rezervat pentru discurile amovibile şi este pe 1 când există un nou suport în unitate;
- bit 4 (IDNF – ID Not Found) – indică faptul că sectorul cerut nu a fost găsit;
- bit 3 (MCR – Media Change Requested) – este rezervat pentru dicurile amovibile şi este pe 1 când este detectată o cerere de eliminare a suportului;
- bit 2 (ABRT – Aborted Command) – indică abandonarea comenzi cerute deoarece comanda sau parametrii acesteia nu sunt valizi;
- bit 1 (NM – No Media) – rezervat discurilor amovibile şi indică că nu există suport în unitatea cerută;
- bit 0 – rezervat.

2. **La scriere** – reprezintă **registrul de setare caracteristici** şi poate fi utilizat pentru specificarea diferitelor caracteristici ale interfeţei, de exemplu, pentru validarea sau invalidarea memoriei cache prin comanda *SetFeatures*. Este permisă scrierea acestui registru numai dacă biţii BSY şi DRQ din registru de stare sunt 0.

1F2h – registrul contor de sectoare

Acest registru este înscris cu numărul sectoarelor de date care trebuiesc transferate într-o operaţie de citire sau scriere între calculator şi unitate. Registrul trebuie înscris numai dacă biţii BSY şi DRQ din registrul de stare sunt ambii egali cu 0. Conţinutul acestui registru devine un parametru al comenzii atunci când codul comenzii este înscris în registrul de comandă. Pentru comenzile de acces la suport acest registru conţine valoarea 0 la terminarea comenzii dacă nu au fost erori. În cazul apariţiei unei erori acest registru conţine numărul de sectoare care trebuie transferat în scopul terminării operaţiei.

1F3h – registrul de adresă

Conţine adresa inferioară LBA pentru interfaţa LBA pe 32 şi 48 biţi – numărul sectorului asupra căruia se va lucra.

1F4h – registrul de adresă

Conţine adresa mijlocie LBA pentru interfaţa LBA pe 48 biţi şi adresa superioară pentru interfaţa LBA cu acces pe 32 biţi – reprezintă valoarea cea mai puţin semnificativă din numărul cilindrului asupra căruia se va lucra.

1F5h – registrul de adresă

Conţine adresa superioară LBA pentru interfaţa LBA cu acces pe 48 biţi şi este 0 pentru interfaţa LBA cu acces pe 32 biţi – reprezintă valoarea cea mai semnificativă din numărul cilindrului asupra căruia se va lucra.

Ultimele 3 registre prezentate permit înscrierea adresei sectorului la comenzile de citire / scriere care utilizează adresarea LBA. Registrele trebuiesc înscrise numai dacă bitul BSY şi bitul DRQ din registrul de stare sunt egali cu 0. Conţinutul acestor registre devin parametrii ai comenzii atunci când codul comenzii este înscris în registrul de comandă.

1F6h – registrul selecţie dispozitiv

Acest registru se utilizează pentru adresarea unităţii de discuri şi a sectorului. Registrul trebuie înscris numai dacă bitul BSY şi bitul DRQ din registrul de stare sunt ambii egali cu 0. Semnificaţia biţilor din acest registru este următoarea:

7	6	5	4	3	2	1	0
X	LBA	X	DEV	HS3	HS2	HS1	HS0

- bit 7 – rezervat;
- bit 6 (LBA) – selectează modul de adresare pentru sectoare. Dacă acest bit este 0 se selectează adresarea CHS, dacă este 1 se selectează adresarea LBA;
- bit 5 – rezervat;
- bit 4 (DEV – Device Select) selectează prin valoarea 0 prima unitate (master) de pe interfaţa ATA iar prin valoarea 1 cea de-a doua unitate (slave) de pe interfaţa ATA;
- bit 3 ÷0 – (HS3, HS2, HS1 şi HS0) – dacă bitul LBA este 0 aceşti 4 biţi conţin numărul capului din cadrul adresei CHS. Dacă bitul LBA este 1 aceşti biţi conţin cei mai semnificativi 4 biţi ai adresei LBA în cazul adresării pe 28 biţi şi sunt nedefiniţi în cazul adresării pe 48 biţi.

1F7h – registru de comandă sau de stare dispozitiv

1. **La scriere** – în acest **registru se scrie codul comenzi**i care trebuie transmis controlerului unităţii de discuri. Execuţia comenzii începe imediat după ce codul comenzii este înscris în registrul de comandă. Scrierea acestui registru şterge orice condiţie de întrerupere. Cu excepţia comenzii *Device Reset* registrul de comandă trebuie înscris numai dacă bitul BSY şi bitul DRQ din registrul de stare sunt ambii egali cu 0.

2. **La citire** – reprezintă **registrul de stare** care conţine starea curentă a unităţii. Dacă bitul BSY este 0 ceilalţi biţi ai registrului conţin informaţii valide, în caz contrar informaţiile conţinute de ceilalţi biţi sunt invalide. Dacă acest registru este citit de calculator în timpul

unei întreruperi în curs, condiţia de întrerupere este ştearsă. Semnificaţia biţilor din acest registru este următoarea:

7	6	5	4	3	2	1	0
BSY	DRDY	DF	DSC	DRQ	CORR	IDX	ERR

- bit 7 (BSY - Busy) – este pe 1 ori de câte ori unitatea de discuri are controlul asupra registrelor interfeţei, cu excepţia registrului de control al dispozitivului şi registrului alternativ de stare. Dacă bitul BSY este pe 1 unitatea va ignora o scriere în oricare din registrele asupra cărora are controlul. Unitatea va modifica starea bitului DRQ numai dacă bitul BSY este setat. Dacă bitul BSY este 0, unitatea va modifica numai biţii DRDY, DF, DSC , CORR şi IDX din registrul de stare şi conţinutul registrului de date.

- bit 6 (DRDY – Device Ready) – este setat pe 1 pentru a indica faptul că unitatea de discuri acceptă comenzi. Dacă starea acestui bit se modifică, unitatea nu trebuie să modifice din nou starea bitului până la citirea registrului de stare de către calculator. Dacă bitul DRDY este 0, unitatea va accepta şi va încerca execuţia comenzilor *Device Reset* şi *Execute Device Diagnostic*. Celelalte comenzi nu vor fi acceptate, fiind setat bitul ABRT din registrul de eroare şi bitul ERR din registrul de stare, înaintea resetării bitului BSY pentru a indica terminarea comenzii.

- bit 5 (DF – Device Fault) – indică prin valoarea 1 detectarea unei erori de dispozitiv.

- bit 4 (DSC – Device Seek Complete) – indică prin valoarea 1 poziţionarea capetelor unităţii de discuri deasupra unei piste. La apariţia unei erori, acest bit nu trebuie modificat de unitate până la citirea registrului de stare de către sistem, după care bitul DSC va indica starea curentă.

- bit 3 (DRQ – Data Request) – indică prin valoarea 1 faptul că unitatea de discuri este gata pentru transferul datelor între calculator şi unitate. Dup ce calculatorul înscrie codul unei comenzi în registrul de comandă, unitatea setează bitul BSY la valoarea 1 sau bitul DRQ la valoarea 1 până la terminarea comenzii sau până la eliberarea magistralei în cazul unei comenzi suprapuse.

- bit 2 (CORR – Corrected Data) – este utilizat pentru a indica o eroare de date corectabilă. Această condiţie nu determină încheierea unui transfer de

date. Acest bit nu mai este folosit în ultimele versiuni ale standardului ATA.

- bit 1 (IDX – Index) – este specific diferiţilor producători. Acest bit nu mai este definit în ultimele versiuni ale standardului ATA.

- bit 0 (ERR – Error) – este 1 în momentul în care a apărut o eroare în timpul execuţiei comenzii precedente. Biţi din registrul de eroare vor conţine informaţii suplimentare despre cauza erori.

5.3 Execuţia transferurilor de date

În specificaţia ATA sunt definite protocoalele utilizate pentru transferurile de date între calculator şi unitatea şi durata ciclilor de citire / scriere.

Un exemplu de protocol pentru citire în modul PIO, care este un protocol de transfer fără confirmare, şi care pentru fiecare cuvânt transferat procesorul trebuie să execute o secvenţă de program.

1. Se aşteaptă ca bitul BSY să devină 0 prin interogarea registrului de stare;
2. Se scrie în registrul de selecţie al dispozitivului valoarea corespunzătoare pentru dispozitiv;
3. Se aşteaptă bitul BSY=0 şi bitul DRDY =1;
4. Se scriu parametrii comenzii în registrele corespunzătoare;
5. Se scrie codul comenzi în registrul de comandă;
6. Unitatea setază bitul BSY şi se pregăteşte pentru transferul primului bloc de date;
7. Când blocul este disponibil unitatea setează bitul DRQ. Dacă apare o eroare sunt setaţi biţii corespunzători de eroare şi de stare. În final unitatea resetează bitul BSY şi activează semnalul INTRQ;
8. Când se detectează bitul BSY pe 0 sau s-a generat o întrerupere calculatorul citeşte şi salvează conţinutul registrului de date;
9. Dacă DRQ este setat, calculatorul transferă un bloc de date prin citirea registrului de date;

10. Ca răspuns la citirea registrului de stare, unitatea dezactivează semnalul INTRQ. Ca răspuns la citirea întregului bloc de date, se execută una din următoarele operaţii:

    a. Dacă nu a apărut nici o eroare unitatea setează bitul BSY şi secvenţa de mai sus se repetă de la pasul 7;

    b. Dacă a apărut o eroare unitatea şterge bitul DRQ iar execuţia comenzii este terminată;

    c. Dacă s-a transferat ultimul bloc, unitatea şterge bitul DRQ, execuţia comenzii fiind terminată.

5.4 Comenzi ATA

În continuare sunt prezentate principalele comenzi ATA, codurile acestora şi regiştrii care trebuiesc încărcaţi cu parametrii comenzii.

Comandă	**Cod**	**Registru afectat**			
		1F1h	1F2h	1F3÷5h	1F6h
Check Power Mode	E5h				D
Device Configuration Identify	B1h	C2h			D
Device Configuration Restore	B1h	C0h			D
Device Configuration Set	B1h	C3h			
Device Reset	08h				D
Download Microcode	92h	X	X	X	X
Execute Device Diagnostic	90h				
Flush Cache	E7h				D
Flush Cache Ext	EAh				D
Get Media Status	DAh				D
Identify Device	ECh				D
Identify Packet Device	A1h				D
Idle	E3h	X			D

Idle Immediate	E1h				D
Media Eject	EDh				D
Media Lock	DEh				D
NOP	00h				D
Packet	A0h	X	X	X	D
Read Buffer	E4h				D
Read DMA	C8h		X	X	X
Read DMA Ext	25h		X	X	D
Read Multiple	C4h		X	X	X
Read Multiple Ext	29h		X	X	D
Read Sector(s)	20h		X	X	X
Read Sector(s) Ext	24h		X	X	D
Read Verify Sector(s)	40h		X	X	X
Read Verify Sector(s)	42h		X	X	D
Seek	70h			X	X
Service	A2h				D
Set Features	EFh	X	X	X	D
Set Max Address	F9h			X	X
Set Max Address Ext	37h			X	D
Sleep	E6h				D
Standby	E2h		X		D
Standby Immediate	E0h				D
Write Buffer	E8h				D
Write DMA	CAh		X	X	X
Write DMA Ext	35h		X	X	D
Write Multiple	C5h		X	X	X

Write Multiple Ext	39h		X	X	D
Write Sector(s)	30h		X	X	X
Write Sector(s)	34h		X	X	D

5.5 Probleme propuse

Problema 1: Să se scrie un program care utilizând comanda *Identify Device* (codul ECh) adresată controlerului de disc va determina parametrii unităţii de disc (număr de cilindrii, capete, sectoare pe pistă) precum şi tipul unităţii de disc (model, revizie, număr de serie). Pe baza numărului total de sectoare adresabile, calculaţi capacitatea totală în GB a discului, ţinând cont că un sector conţine 512 octeţi.

Protocolul pentru transmiterea comenzii *Identify Device* şi preluarea informaţiilor întoarse de unitate este:

1. Se aşteaptă ca bitul BSY din registrul de stare să devină 0;
2. Se înscrie în registrul de comandă codul comenzii (ECh);
3. Se aşteaptă ca bitul BSY din registrul de stare să devină 0;
4. Se aşteaptă ca bitul DRQ din registrul de stare să devină 1;
5. Se citeşte un cuvânt din registrul de date şi se memorează;
6. Dacă nu s-au citit toate cele 256 de cuvinte, se continuă cu operaţia de la pasul 3.

Obs:

Comanda *Identify Device* permite citirea parametrilor unităţii de discuri. La recepţionarea acestei comenzi unitatea transmite un bloc de 256 cuvinte cu informaţii care conţin toate detaliile asupra unităţii: producătorul, model, seria, parametri de funcţionare etc. Semnificaţia celor 256 de cuvinte întoarse de controlerul de disc pentru comanda *Identify Devive* este:

Cuvânt	**Semnificaţie**
0	Informaţii generale de configuraţie;

1,2	Număr de cilindrii pentru modul de transfer implicit;
3	Număr de capete pentru modul de transfer implicit;
4,5	Nespecificaţi;
6	Număr de sectoare pe pistă pentru modul de transfer implicit;
7,8,9	Nespecificaţi;
10-19	Număr de serie (dacă cuvântul 10 nu este 0000h seria este reprezentată pe 20 caractere ACSII);
20-22	Nespecificaţi;
23-26	Revizie (dacă cuvântul 23 nu este 0000h codul de revizie reprezină 8 caractere ASCII);
27-46	Model (dacă cuvântul 27 nu este 0000h modelul este reprezentat pe 40 de caractere ASCII);
47-53	Nespecificat;
54	Număr de cilindrii pentru modelul de translatare curent;
55	Număr de capete pentru modelul de translatare curent;
56	Număr de sectoare pe pistă pentru modelul de translatare curent;
57-58	Număr de sectoare în modul pentru modul de translatare curent;
59	Nespecificat;
60-61	Numărul total de sectoare adresabile (adresare pe 28 de biţi);
62-99	Nespecificat
100-103	Numărul total de sectoare adresabile (adresare pe 48 de biţi);
104-159	Nespecificaţi;
160-255	Rezervaţi;

6 Comunicaţia serială. Standardul RS-232C

6.1 Scopul lucrării

Lucrarea prezintă portul serial standard disponibil în calculatoarele compatibile IBM şi urmăreşte familiarizarea studenţilor cu principiile de comunicaţie serială. Sunt prezentate regiştrii interfeţei de comunicaţie serială, modul de utilizare şi întreruperile aferente acesteia.

6.2 Consideraţii generale

Porturile seriale sunt folosite în general pentru dispozitivele care trebuie să comunice bidirecţional cu sistemul; printre acestea numărându-se modemul, mouse-ul, scannerul, digitizorul sau orice alt dispozitiv care trimite şi primeşte date de la PC.

Interfaţa serială asincronă este dispozitivul de bază în comunicaţia dintre sisteme. Se numeşte *asincronă* pentru că nu există nici un semnal de sincronizare sau de ceas, astfel încât caracterele pot fi trimise la orice interval de timp (ca atunci când operatorul introduce date de la tastatură). Fiecare caracter transmis printr-o linie serială este încadrat de un semnal de start şi unul de stop. Un singur bit 0, numit bit de start, precede fiecare caracter, pentru a anunţa sistemul destinatar că următorii opt biţi constituie un octet de date. Caracterul este urmat de unu sau doi biţi de stop, care anunţă terminarea transmiterii lui. La recepţie, caracterele sunt recunoscute după semnalele de start şi de stop, şi nu după temporizarea sosirii lor. Interfaţa asincronă este orientată spre caracter şi are o suprasarcină (un excedent de date) de aproximativ 20% datorită informaţilor suplimentare necesare identificării fiecărui caracter. Portul serial existent în PC respectă standardul RS 232 (Reference Standard 232).

Atributul *serial* se referă la datele transmise pe o linie, biţii succedându-se în serie pe măsură ce sunt transmişi. Acest tip de comunicaţie, se foloseşte la sistemul telefonic, deoarece el furnizează câte o linie de date pentru fiecare direcţie.

Standardul RS 232 este cel mai folosit standard de cuplare serială a două echipamente de calcul. Cuplele de legătură serială prezente în PC sunt cu 9, respectiv cu 25 pini. Semnalele prezente în cupla cu 9 pini sunt:

Pin1: DCD (Data Carrier Detect) detecţie purtătoare (intrare);

Pin2: RD – recepţie date (intrare);

Pin3: TD – transmisie date(ieşire);

Pin4: DTR (Data Terminal Ready) – terminal pregăti (ieşire);

Pin5: GND – masa;

Pin6: DSR (Data send ready)- modem pregătit (intrare);

Pin7: RTS (Request to send) – cerere de emisie (ieşire);

Pin 8: CTS (clear to send) – gata de emisie (intrare);

Pin9: RI (Ring indicator) – indicator de apel (intrare).

Lungimea maximă a liniei de comunicaţie pentru care schimbul de informaţie este corect este de 15m.

În PC comunicaţia serială se poate face prin intermediul a patru porturi COM. Zona de date BIOS conţine o listă a celor patru adrese de bază corespunzătoare celor patru porturi (adresele 0:0400h…0:0407h). În cursul rutinei de iniţializare POST se testează şi se iniţializează porturile COM1 şi COM2. Adresele de port corespunzătoare acestora sunt:

COM1 – 3F8h..3FFh;

COM2 – 2F8h..2FFh.

Asupra celor patru porturi se poate opera direct, prin instrucţiuni de intrare / ieşire sau prin intermediu întreruperii BIOS 14h. Această întrerupere va lucra cu oricare din cele patru porturi, cu condiţia ca adresa de bază a portului serial solicitat să se găsească în tabela din zona de date BIOS. Totodată, este necesar ca două porturi să nu împartă acelaşi spaţiu de adresare, caz în care nici unul va funcţiona corect.

Utilizarea porturilor seriale se poate face prin interogare , sau prin întreruperi. În acest din urmă caz liniile de întrerupere folosite, corespunzătoare controlului de întreruperi 8259A sunt:

COM1 – IRQ4 – INT 0Ch;

COM2 – IRQ3 – INT 0Bh.

6.3 Regiştrii asociaţi cuplorului de comunicaţie serială

Regiştrii asociaţi unui cuplor de comunicaţie serială sunt (adresele sunt date pentru COM 1):

3F8h – registrul de date, registrul de divizare

1. *La scriere:* – registrul de date, conţinând cei 8 biţi ai caracterului ce trebuie transmis. Când bitul DLAB=1, conţine octetul inferior al divizorului frecvenţei ceasului, care împreună cu octetul superior, scris la adresa 3F9h determină rata transmisiei seriale după cum urmează:

Rata(b/s)	Valoarea de divizare (zecimal)
75	1536
110	1047
150	768
300	384
600	192
1200	96
2400	48
4800	24
9600	12
19200	6
38400	3
57600	2
115200	1

Formula pentru calculul valorii pentru Baud Rate este:

$$BaudRate = \frac{frecventa\_oscilator}{16 * valoare\_divizare}$$

Pentru placa de bază a unui sistem de calcul frecvența oscilatorului de cuarț propriu de la care se pornește calculul ratei de eșantionare este de 1,843200MHz. Astfel în momentul în care cunoaștem exact valoarea de baud Rate pe care vrem sa o specifică se poate calcula valoarea de divizare.

2. *La citire*: buffer de recepţie, conţinând cei 8 biţi ai caracterului recepţionat

3F9h – registrul validare întrerupere, registrul de divizare

1. *La scriere*: Când bitul DLAB =1 conţine octetul superior al valorii de divizare a frecvenţei ceasului. Când DLAB =0 reprezintă registrul de validare întreruperi:

7	6	5	4	3	2	1	0
0	0	0	0	IMODEM	IERR	IE	IRD

- bit 0 (IRD) – valoarea 1 validează generarea unei întreruperi la recepţie de date;
- bit 1 (IE) – valoarea 1 validează generarea unei întreruperi când buffer-ul de emisie este gol;
- bit 2 (IERR) – valoarea 1 validează generarea unei întreruperi la detectarea unei erori sau oprire;
- bit 3 (IMODEM) – valoarea 1 validează generarea unei întreruperi la schimbarea stării modem-ului (CTS,DSR,RI,RLSD);
- bit 4..7 – au valoarea 0 (sau 1 depinzând de producătorul plăcii de bază).

3FAh – registrul cauză a întreruperii

1. *La citire:* registrul cauză a întreruperii. La apariţia unei întreruperi citirea acestui registru determină natura întreruperii. Semnificaţia biţilor este următoarea:

bit 0 – valoarea 1 specifică faptul că întreruperea este activă (poate fi folosit pentru interogare);

bit 2,1 – valoarea 00 specifică întreruperea cauzată de o eroare (suprascriere, paritate, încadrare) sau oprire. Este resetat prin citirea registrului de stare a liniei (3FDh);

– valoarea 10 specifică faptul că sunt date recepţionate disponibile. Este resetat prin citirea bufferului de recepţie (3F8h);

– valoarea 01 specifică faptul că bufferul de emisie este gol. Este resetat prin scrierea în bufferul de emisie(3F8h);

– valoarea 11 specifică faptul că întrerupere este cauzată de schimbarea stării modem-ului. Este resetat prin citirea registrului de stare a modem-ului (3Feh);

bit 3…7 – sunt 0.

3FBh registrul de control al liniei

Disponibil la scriere sau citire:

7	6	5	4	3	2	1	0
DLAB	SLinie	X	P1	P0	Stop	WL1	WL0

- bit 0,1(WL0,WL1) - reprezintă lungimea cuvântului de date:
  - 00= pentru 5 biţi date;
  - 01= pentru6 biţi date;
  - 10= pentru 7 biţi date;
  - 11= pentru 8 biţi date.
- bit 2(Stop) - numărul de biţi de stop: 0 – pentru 1 bit de stop, 1 – pentru 2 biți de stop;
- bit 3,4(P0,P1) - paritate: x0=fără,01=impară,11=pară;
- bit 5 - nu este folosit de către BIOS;
- bit 6(SLine) - validează controlul opririi, 1=transmisia începe prin emiterea de 0-uri (spaţii);
- bit 7(DLAB) - DLAB (Divisor Latch Access Bit),
  - 1=se programează rata de transmisie;
  - 0=normal.

3FCh*:* registrul de control al modem-ului

Disponibil numai pentru scriere:

7	6	5	4	3	2	1	0
0	0	0	X	OUT2	OUT1	RTS	DTR

- bit 0 - 1=activează DTR, 0=dezactivează DTR;
- bit 1 - 1=activează RTS, 0=deactivează RTS;
- bit 2 - 1=activează OUT1;
- bit 3 - 1=activează OUT2 (necesar în cazul lucrului prin întreruperii);
- bit 4 - 1=activează bucla pentru testare;
- bit 5..7: sunt 0.

3FDh*:* registrul de stare a liniei.

Disponibil pentru citire. Biţii 1..4 cauzează generarea unei întreruperi dacă aceasta este validă.

- bit 0 – 1 = date receptate. Este resetat prin citirea buffer-ului de recepţie;
- bit 1 – 1 = eroare de suprascriere, caracterul precedent fiind pierdut;
- bit 2 – 1 = eroare de paritate. Este resetat prin citirea registrului de stare a liniei;
- bit 3 – 1 = eroare de încadrare, caracterul recepţionat conţine un bit de stop invalid;
- bit 4 – 1 = este indicată oprirea ; se recepţionează spaţii;
- bit 5 – 1 = buffer de emisie gol, este cerută emisia următorului caracter;
- bit 6 – 1 = transmiţătorul este inactiv, nici o dată nefiind procesată;
- bit 7 – nefolosit.

3Feh*:* registrul de stare al modem-ului.

Disponibil pentru citire. Biţii 0..3 determină generarea unei întreruperi dacă aceasta este validată.

7	6	5	4	3	2	1	0
DCD	RI	DSR	CTS	ΔDCD	ΔRI	ΔDSR	ΔCTS

- bit 0 – 1 = delta CTS şi-a schimbat starea;
- bit 1 – 1 = delta DSR şi-a schimbat starea;

- bit 2 – 1 = se detectează fronturi ale semnalului RI;
- bit 3 – 1 = delta DCD şi-a schimbat starea;
- bit 4 – 1 = CTS este activ;
- bit 5 – 1 = DSR este activ;
- bit 6 – 1 = RI este activ;
- bit 7 – 1 = DCD este activ.

6.4 Întreruperea 14h

Întreruperea 14h pune la dispoziţie următoarele servicii pentru interfaţa serială

AH=00h- iniţializarea portului serial

La apel: DX = numărul portului (0,1)

AL = parametrii de iniţializare, conform următoarelor câmpuri;

Bit: 0,1: lungimea cuvântului de date(10=7biţi,11=8biţi);

2: numărul biţilor de stop(0=1,1=2);

3,4: paritatea (x0=fară,01=impară, 11=pară);

5..7: rata de transmisie: 000=110, 000=150, 010=300, 011=600, 100=1200, 101=2400, 110=4800, 111=9600.

La revenire: AH= starea linii seriale

Bit: 0: 1 = date receptate;

1: 1 = eroare de suprascriere, caracterul precedent fiind pierdut;

2: 1 = eroare de paritate;

3: 1 = eroare de încadrare;

4: 1 = oprire detectată;

5: 1 = buffer de transmisie gol;

6: 1 = registrul de deplasare la transmisie gol;

7: 1 = timeout.

AH= stare modem

Bit: 0: 1 = delta CTS;

1: 1 = delta DSR;

2: 1 = se detectează fronturi ale semnalului RI;

3: 1 = delta DCD;

4: 1 = CTS este activ;

5: 1 = DSR este activ;

6: 1 = RI este activ;

7: 1 = DCD este activ;

AH=01h- emisie caracter

La apel: DX = numărul portului (0,1)

AL = caracterul ce trebuie emis

La revenire: AL = codul caracterului transmis

Dacă bitul 7 al lui AH este setat a intervenit o eroare şi ceilalţi 7 biţi ai lui AH conţin starea liniei de comunicaţie (ca mai sus)

AH=02h recepţie caracter

La apel: DX = numărul portului

La revenire:

AL = caracterul emis;

AH este diferit de zero dacă a apărut o eroare.

Ah=03h – citirea stării portului serial

La apel: DX = numărul portului(0..1);

La revenire: AH – starea liniei seriale (ca mai sus).

O problemă obişnuită existentă în transmisiile seriale este aceea a protocolului de comunicaţie între emiţător şi receptor. La emiţător / receptor există în mod obişnuit câte un buffer de dimensiune finită pentru datele ce urmează a fi emise / recepţionate. Când buffer-ul este plin calculatorul ignoră datele noi până ce buffer-ul se goleşte suficient.

Pentru evitarea acestui fenomen se implementează protocoale hard sau soft, prin intermediul cărora emiţătorul şi receptorul îşi semnalizează unul altuia umplerea buffer-ului propriu.

Protocolul XON/XOFF este un protocol soft care funcţionează pe următorul principiu: când buffer-ul de recepţie este plin , receptorul trimite un caracter XOFF (19 în zecimal) pentru a comunica emiţătorului să oprească emisia datelor. Când buffer-ul de recepţie devine suficient de gol, receptorul trimite un caracter XON (17 în zecimal) pentru a indica faptul că transmisia datelor poate reîncepe. Când este folosit protocolul XON / XOFF codurile XON şi XOFF vor fi folosite întotdeauna ca şi coduri de control şi nu ca date, deci acest protocol nu este recomandat în cazul transmisiilor binare.

6.5 Probleme propuse

Problema 1. Să se configureze portul serial pentru transmiterea şi recepţia datelor la diferite rate de comunicaţie prin polling (interogare).(Se va modifica rata de la 110 la 115200).

Problema 2. Să se realizeze un program de comunicaţie pe portul serial între două calculatoare (prin polling). Un program de chat pe portul serial intre 2 calculatoare.

Problema 3. Să se realizeze un program de comunicaţie pe portul serial prin întrerupere, indirectând întreruperea 0Ch. (Vezi Anexa 3). Se va reface programul anterior în care transmisia se va realiza prin interogare iar recepția prin întrerupere.

Obs: - La iniţializarea portului serial se va activa IRQ4 în controlerul 8259A resetând bitul 4 din octetul citit de la portul 21h.

- La părăsirea rutinei de tratare a întreruperii 0Ch trebuie semnalizat controlerului de întreruperi (8259A) faptul ca această întrerupere s-a încheiat prin scrierea cuvântului 20h (EOI) la portul 20h.

Problema 4. Se consideră un dispozitiv conectat la portul serial al unui sistem de calcul. Dispozitivul are specificat următoarele caracteristici de comunicație pe portul serial: 19200b/s, 8 biți date, 1 bit stop, fără paritate. Dispozitivul răspunde la următoarele comenzi:

- '0'…'7' – aprinde un led din cele 8 disponibile;
- '8' – aprinde toate cele 8 led-uri;
- '9' – stinge toate cele 8 led-uri;
- 's' sau 'S' – emite pe portul serial caracterul 'L' sau 'A' în funcție de starea unui buton de pe dispozitiv;
- 'p' sau 'P' – din acest moment orice caracter recepționat și dispozitivul nu știe să îl interpreteze îl va transmite înapoi spre PC;
- 'o' sau 'O' – se oprește transmiterea în ecou a caracterelor netratate;

Să se realizează un program care să ne permită să transmitem comenzi la acest dispozitiv și să putem vizualiza ceea ce transmite dispozitivul.

7 Citirea cartelelor electronice

7.1 Scopul lucrării

Lucrarea are ca scop prezentarea modului de funcţionare a cartelelor electronice şi propune o conectare a acestora pe un port de comunicaţie serială, chiar dacă acestea nu folosesc protocolul de comunicaţie serială pentru transmiterea informaţilor.

7.2 Consideraţii generale

O cartelă telefonică, este o memorie de tip EPROM, având două zone de memorie şi o capacitate de 128, 256 sau 512 biţi. Ea are o ieşire serială, un contor de adrese şi câţiva pini de control. Semnificaţia pinilor este următoarea:

Pin	**Semnificaţie**
Pin 1	Vcc
Pin 2	RST
Pin 3	CLK
Pin 4	GND
Pin 5	Vpp
Pin 6	I/O

7.3 Organizarea memoriei

În prima zonă de memorie sunt stocate date referitoare la producător (un cod al acestuia), seria cartelei, valoarea cartelei, date şi centre de distribuţie. Această zonă este protejată la scriere, prin arderea unui fuzibil intern de către producător în timpul procesului de fabricaţie.

În cea de-a doua zonă de memorie sunt conţinute datele referitoare la creditul rămas disponibil pe cartelă. Înscrierea de date în această zonă se face pe principiul programării EPROM-urilor şi de aceea nu este posibil ca să reîncărcăm o cartelă telefonică, ci doar să-i micşorăm creditul rămas. Pentru a

putea fi reâncărcată, o cartelă telefonică trebuie mai întâi ştearsă prin expunerea la raze ultraviolete, dar aceasta nu este posibil deoarece cartela nu este prevăzută cu o fereastră de ştergere ca în cazul EPROM-urilor de uz general.

Cartelele telefonice utilizate în România sunt memorii de 128 de biţi şi sunt realizate în tehnologie CMOS. Din aceşti 128 de biţi, primi 64 sunt protejaţi la scriere (Read Only), următorii 40 de biţi sunt de tip Read-Write şi ultimi 24 sunt setaţi în "1" logic din fabricaţie. Harta memoriei este prezentată în tabelul următor (pentru o cartelă telefonică franceză):

Octet	Bit	Valoarea hexa
1-2	1…16	10h/2Bh
3	17…24	2Fh
4	25…32	2Ah – producător: Solaic 4Ah – producător: ODS 8Ah – producător: G+D CAh – producător: Gemplus
5-8	33…64	Identificatorul producătorului
9-13	65…104	Zona de numărare
14-16	105…128	Zona de biţi setaţi pe "1"

Identificatorul cartelei este o zonă de 40 de biţi read-only ce conţine numărul de serie al cartelei, valoarea iniţială, data de fabricaţie, centrele de distribuţie ale cartelei.

Zona de numărare conţine date despre valoarea creditului rămas disponibil pe cartelă. Numărarea unităţilor rămase disponibile se face în octal şi are la bază următorul principiu: valoarea fiecărui octet din cei cinci ce compun zona de numărare va fi dată de numărul de biţi setaţi în "1" logic astfel : 00001111 va fi egal cu 4, 00011111 va fi egal cu 5. Valoarea totală a unităţilor disponibile se calculează ca fiind $\sum_{k=0}^{4} 8^k * V_k$ unde V_k este valoarea octetului k.

Exemplu:

Octet 9	Octet 10	Octet 11	Octet 12	Octet 13
00000111	00111111	01111111	00000001	00000011

3(octal)	6(octal)	7(octal)	1(octal)	2(octal)
Total unităţi: $3*8^4 + 6*8^3 + 7*8^2 + 1*8^1 + 2*8^0 = 15818$ unităţi				

7.4 Principiul de funcţionare

In ceea ce priveşte transferarea datelor din cartelă în calculator trebuie respectat următorul protocol de comunicaţie:

Iniţierea porneşte prin resetarea cartelei telefonice astfel: se trece liniile RST si CLK pe „0” pentru o perioada de timp, urmat de trecerea liniei RST pe „1” şi după o perioada de timp şi a liniei CLK. După această operaţiune, indicatorul de adresă (contorul de adresă) care stochează poziţia de citire din memoria cartelei, va fi resetat la valoarea 0. Trebuie avut în vedere că acest indicator nu poate fi resetat atâta timp cât are valoarea cuprinsă între 0 şi 7. Pentru a putea reseta, acest indicator trebuie incrementat astfel ca valoarea lui să depăşească valoarea 7. Incrementarea se face ţinând linia de reset în starea logică “1” şi aplicarea unui număr de impulsuri pe linia de tact.

Pentru extragerea datelor se procedează astfel: se setează linia de reset în starea logică “1” iar pe linia CLK se aplică un semnal de tact. Indicatorul de adresă este incrementat cu 1 la fiecare front descrescător al semnalului de tact, iar datele conţinute în fiecare bit adresat sunt scoase la ieşire spre pinul I/O pe fiecare front crescător al tactului. Decrementarea indicatorului de adrese se realizează prin resetarea cartelei şi implicit a indicatorului după care se incrementează indicatorul la valoarea dorită. Primul bit care se obţine este bitul 0 (cel mai puţin semnificativ) din primul octet. Dispozitivul inversează hardware valoare de pe pinul I/O de aceea valoare obţinută trebuie inversată software.

Obs. *Linia RST de la cartelă este legata la linia RTS a portului serial, linia CLK de la cartelă este legata la linia DTR de la portul serial iar linia de I/O de la cartela este conectata la linia CTS din portul serial*.

7.5 Probleme propuse

Problemă: Să se scrie un program care citeşte o cartelă telefonică şi afişează datele citite în valoarea lor hexa precum şi numărul de unităţi rămase disponibile pe cartelă.

8 Utilizarea comunicaţiei paralele în PC

8.1 Scopul lucrării

Lucrarea prezintă porturile paralele ale calculatoarelor compatibile IBM PC şi urmăreşte familiarizarea studenţilor cu diferite soluţii de conectare a unor echipamente externe la aceste porturi. Sunt prezentate semnalele şi regiştrii disponibili ai interfeţei pentru toate cele trei tipuri de porturi paralele: portul bidirecţional standard, portul EPP şi portul ECP. De asemenea sunt prezentate metodele de transfer al datelor între două calculatoare prin interogare şi întreruperi.

8.2 Consideraţii generale

Porturile paralele se folosesc în primul rând pentru imprimante şi funcţionează normal ca porturi unidirecţionale, deşi câteodată pot fi folosite şi bidirecţional. Un port paralel are opt linii pentru transmiterea simultană a tuturor biţilor unui octet de date prin intermediul a opt fire. Interfaţa este rapidă şi, de obicei, rezervată pentru imprimante, nu pentru comunicaţia dintre calculatoare. O problemă a porturilor paralel este că nu se pot extinde cablurile la orice lungime fără a amplifica semnalul, întrucât apar erori de transmisie a datelor.

Semnalele pinilor pentru portul paralel din PC standard (SPP):

Pin	Descriere	Direcţia	Registrul	Inversat hard
1	Strob	Ieşire /Intrare	Control	Da
2	Bit de date 0	Ieşire	Date	
3	Bit de date 1	Ieşire	Date	
4	Bit de date 2	Ieşire	Date	
5	Bit de date 3	Ieşire	Date	
6	Bit de date 4	Ieşire	Date	
7	Bit de date 5	Ieşire	Date	
8	Bit de date 6	Ieşire	Date	

9	Bit de date 7	Ieşire	Date	
10	Confirmare	Intrare	Stare	
11	Ocupat	Intrare	Stare	Da
12	Terminare hârtie	Intrare	Stare	
13	Selecţie	Intrare	Stare	
14	Salt la rând nou automat	Ieşire /Intrare	Control	Da
15	Eroare	Intrare	Stare	
16	Iniţializare imprimantă	Ieşire /Intrare	Control	
17	Selecţie intrare	Ieşire /Intrare	Control	Da
18	retur bit 0 (masă)	Intrare		
19	retur bit 1 (masă)	Intrare		
20	retur bit 2 (masă)	Intrare		
21	retur bit 3 (masă)	Intrare		
22	retur bit 4 (masă)	Intrare		
23	retur bit 5 (masă)	Intrare		
24	retur bit 6 (masă)	Intrare		
25	retur bit 7 (masă)	Intrare		

Există mai multe tipuri de porturi paralele:

- unidirecţionale pe patru biţi;
- bidirectional pe 8 biţi SPP (Standard Parallel Port);
- EPP (Enhanced Parallel Port);
- ECP (Enhanced Capabilites Port).

8.3 Portul unidirecţional pe 4 biţi

Natura unidirecţională a portului paralel al calculatorului PC original este conformă utilizării sale iniţiale, şi anume transmiterea de date către o imprimantă. Totuşi, existau situaţii în care era de dorit să ai un port

bidirecţional, de exemplu când sunt necesare semnale venite de la o imprimantă. Acest port a fost modificat astfel încât să permită utilizarea unei intrări pe patru biţi pentru a prelua patru linii de semnale. Astfel, aceste porturi puteau să furnizeze ieşiri pe 8 biţi sau să preia intrări pe 4 biţi. Aceste porturi ating de obicei rate de transfer de 40-60 Ko pe secundă, cu anumite modificări pot ajunge la 140 Ko pe secundă.

8.4 Portul bidirecţional pe 8 biţi (SPP)

Aceste porturi, lansate în 1987, se întâlnesc în calculatoarele personale actuale şi sunt denumite porturi paralele de tip PS/2. Prin aceste porturi se poate comunica între calculator şi diferite dispozitive periferice, oferind o rată de 80 până la 300Ko . De fapt este portul unidirecţional pe 4 biţi la care s-au mai definit câţiva dintre pinii anterior nefolosiţi ai conectorului paralel şi prin definirea unui bit de stare care să indice direcţia în care circulă informaţia de-a lungul canalului de date.

8.5 Portul EPP (Enhanced Parallel Port)

Aceasta este o specificaţie mai nouă, denumită câteodată port paralel Fast Mode(în mod rapid). Portul EPP lucrează la o frecvenţă mai mare şi oferă o creştere de 10 ori a capacităţii de transfer faţă de un port paralel convenţional. A fost conceput pentru periferice care utilizează porturi paralele, cum ar fi adaptoarele LAN, unităţi de disc şi unităţi de bandă. Ratele de transfer ale acestui port sunt de la 500KB/s până la 2 MB/s pe secundă. Portul EPP generează şi controlează toate transferurile la şi de la periferic. Pentru a utiliza modul EPP, un set diferit de regiştrii şi etichete sunt asignate pentru fiecare linie.

Semnalele pinilor pentru portul paralel în modul EPP

Pin	Semnalele SPP	Semnalele EPP	IN/OUT	Funcţia
1	Strobe	Write	Out	“0” – indică scriere “1” – indică citire
2-9	Data 0-7	Data 0-7	IN/OUT	Busul de date bidirecţional
10	Ack	Interrupt	In	Linia de întrerupere

11	Busy	Wait	In	Utilizat pentru sincronizare. Un ciclu poate fi startat când linia este în “0”şi terminat când este în “1”
12	Paper out	-	In	Neutilizat în EPP mode
13	Select	-	In	Neutilizat în EPP mode
14	Auto Linefeed	Data Strobe	Out	“0” indică transfer de date
15	Error	-	In	Neutilizat în EPP mode
16	Initialize	Reset	Out	Reset activ pe “0”
17	Select Printer	Address Strobe	Out	“0” – transfer de adresă
18-25	Ground	Ground	GND	

Paper Out, Select şi Error nu sunt folosite în modul EPP. Aceste linii pot fi utilizate în alt scop de către utilizator. Starea acestor linii pot fi determinate în orice moment citind regiştrii modului SPP.

8.5.1 Regiştrii modului EPP

Portul EPP are un set suplimentar de regiştrii în comparaţie cu modul SPP. Totuşi regiştrii modului SPP nu sunt înlocuiţi ci sunt completaţi cu alţii noi.

În tabelul următor sunt prezentate adresele portului EPP.

Adresa	Tipul registrului	Read /write
Baza +0	Date (SPP)	Write
Base +1	Stare (SPP)	Read
Base + 2	Control (SPP)	Write
Base + 3	Adrese (EPP)	Read /Write
Base + 4	Date (EPP)	Read /Write
Base + 5	Nedefinit (transfer pe 16 /32 biţi)	
Base + 6	Nedefinit (transfer pe 32 biţi)	
Base + 7	Nedefinit (transfer pe 32 biţi)	

Primele 3 adrese sunt exact ca la modul SPP şi se utilizează în acelaşi fel. Restul sunt utilizate doar în modul EPP. Pentru a trimite date pe portul paralel setat în modul EPP datele se pot scrie la adresa Base +0, şi astfel se obţine o comunicaţie la fel ca pe portul SPP standard. Pentru o comunicaţie pe portul EPP cu un dispozitiv compatibil, datele trebuie scrise la adresa Base +4 şi portul va genera toate semnalele de sincronizare necesare. Datele recepţionate se citesc din acelaşi registru. Dacă se doreşte trimiterea unei adrese aceasta trebuie scrisă la adresa Base+3.

8.6 Portul ECP (Enhanced Capabilities Port)

Este un alt tip de port paralel de mare viteză realizat de firma Microsoft în colaborare cu Hewlett-Packard, oferind performanţe îmbunătăţite dar necesită o logică hardware specială. Spre deosebire de EPP, portul ECP nu a fost gândit pentru periferice care lucrează cu sistemele portabile prin portul paralel; scopul său fiind acela de a permite ataşarea ieftină a unei imprimante de foarte mare performanţă. Mai mult portul ECP necesită utilizarea unui canal DMA, ceea ce la portul EPP nu este specificat, astfel că pot apărea conflicte cu alte dispozitive care utilizează transferul DMA. Cele mai multe calculatoare au capacitatea de a lucra în modul ECP sau EPP. De obicei, portul EPP este o soluţie bună pentru perifericele care utilizează porturile.

De asemenea se utilizează un buffer de tip FIFO pentru transferul şi recepţia datelor.

Pini cuplorului de comunicaţie pentru portul paralel în modul ECP.

Pin	Semnalele SPP	Semnalele ECP	IN/OUT	Funcţia
1	Strobe	HostCLK	Out	“0” – indică că datele sunt valide pentru a fi preluate
2-9	Data 0-7	Data 0-7	IN/OUT	Busul de date bidirecţional
10	Ack	PeriphCLK	In	“0” – indică că datele de la dispozitiv sunt valide
11	Busy	PeriphAck	In	“1” – indică că se transmit date “0” – ciclul de comandă

12	Paper out	NAckReverse	In	“0” - Ack pentru dispozitiv
13	Select	X-Flag	In	Extensibility Flag
14	Auto Linefeed	Host Ack	Out	“1” – se transmit date “0” – ciclu de comandă
15	Error	Periph Request	In	“0” – datele de la dispozitiv sunt disponibile
16	Initialize	nReverseRequest	Out	“0” – se recepţionează datele de la periferic “1” – se transmit date la periferic
17	Select Printer	1284 Active	Out	“1” – 1284 transfer mode
18-25	Ground	Ground	GND	

Liniile HostAck şi PeriphAck indică dacă semnalele de pe liniile de date sunt date sau comenzi.

8.6.1 Regiştrii modului ECP

Adresa	Tipul registrului	Read /write
Baza +0	Date (SPP)	Write
	Adresa FIFO (ECP)	Read /Write
Base +1	Stare (SPP, ECP)	Read/ Write
Base + 2	Control (SPP, ECP)	Read/ Write
Base + 400h	Date FIFO (modul FIFO)	Read /Write
	Date FIFO (modul ECP)	Read /Write
	Test FIFO (Modul Test)	Read /Write
	Configurare regiştri A (modul configurare)	Read /Write
Base + 401h	Configurare regiştri B	

	(modul configurare)	
Base + 402h	Extended Control Register (toate modurile)	Read /Write

Cel mai important registru de la portul ECP este registrul ECR (Extended Control Register). Acest registru setează modul de operare al portului ECP plus stările memoriei FIFO. Setarea acestui registru este următoarea:

BIT		
7 : 5	Select Curent mode of Operation	
	000	Standard mode
	001	Byte Mode
	010	Paralel port FIFO Mode
	011	ECP FIFO mode
	100	EPP mode
	101	Reserved
	110	FIFO test mode
	111	Configuration mode
4	ECP Interrupt Bit	
3	DMA Enable Bit	
2	ECP Service Bit	
1	FIFO full	
0	FIFO empty	

Cei mai semnificativi biţi din ECR setează modul de operare. Sunt posibile 7 moduri dar nu toate porturile suportă toate modurile. De exemplu EPP mode nu este disponibil la toate porturile .

Prin regiştri de configurare A şi B se poate configura modul de utilizare al portului ECP, tipul semnalului de întrerupere, numărul de biţi de date, întreruperea pe care să se activeze, numărul canalului DMA etc.

8.7 Portul SPP standard

În PC comunicaţia paralelă se face prin intermediul a trei porturi de comunicaţie, denumite LPT1, LPT2 şi LPT3. Zona de date BIOS conţine, începând de la adresa 0:0408h o listă a adreselor de bază corespunzătoare cuploarelor paralele. Adresele pentru LPT1 şi LPT2 sunt:

LPT1 – 378h…37fh

LPT2 – 278h…27fh

Asupra acestor porturi se poate opera direct, prin instrucţiuni de intrare –ieşire (in, out), sau prin intermediul întreruperii BIOS 17h.

Utilizarea comunicaţiei paralele se poate face prin interogare,(polling), sau prin întreruperi. Liniile de întrerupere folosite, corespunzătoare controlorului de întreruperi 8259A, respectiv tipurile corespunzătoare ale întreruperilor în PC, sunt

LPT1 – IRQ7 – INT 0Fh

LPT2 – IRQ5 – INT 0Dh

8.8 Regiştrii asociaţi unui cuplor de comunicaţie paralelă

Regiştrii pentru cuplorul de comunicaţie paralelă LPT1 sunt:

Adresa 378h – registrul de date al cuplorului paralel

Scriere: conţine codul ASCII trimis imprimantei;

Citire: conţine ultimul caracter trimis.

Adresa 37Ah Citire / scriere – registrul de control al imprimantei

7	6	5	4	3	2	1	0
0	0	0	IRQ	SLCT IN	INIT	AUTO LF	STROB

Bit:

0 - Strob, este 1 la transmiterea unui octet;

1 - AUTO LineFeed, 1 cauzează trimiterea automată a unui LF după un CR;

2 - INIT, iniţializare, 0- resetează imprimanta;

3 - SLCT IN, selecţie, 1 selectează imprimanta;

4 - IRQ Enable, validare întreruperi, 1 permite generarea unei întreruperi hardware când ACK este 0;

5,6,7 sunt 0 (sau 1 depinde de tipul placi de bază).

Adresa 379h Citire – registrul de stare al imprimantei

7	6	5	4	3	2	1	0
BUSY	ACK	PE	SLCT	ERROR	0	0	0

Bit: 0,1,2 – sunt 0

3 – ERROR, când e 0 imprimanta semnalizează o eroare

4 – SLCT, 1: imprimanta e selectată

5 – PE, 1: lipsă hârtie în imprimantă

6 – ACK, 0 : este permisă trimiterea următorului caracter

7 – BUSY, 0 – buffer–ul imprimantei este plin, imprimanta este decuplată sau a apărut o altă eroare

8.9 Întreruperea BIOS 17h

pune la dispoziţie următoarele servicii pentru interfaţă paralelă

AH = 00h – imprimarea unui caracter

La apel:

AL – codul ASCII al caracterului;

DX – numărul imprimantei (0,1 sau 2).

La revenire

AH – starea imprimantei

Bit 0 – 1: timeout;

Bit 1,2 – nefolosiţi;

Bit 3 – 1: eroare I/O;

Bit 4 – 0: imprimantă decuplată, 1: imprimantă selectată;

Bit 5 – 1: lipsă hârtie în imprimantă;

Bit 6 – 1:ACK;

Bit 7 – 1: imprimantă ready;

AH = 01h Iniţializarea portului paralel

La apel:

DX – numărul imprimantei;

La revenire:

AH – starea imprimantei (la fel ca la funcţia 0).

AH = 02h citirea stării imprimantei

La apel:

DX – numărul imprimantei;

La revenire:

AH – starea imprimantei(la fel ca la funcţia 0).

Sincronizarea pe portul SPP se poate realiza în 5 paşi (de exemplu sincronizarea cu imprimanta):

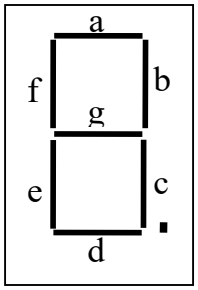
1. Se scriu datele în portul de date;
2. Se verifică dacă imprimanta este ocupată. Dacă este ocupată ea nu va recepţiona datele şi astfel datele scrise vor fi pierdute;
3. Se trece pinul Strobe pe “0”. Astfel imprimanta va prelua corect datele de pe liniile de date;
4. După aproximativ 5 microsecunde se trece pinul Strobe în “1”;
5. Se verifică Ack de la periferic.

8.10 Probleme propuse

Problema 1

Să se scrie un program care afişează pe un display conectat la portul paralel toate valorile de la 0 la F (număr în hexa) în ordine crescătoare. Se va realiza direct scrierea în regiştrii cuplorului de comunicaţie.

Obs: - valorile „B" şi „D" se vor afişa pe display cu litere mici pentru a nu se



confunda cu cifrele „8" şi „0".

- între display şi portul paralel sunt realizate următoarele conexiuni, biţii de date D0-D7 ai portului paralel sunt conectaţi la câte o celulă (led) al display-ului în felul următor:

Pin	Descriere		Celulă display
Pin 2	(Bit de date 0)	⇨	a
Pin 3	(Bit de date 1)	⇨	b
Pin 4	(Bit de date 2)	⇨	c
Pin 5	(Bit de date 3)	⇨	d
Pin 6	(Bit de date 4)	⇨	e
Pin 7	(Bit de date 5)	⇨	f
Pin 8	(Bit de date 6)	⇨	g
Pin 9	(Bit de date 7)	⇨	punct
Pin 25	(masa)	⇦⇨	GND

Problema 2

Să se scrie un program de comandă a unui motor pas cu pas pe portul paralel.

Sunt realizate următoarele legături între motor şi portul paralel:

- comandă înfăşurarea 1 a motorului - la pinul 2 al portului paralel(bit de date 0 în reg. 378h);
- comandă înfăşurarea 2 - la pinul 3 (bit de date 1 în registrul 378h);
- comandă înfăşurarea 3 - la pinul 4 (bit de date 2 în registrul 378h);
- comandă înfăşurarea 4 - la pinul 5 (bit de date 3 în registrul 378h).

În funcţie de sensul de rotaţie dorit se poate comanda înfăşurările în diferite direcţii. Pentru un cuplu al motorului mai bun se poate comanda la un moment dat 2 înfăşurări (comandă cu micro-păşire).

Exemplu de comandă (comanda cu micro-păşire):

- înfăşurarea 1;
- înfăşurarea 1 şi 2;
- înfăşurarea 2;
- înfăşurarea 2 şi 3;
- înfăşurarea 3;
- înfăşurarea 3 şi 4;
- înfăşurarea 4;
- înfăşurarea 4 şi 1.

Problema 3

Să se realizeze un program de comunicaţie între două calculatoare utilizând cuplorul paralel de comunicaţie.

Observaţii

1.Comunicaţia se va realiza utilizând un cablu de comunicaţie paralelă cu 11 fire care realizează următoarele legături:

Pin	Descriere		Pin	Descriere
Pin 2	(Bit de date 0)	⇨	Pin 15	(Eroare)
Pin 3	(Bit de date 1)	⇨	Pin 13	(Select)
Pin 4	(Bit de date 2)	⇨	Pin 12	(Paper end)
Pin 5	(Bit de date 3)	⇨	Pin 10	(Acknowledge)
Pin 6	(Bit de date 4)	⇨	Pin 11	(Busy)
Pin 15	(Eroare)	⇦	Pin 2	(Bit de date 0
Pin 13	(Select)	⇦	Pin 3	(Bit de date 1)
Pin 12	(Paper end)	⇦	Pin 4	(Bit de date 2)
Pin 10	(Acknowledge)	⇦	Pin 5	(Bit de date 3)
Pin 11	(Busy)	⇦	Pin 6	(Bit de date 4)
Pin 25	(masa)	⇦⇨	Pin 25	(masa)

2. Înainte de transmiterea datei se va realiza o sincronizare între cele două calculatoare.

3. Cuplorul de comunicaţie inversează bitul Busy prin hard

Problema 4

Să se scrie un program care inditectează întreruperea serviciilor pentru imprimantă (INT 17h), cu scopul salvării într-un fişier a tuturor datelor trimise spre imprimantă.

9 Magistrale

9.1 Scopul lucrării

Lucrarea îşi propune familiarizarea studenţilor cu modul de comunicaţie a unui calculator compatibil IBM cu plăcile de extensie conectate pe magistralele ISA sau PCI disponibile în aceste calculatoare. De asemenea sunt prezentate regiştrii asociaţi magistralei PCI pentru identificarea şi comanda cu plăcile de extensie conectate pe magistrala PCI.

9.2 Consideraţii generale

Magistralele reprezintă căile de transmitere a semnalelor între diferitele elemente ale unui sistem de calcul precum şi protocoalele folosite pentru transmiterea informaţiilor. Din acest punct de vedere magistralele se grupează în magistrale paralele şi magistrale seriale. După modul de control al transferului de informaţii, magistralele pot fi *sincrone* sau *asincrone.* La magistralele sincrone toate operaţiile sunt sincronizate de un semnal de ceas şi necesită un număr întreg de perioade de ceas, numite *cicluri de magistrală.* Magistralele asincrone nu utilizează un semnal de ceas iar ciclurile de magistrală pot avea în acest caz orice durată, şi aceasta nu trebuie să fie aceeaşi între toate perechile de dispozitive.

Magistralele permit efectuarea transferului de date între calculator şi diferitele plăci de extensie. Magistralele grupează semnale de tipul linii de date, linii de adresă, linii de comandă /stare, linii de alimentare etc.

Există mai multe magistrale standardizate în interiorul calculatorului.

9.3 Magistrala ISA

Magistrala ISA (Industry Standard Arhitecture) este arhitectura de magistrală utilizată la primul IBM PC în 1982. Iniţial a fost o magistrală pe 8 biţi, dar mai târziu a fost extinsă şa 16 biţi şi utilizată în modelul IBM PC/AT, lansat pe piaţă în 1984. Poate părea uimitor că o arhitectură atât de veche este folosită în zilele noastre în sistemele de înaltă performanţă, dar aceasta se întâmplă din cauza siguranţei, accesibilităţii şi compatibilităţii sale, la care de adaugă faptul că

această veche magistrală continuă să fie mai rapidă decât multe din perifericele pe care le conectăm la ea. Există două versiuni de magistrale ISA, care se deosebesc prin numărul de biţi de date ce pot fi transferaţi simultan: magistrala pe 8 biţi şi magistrala pe 16 biţi. Versiunea pe 8 biţi era utilizată în PC şi XT la 4.77MHz. versiunea pe 16 biţi a fost utilizată în AT şi lucra la 6MHz şi apoi la 8MHz. Ulterior, producătorii au stabilit la 8.33MHz valoarea frecvenţei standard a magistralei ISA, pentru a asigura compatibilitatea cu versiunile vechi. Există sisteme în care magistrala ISA poate funcţiona la frecvenţe mai mari, dar unele adaptoare nu pot face faţă vitezelor de lucru mari.

Redăm în continuare semnificaţia pinilor primului segment al unui slot ISA pe 8 biţi:

PIN	Nume Semnal	PIN	Nume Semnal
B1	Masă	A1	Verificare canal I/O
B2	Reset	A2	Data bit 7
B3	+5Vcc	A3	Data bit 6
B4	IRQ9	A4	Data bit 5
B5	-5Vcc	A5	Data bit 4
B6	DMA Request 2	A6	Data bit 3
B7	-12Vcc	A7	Data bit 2
B8	0 Wait	A8	Data bit 1
B9	+12Vcc	A9	Data bit 0
B10	Masa	A10	Canal I/O ready
B11	Scriere memorie MemW	A11	Activare adresă AEN
B12	Citire memorie MemR	A12	Linie adrese 19
B13	Scriere I/O IOWR	A13	Linie adrese 18
B14	Citire I/O IORD	A14	Linie adrese 17
B15	DMA Acknowledge 3	A15	Linie adrese 16
B16	DMA Request 3	A16	Linie adrese 15

B17	DMA Acknowledge 1	A17	Linie adrese 14
B18	DMA Request 1	A18	Linie adrese 13
B19	Refreş	A19	Linie adrese 12
B20	Clock (8.33MHz)	A20	Linie adrese 11
B21	IRQ 7	A21	Linie adrese 10
B22	IRQ 6	A22	Linie adrese 9
B23	IRQ 5	A23	Linie adrese 8
B24	IRQ 4	A24	Linie adrese 7
B25	IRQ 3	A25	Linie adrese 6
B26	DMA Acknowledge 2	A26	Linie adrese 5
B27	Terminal count	A27	Linie adrese 4
B28	BALE	A28	Linie adrese 3
B29	+5Vcc	A29	Linie adrese 2
B30	Oscilator (14.3MHz)	A30	Linie adrese 1
B31	Masa	A31	Linie adrese 0

Dezavantajul major al acestei magistrale este viteza relativ scăzută. Această limitare a vitezei este o consecinţă a epocii primului sistem PC, la care magistrala de I/O opera la aceeaşi viteză cu magistrala procesorului. În timp ce viteza magistralei procesorului a crescut, magistrala I/O a cunoscut doar ajustări ale vitezei nominale, în principal prin creşterea lăţimii de bandă. Magistrala I/O trebuia să lucreze la o viteză scăzută deoarece imensa majoritate a adaptoarelor instalate funcţionau doar la viteze mai mici.

9.4 Magistrala PCI

Propusă de firma *Intel* in 1992, magistrala locală denumită PCI (Peripheral Component Interconnected) se dorea să permită interconectarea circuitelor rapide pe placa de bază. *Intel* trebuia să modifice arhitectura magistralei locale

pentru a mări performanţele sistemului, ceea ce implica şi modificarea circuitelor interfaţă pentru periferice.

Magistrala PCI urma să se conecteze la magistrala locală a procesorului prin intermediul unui circuit special destinat acestui scop. Astfel, la fiecare schimbare a procesorului şi a magistralei locale, trebuia schimbat doar circuitul de legătură, circuitele de interfaţă cu perifericele nefiind afectate.

Iniţial specificaţia PCI nu prevedea conectori de extensie. *Intel* a actualizat specificaţiile PCI pentru ca aceasta să admită şi conectori de extensie. Astfel, ajungându-se la ce-a mai utilizată arhitectură de magistrală.

Pe magistrala PCI sunt posibile transferuri de 32 sau de 64 biţi, lucrând, în versiunea 2.1 la o frecvenţă de 66 MHz, ceea ce permite obţinerea unei rate maxime de transfer de 264MB/s.

Magistrala PCI este conectată la magistrala sistem a procesorului prin intermediul unui „bridge”, având acces direct la memoria principală astfel încât transferurile între procesor şi memoria cache, respectiv între dispozitivele de I/O şi memoria principală pot avea loc simultan. Astfel pe magistrala PCI se pot conecta dispozitive de viteză ridicată care solicită în măsură redusă procesorul.

9.4.1 Regiştrii magistralei PCI

Pe magistrala PCI fiecare dispozitiv funcţional dispune de un bloc de 64 cuvinte duble rezervat pentru regiştrii de configuraţie. Formatul primelor 16 cuvinte este predefinit de specificaţiile PCI. Aceasta reprezentând antetul de configuraţie al dispozitivului. În specificaţia PCI sunt descrise 2 forme ale antetului. Tipul 0 este specific dispozitivelor dezvoltate pe PCI şi va fi descris în continuare. Tipul 1 este specific punţilor de legătură (bridge) între două magistrale PCI pentru sistemele cu mai multe magistrale PCI.

Pentru tipul 0 registrele din antet sunt utilizate pentru identificarea dispozitivului, controlul funcţiilor sale şi detectarea stării acestuia.

00h – Registru identificare producător / dispozitiv

Acest dublu registru este disponibil numai pentru citire.

Octeţii 1 şi 0 din acest registru (cmps – cei mai puţin semnificativi 16 biţi) identifică producătorul dispozitivului. Valoarea 0FFFFh este rezervată şi este

întoarsă automat de bridge-ul PCI când se încearcă citirea registrului de identificare al producătorului dintr-un dispozitiv inexistent.

Octeţii 3 şi 2 din acest registru (cms – cei mai semnificativi 16 biţi) identifică tipul dispozitivului şi este asignat de producător.

01h – Registru de comandă şi de stare

Octeţii 1 şi 0 din acest registru (cmps 16 biţi) sunt utilizaţi de registrul de comandă. Din cei 16 biţi rezervaţi sunt utilizaţi doar primii 10 biţi ceilalţi fiind rezervaţi pentru dezvoltări ulterioare. Dintre funcţiile care pot fi specificate de aceşti regiştrii sunt:

- validare dispozitive;
- validare acces la memoria dispozitivului (dacă acesta dispune de memorie);
- validează funcţia de „master” a dispozitivului (dacă acesta are această funcţie);
- validare monitorizare cicluri speciale PCI – utilitare pentru transmiterea de mesaje dispozitivelor sclave;
- Validare paritate – când este validat, dispozitivul va raporta erorile de paritate;
- Actualizarea regiştrilor de paletă a culorilor ale unei interfeţe VGA fără confirmarea acceptări datelor – utilă când în sistem se folosea şi un accelerator grafic separat de placa grafică.

Octeţii 3 şi 2 din acest registru (cms 16 biţi) reprezintă registrul de stare care păstrează starea evenimentelor legate de PCI. Acest registru poate fi citit, iar la scriere fiind posibile doar ştergerea valorilor nu şi activarea lor. Acest registru conţine informaţii legate de frecvenţa de funcţionare, raportarea parităţii, abandonarea unei comunicaţii dacă este în modul slave, semnalarea unei erori de sistem, etc.

02h – Registru identificare revizie şi cod clasă

Primul octet (cmps octet) conţine o valoare specificată de producător care reprezintă numărul de revizie al dispozitivului.

Ceilalţi 3 octeţi reprezintă codul de clasă care identifică clasa de bază, subfuncţia şi interfaţa de programare a dispozitivului. Clasa de bază a

dispozitivului reprezintă funcţia acestuia şi este codificată pe un octet. Următorul octet reprezintă o subclasă a dispozitivului şi în anumite cazuri ultimul octet reprezintă o interfaţă de programare.

03h – Registrul tipului de antet, dimensiune cache, contor întârziere

Primul octet reprezintă dimensiunea cache-ului dacă este utilizat, cel de-al doilea reprezintă contor de întârziere introdus de dispozitiv. Al treilea bit reprezintă formatul antetului de configuraţie. Actualmente fiind definite 2 formate ale antetelor.

04h-09h – Registre de adrese de bază

6 regiştrii pe 32 biţi fiecare în care se pot specifica adresa care se doreşte accesată în dispozitiv

0Bh – Registru identificare producător subsistem, identificare subsistem

Acest dublu registru este disponibil numai pentru citire.

Octeţii 1 şi 0 din acest registru (cmps – cei mai puţin semnificativi 16 biţi) identifică producătorul subsistemului utilizat de dispozitiv în cazul acesta foloseşte un subsistem.

Octeţii 3 şi 2 din acest registru (cms – cei mai semnificativi 16 biţi) identifică tipul subsistemului şi este asignat de producător.

0Ch – Registru adresă de bază memorie ROM

În cazul în care dispozitivul foloseşte o memorie ROM de extensie în acest registru se specifică adresa acestei memorii pentru sistem.

9.5 Magistrala AGP

Magistrala AGP (Accelerated Graphics Port) este o nouă interconexiune pentru acceleratoarele grafice din sistemele bazate pe procesoare Pentium utilizate pentru grafică 3D şi redarea secvenţelor video. Această tehnologie îmbunătăţeşte performanţele sistemului punând la dispoziţie o cale rapidă între controlerul grafic şi memorie sistem. Astfel se permite controlerului grafic să facă accese direct în memoria sistem în timpul interpretării grafice, fără să le mai încarce în prealabil în memoria video locală. Segmentele din memoria sistem care pot fi accesate de controlerul grafic pot fi rezervate în mod dinamic de către sistemul

de operare. Astfel se permit rezoluţii mari ale ecranului şi se elimină restricţiile de dimensiune pe care memoria video locală le impunea asupra texturilor care trebuiau afişate.

AGP este un port şi nu o magistrală, deoarece la o magistrală se pot conecta mai multe dispozitive, în timp ce pe AGP există o singură conexiune doar între placa video şi procesor. AGP este o interfaţă de 64 biţi care poate funcţiona la 66 Mhz (AGP 1X) şi se bazează pe extensia la 64 biţi a specificaţiei PCI 2.1.

Avantajul AGP este că poate funcţiona la viteza maximă a magistralei sistem în comparaţie cu PCI care funcţiona la jumătatea acestei viteze. Pe lângă dublarea vitezei magistralei, AGP a definit şi un mod 2X care utilizează un protocol special care permite dublarea volumului de date care se transmit pe port la aceeaşi frecvenţă. Creşterea este obţinută prin transferarea datelor atât pe frontul crescător cât şi pe cel descrescător al ceasului de 66MHz. Ajungându-se astfel la o rată de transfer de 533Mb/s.

9.6 Circuitul EEPROM 93C46

Se consideră o placă conectată pe magistrala ISA care răspunde la o adresă de forma 110A.AAAA, deci în domeniul 0C000h—0D000h unde sunt cuprinse modulele suplimentare de BIOS.

Placa conţine o memorie de tip EPROM de 2ko care poate fi doar citită şi răspunde la adresa de mai sus cu linia A11 pe 0 şi un circuit de memorie secvenţială de tip EEPROM care răspunde tot la adresa de mai sus dar cu A11 pe 1.

Circuitul de memorie EEPROM (93C46) este organizat ca 64 de cuvinte a câte 16 biţi fiecare. Dintre principalele caracteristici ale acestui tip de memorie EEPROM aminti:

- peste 10000 operaţii de ştergere / scriere;
- timp de menţinere a informaţiei de minim 10 ani;
- număr nelimitat de accese de citire;
- accesare pe cuvânt;
- necesită o singură alimentare chiar şi pentru operaţia de scriere.

Pini circuitului EEPROM 93C46 sunt:

- Pin 1: CS (cip select) – semnal selecţie circuit;
- Pin 2: CLK (clock) – semnal sincronizare circuit cu dispozitivul de comandă;
- Pin 3: DI (data input) – pin disponibil pentru înscrierea datelor secvenţial;
- Pin 4: DO (data output) – pin disponibil pentru citirea secvenţială a datelor din circuit.

Acest circuit permite scrierea unei singure locaţii sau a întregului circuit iar citirea se face locaţie cu locaţie. Pentru selectarea diferitelor funcţii circuitul recunoaşte 7 comenzi:

Comandă	Acţiune
0000.XXXX	Inhibare scriere
0001.XXXX	Scrie tot EEPROM-ul cu aceeaşi valoare
0010.XXXX	Şterge tot EEPROM-ul
0011.XXXX	Activare scriere
01AA.AAAA	Scriere locaţie
10AA.AAAA	Citire locaţie
11AA.AAAA	Ştergere locaţie

Unde AA.AAA reprezintă adresa locaţiei de memorie care se doreşte accesată iar XXXX reprezintă faptul că acei biţi nu sunt luaţi în considerare.

Astfel după **operaţia de resetare** a circuitului care constă în selectarea circuitului, punerea pinului DI pe 1 şi setarea şi resetarea clock-ului, circuitul interpretează următorii 8 biţi ca fiind comanda şi adresa astfel: în cazul comenzilor care specifică o adresă fixă primii 2 biţi reprezintă comanda iar următorii 6 biţi (2^6=64 locaţii) reprezintă adresa locaţiei. Pentru comenzile care nu specifică o locaţie anume primii patru biţi reprezintă comanda iar următorii sunt neglijaţi.

Comanda de ştergere – după primirea **comenzii de ştergere** şi a locaţiei dorite, locaţia respectivă este înscrisă cu valoarea „FFFF”. Aceasta este singura posibilitate prin care se poate înscrie în circuit o valoare mai mare în hexayecimal decât cea existentă înainte. Deci înainte de scrierea unei noi valori,

pentru a putea fi siguri că se înscrie valoarea dorită, locaţia respectivă trebuie în prealabil ştearsă.

La recepţionarea **comenzii de scriere**, după citirea octetului de comandă, circuitul aşteaptă introducerea unui cuvânt de 16 biţi care cuprinde valoarea care se doreşte a fi scrisă în locaţia respectivă, cuvânt care se introduce secvenţial, începându-se cu bitul cel mai semnificativ şi terminându-se cu bitul cel mai puţin semnificativ.

La recepţionarea **comenzii de citire** a unei locaţii, după octetul respectiv circuitul pune pe linia D0, disponibil pentru următoarele tacte, cuvântul ce se află în locaţia specificată începând cu bitul cel mai semnificativ.

Pentru ultimele 4 comenzi, cale care încep cu 00, circuitul aşteaptă introducerea tuturor celor 8 biţi din comandă dar ultimi 4 biţi nu îi ia în considerare. Pentru comanda „0001” (Scrie tot circuitul la fel) circuitul aşteaptă introducerea următorilor 16 biţi care reprezintă valoarea ce se doreşte a fi scrisă, după care se înscrie valoarea respectivă în toate cele 64 de locaţii. Comanda „0010” realizează înscrierea în toate locaţiile memoriei a valorii „FFFF”. Ultimele două comenzi „0000” şi „0011” realizează blocarea / deblocarea circuitului de memorie în cea ce priveşte operaţiile de scriere.

9.7 Probleme propuse

Problema 1: Se consideră un circuit de memorie de tip 93C46 conectat pe o magistrală ISA. Să se scrie un program care permite scrierea şi citirea tuturor celor 64 de locaţii de memorie din acest circuit.

Pinii circuitului EEPROM 93C46 sunt legaţi la magistrala de date conform următorului tabel:

Pin EEPROM	**Linia de date a magistralei**
Pin 1: CS (cip select)	D1 (linia de date 1)
Pin 2: CLK (clock)	D2 (linia de date 2)
Pin 3: DI (data input)	D0 (linia de date 0)
Pin 4: D0 (data output)	D4 (linia de date 4)

Pe tot parcursul scrierii şi citirii linia de date D3 trebuie ţinută pe 0.

O procedură de scriere în circuitul de memorie ar arăta sub forma:

```
#define ADRSEGM 0xD000

void scriere(unsigend char aValue)
{
   unsigend int adrSegm = ADRSEGM;
   aValue &= 0xf7;              //forţăm D3 pe 0 tot
                                //timpul
   adrSegm |= 0x0080;           // A11 pe 1
   pokeb(adrSegm, 0, aValue);   //se memorează datele
                                //de intrare în buffer
   adrSegm = ADRSEGM;           //A11 pe 0
   pokeb(adrSegm, 0, aValue);   // datele din buffer
                                // sunt disponibile la
                                //ieşire
}
```

O procedură de citire din circuitul de memorie ar avea următoarea formă:

```
unsigned char citire(void)
{
 int value;
 unsigend int adrSegm = ADRSEGM;
   adrSegm |= 0x0080;          // A11 pe 1
   value = peek(adrSegm, 0);   // memorăm datele
                               //de ieşire în buffer
   value &= 0x10;              //păstrăm doar D0
   return (value);
}
```

Anexa 1

```
//desenează un dreptunghi in modul grafic 12h si 13h
//folosind întreruperea int 10h
//la apăsarea tastelor „+” sau „-„ se modifica culoarea
//la apăsarea tastei „d” se modifica modul de scriere al
//unui pixel prin accesul direct la memoria video
//la apăsarea tastei „i” se modifica modul de scriere al
//unui pixel folosind int 10h
//la apăsarea tastei „c” se afişează pe ecran toate
//culorile modului curent
//la apăsarea tastei „2” se modifica in modul grafic 12h
//la apăsarea tastei „3” se specifica modul grafic 13h
//la apăsarea tastei „ESC” se termină aplicaţia

#include <dos.h>
#include <conio.h>
#define ESC 27
short mod_acces = 0;

//seteaza un mod video specificat
void SetVideoMode(unsigned short aVideoMode)
{
   union REGS r;
   r.h.ah = 0x00;
   r.h.al = aVideoMode;
   int86(0x10,&r,&r);
}
//citeste modul video curent folosind int 10h
unsigned short GetVideoMode(void)
{
   union REGS r;
   r.h.ah = 0x0F;
   int86(0x10,&r,&r);
   return r.h.al;
}
//afiseaza un pixel pe ecran folosind int 10h sau scrierea
//direct in memorie in functie de variabila mod_acces
void PutPixel(int aRand,int aColoana, short aCuloare)
{
   union REGS r;
   if (mod_acces == 0)
   {
      if (GetVideoMode() == 0x12)
         pokeb(0xA000,640*aRand+aColoana,aCuloare);
      else
```

```
         pokeb(0xA000,320*aRand+aColoana,aCuloare);
   }else
   {
      r.h.ah = 0x0c;
      r.h.bh = 0;
      r.x.dx = aRand;
      r.x.cx = aColoana;
      r.h.al = aCuloare;
      int86(0x10,&r,&r);
   }
}
//deseneaza un dreptunghi pe ecran la coordonatele
//(aX_i,aY_i)-(aY_f,aY_f)
void dreptunghi(int aX_i, int aX_f, int aY_i, int aY_f,
short aCuloare)
{
   for(int i = aX_i; i <= aX_f; i++)
      for(int j = aY_i; j <= aY_f; j++)
         PutPixel(i, j, aCuloare);
}

void main(void)
{
   short mode = GetVideoMode();
   SetVideoMode(0x12);
   short color = 1, gata = 0;
   do{
        if(gata == 0)
        {
           if(GetVideoMode() == 0x12)
           dreptunghi(0,199,0,319,color);
           else
              dreptunghi(0,199,0,319,color);
        }
        gata = 1;
        switch(getch())
        {
           case ESC:gata = 2; break;
           case '+':
           {
                color++;
                color %= 16;
                gata = 0;
           }; break;
           case '-':
           {
                color = (color == 0) ? 16 : color-1;
                gata = 0;
```

```
            }; break;
            case 'c':
            {
                 int y_start=0,y_stop=0;
                 if(GetVideoMode() == 0x12)//are 12
                                           //culori,640*480
                     for(int g = 0; g < 16; g++)
                     {
                        y_start = y_stop;
                        y_stop = y_start + 480 / 16;
                        dreptunghi(y_start,y_stop,0,639,g);
                     }
                 else
                 {//pentru modul 13h care are 256
                      //culori,320*200
                     for(int g = 0; g < 200; g++)
                     {
                        y_start = y_stop;
                        y_stop = y_start + 1;
                        dreptunghi(y_start,y_stop,0,319,g);
                     }
                }
             }; break;
          case 'd': mod_acces = 0; gata = 0; break;
          case 'i': mod_acces = 1; gata = 0; break;
          case '2': SetVideoMode(0x12); gata = 0; break;
          case '3': SetVideoMode(0x13); gata = 0; break;
        }
      }while (gata != 2);
   SetVideoMode(mode);
}
```

Anexa 2

```
// program de test al rutinei de tratare a erorii
// critice INT 24h
#pragma inline
#include <dos.h>
#include <stdio.h>

void interrupt(*old_24)(...);
unsigned int old_24_seg;
unsigned int old_24_off;
unsigned semafor = 0;

void interrupt new_24(...)
{
   Semafor = 1;

asm {
       mov [semafor],1
       pop ax;
       pop ax;
       pop ax;
       mov bp, sp;
       mov ax, [bp+18];
       mov word ptr cs:[Intoarce], ax;
       mov ax, [bp+20];
       mov word ptr cs:[Intoarce+2], ax;
       pop ax;
       pop bx;
       pop cx;
       pop si;
       pop di;
       pop bp;
       pop ds;
       pop es;
       pop ax;
       pop ax;
       popf;
       STC;
       mov al,3;
       jmp dword ptr cs:[Intoarce];
       Intoarce dw 0,0
    }
}
```

```
void main(void)
{
   union REGS r;
   unsigned g = 0;
   char dir[6] = {'b',':','\\','a','s',0};
   old_24 = getvect(0x24);  //citesc vechiul handler
   old_24_off = FP_OFF(old_24);
   old_24_seg = FP_SEG(old_24);
   printf("\nInstalare handler nou int 24");
   setvect(0x24,new_24);       //setez noul handler
   do
   {
      asm{
           mov ah, 39h;
           lea dx, dir
           int 21h
         }
      if (semafor == 1)
      {
          printf("\nEroare   disk   int   21   -ESC   pentru
                  parasire");
          r.h.ah = 0;
          int86(0x18,&r,&r);
          if (r.h.ah == 1)
              g = 1;
      }
   }while (g == 0);
   setvect(0x24, old_24); //restaurez vechiul handler
}
```

Anexa 3

```
// program de indirectare întrerupere 0Ch pentru
// emisie şi recepţie caractere pe portul serial
#include <stdio.h>
#include <conio.h>
#include <dos.h>

unsigned int re = 0,em = 0,er = 0,modem = 0;
char q1 = 0,w1 = 0;
void interrupt(*old_0c)(...);

void interrupt newInt_0c(...)
{
  q1=inportb(0x3fa);   //registrul cauza a întreruperii
  q1=q1 & 0x07;
  switch (q1)
  {
     case 0x00:{//eroare
                  w1=inportb(0x3fd);
                  er++;
               };break;
     case 0x02:{//emisie
                  outportb(0x3f8,'A');
                  em++;
               };break;
     case 0x04:{//receptie
                  w1=inportb(0x3f8);
                  re++;
               };break;
     case 0x06:{//schimbarea stare modem
                  w1=inportb(0x3fe);
                  modem++;
                  outportb(0x3f8,'b');
               };break;
  }
  outport(0x20,0x20);
}
void main(void)
{
  char a;
  clrscr();
  old_0c=getvect(0x0c);

  outportb(0x3fb,0x80);  //DLAB=1-programam rata de
                         //transmisie
```

```
  outportb(0x3f8,0x10);  //110 baud
  outportb(0x3f9,0x04);

  outportb(0x3fb,0x07);  //DLAB=0, 8biti de date,2 de
                         //stop, fara paritate

  printf("Activam DTR,RTS\n");
  outportb(0x3fc,0x0b);  //activam DTR,RTS si OUT2

  printf("Setam intreruperea\n");
  setvect(0x0c,newInt_0c);

  printf("Activam intreruperea\n");
                 //activare intrerupere emisie /receptie

  outportb(0x3f9,0x03);   //interupere la receptie date si
                          //bufer emisie gol

  printf("Activam 8259\n");
  a = inportb(0x21);
  a = a & 0xef;
  outportb(0x21,a);       //activare IRQ4 in 8259a

  union REGS r;
  char c2;
  printf( "Bucla\n");
  do
  {
     if (kbhit())
     {
       a = getch();
       outportb(0x3f8,a);
     }
     delay(20);
     gotoxy(1,7);
     printf("Am receptionat %c\n",w1);
     printf("Receptie = %d  Emisie=%d \n",re,em);
     printf("eroare = %d  modem=%d \n",er,modem);
  } while (a != 0x0d);
  a = inportb(0x21);
  a = a | 0x10;
  outportb(0x21,a);       //dezactivare IRQ4 in 8259a

  setvect(0x0c,old_0c);
}
```

Bibliografie selectivă

[1] Hennessy, Patterson, *Computer Architecture. A Quantitative Approach*, Morgan Kaufmann Series, Ediţia a 3-a, 2003

[2] Hans Peter Messmer, *The Indispensable PC Hardware Book*, Ediţia a 3-a, Addison-Weslez Publisher, 1994

[3] Scott Mueller, *PC Depanare şi Modernizare*, ediţia a 2-a, Editura Teora Bucureşti, 1997

[4] Z.F. Baruch, *Sisteme de Intrare Ieşire ale calculatoarelor*, Editura Albastră Cluj Napoca, 2000

[5] *Interfacing The Standard Parallel Port*, www.beyondlogic.org/spp/parallel.htm

[6] *Interfacing the Serial Port*, www.beyondlogic.org/serial/serial.htm

[7] Intel Corporation, *AGP Interface Specification*, http://members.datafast.net.au/dft0802/specs/agp30.pdf

[8] Intel, Compaq, HP,... *Universal Serial Bus Specification*, 2000, www.intel.com/technology/usb/spec.htm

[9] Leo F. Dozle, *Computer Peripherals*, Ediţia a 2-a, Prentice Hall, 1999

[10] William Buchamen, *PC Interfacing, Communications and Windows Programming*, Addison-Wesley Longman, Prima ediţie, 1999

[11] A.S. Tanenbaum, *Operating Systems – Design and Implementation*, Prentice Hall Software Series, Ediţia a 3-a, 2006