Microprocesoare
Îndrumar de laborator

Dan NICULA [1]
Alexandru PIUKOVICI Radu GĂVRUŞ

August, 1999

[1]Universitatea TRANSILVANIA Braşov, Catedra de Electronică şi Calculatoare, Email: nicula@vega.unitbv.ro

Prefaţă

Prima parte a laboratorului cuprinde aplicaţii cu macheta didactică *MPF1-B Microprofessor*, echipată cu microprocesor Z80. Lucrările propuse au mai multe scopuri:

Exersarea unui limbaj de asamblare prin studierea tipurilor de instrucţiuni, modurilor de adresare şi a modului de transformare a instrucţiunilor de asamblare în cod maşină.

Studierea unei scheme minimale a unui sistem cu microprocesor şi a modului de interfaţare dintre microprocesor, memorie şi periferice.

Studierea microprocesorului ca sistem digital prin vizualizarea cu osciloscopul şi analizorul logic a semnalelor generate de acesta.

Partea a doua a laboratorului particularizează informaţiile generale dobîndite la cursul de "Microprocesoare" pentru familia de microprocesoare Intel 80x86. Sînt necesare două justificări:

De ce 80x86? Pentru că microprocesoarele compatibile Intel 80x86 echipează majoritatea calculatoarelor personale existente în prezent pe piaţă. Calculatoarele cu microprocesoare 80x86 pot rula trei sisteme de operare foarte răspîndite: DOS, Windows, Linux. Aplicaţiile care rulează pe aceste calculatoare necesită de multe ori module scrise în limbaj de asamblare. Totodată, un număr mare de alte sisteme digitale (plăci de achiziţie şi prelucrare de date, sisteme de automatizare şi control) conţin microprocesoare din această familie.

De ce 8086 şi nu ultima generaţie Intel? Pentru că aceste lucrări de laborator au scopul de a prezenta elementele de bază ale utilizării unui microprocesor şi de a oferi teme pentru exersarea programării în limbaj de asamblare. Procesorul 8086 reprezintă un "standard" al arhitecturilor 80x86. Folosirea facilităţilor existente la cei mai noi membrii ai familiei (gestiunea memoriei, multitasking, instrucţiuni multimedia) va fi tratată la disciplinele "Aplicaţii ale calculatoarelor", "Sisteme de operare" şi "Multimedia".

Ce este un sistem cu microprocesor?

Un sistem cu microprocesor, deseori numit "calculator", conţine trei mari blocuri constitutive, conectate aşa cum se prezintă în figura 1. Aceste blocuri sînt:

- Unitatea centrală de prelucrare, microprocesorul (Central Processing Unit - CPU);
- Memoria;
- Dispozitivele de intrare/ieşire (Input/Output = I/O).

Unitatea centrală de prelucrare implementată sub forma unui chip microprocesor, este piesa centrală a oricărui sistem de calcul. CPU realizează prelucrări numerice (adunări,

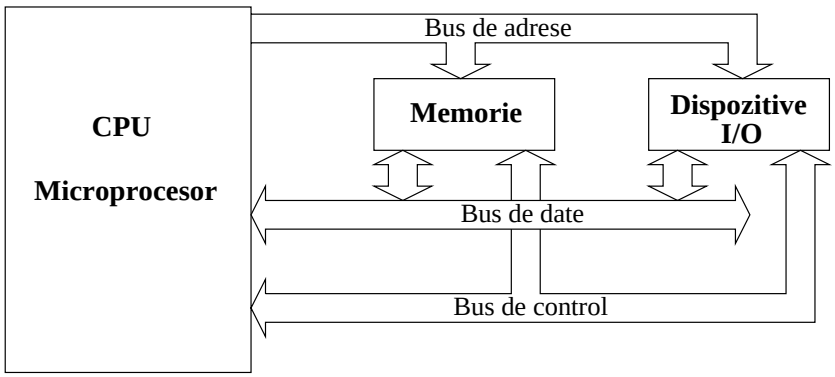

Figura 1: Schema bloc a unui sistem cu microprocesor.

scăderi, înmulţiri, etc.) şi operaţii logice asupra fluxului de date. Operaţiile realizate de CPU sînt controlate printr-o *secvenţă de instrucţiuni* grupate într-un *program*. Programele şi datele sînt înmagazinate în memorie. CPU citeşte cîte o instrucţiune din memorie, o decodifică şi apoi o execută. În procesul de execuţie, CPU poate citi/scrie date din/în memorie. Un microprocesor se structurează în două blocuri funcţionale: *calea de date* şi *calea de control*. Elementele esenţiale din calea de date sînt *registrele* şi *unitatea logico-aritmetică* (Arithmetic Logic Unit = ALU).

Registrele reprezintă locaţii de memorie temporare aflate în interiorul CPU. Registrele sînt fie *dedicate* (*program counter*, *status register*), fie *generale*.

Unitatea logico-aritmetică este unitatea care realizează prelucrarea efectivă a datelor. Operaţiile realizate de ALU sînt fie logice (operanzi interpretaţi ca o mulţime de biţi), fie aritmetice (operanzi interpretaţi ca numere exprimate în baza doi).

Calea de control coordonează activitatea microprocesorului şi realizează secvenţialitatea execuţiei programelor. Circuitele din calea de control decodifică instrucţiunea şi lansează comenzi pentru unităţile interne şi externe în scopul executării acesteia.

Memoria înmagazinează programele şi datele. Programul de iniţializare şi gestionare a resurselor sistemului (monitor, sistem de operare) este menţinut într-o memorie ROM. Restul spaţiului de memorie este ocupat de memorie RAM.

Dispozitivele de intrare/ieşire denumite şi *periferice*, reprezintă mijloacele de comunicare ale microprocesorului cu lumea exterioară. Tastatura, monitorul sau imprimanta sînt controlate de către CPU prin intermediul porturilor de intrare/ieşire.

Magistralele de adrese, date şi control interconectează unitatea centrală cu memoria şi dispozitivele I/O. Pe *bus-ul de date* se transferă bidirecţional informaţii codificate binar, interpretate ca date sau ca instrucţiuni. *Bus-ul de adrese* unidirecţional este folosit de CPU pentru a transmite adrese către memorie şi dispozitive I/O. Pe *bus-ul de control* se transmit comenzi de la CPU spre memorie şi spre dispozitivele I/O.

Întreruperile sînt situaţii în care microprocesorul îşi suspendă execuţia secvenţială a programului pentru a deservi apelul venit de la un periferic. De obicei, într-un sistem există mai multe dispozitive care pot lansa cereri de întrerupere. Pentru a putea fi servite toate, întreruperile trebuiesc ierarhizate prin asocierea unor priorităţi.

Accesul direct la memorie (Direct Memory Access - DMA) reprezintă o soluţie de transfer rapid a datelor de la un periferic în memorie fără ca acestea să mai treacă prin microprocesor. Prin utilizarea DMA, CPU predă controlul magistralelor către un dispozitiv periferic care controlează transferarea datelor direct în memoria sistemului.

Modurile de adresare reprezintă totalitatea modalităţilor de determinare a adreselor pentru accesarea memoriei externe.

Contribuţia autorilor este următoarea:

Dan Nicula (coordonator) - Partea a II-a, Lucrările 6, 7, 8, 9, 10, 11 şi anexele

Alexandru Piukovici - Partea I, Lucrările 1, 2 şi 3

Radu Găvruş - Partea I, Lucrările 4 şi 5

Fişierele cu programele propuse ca exemple în cadrul laboratoarelor pot fi accesate prin ftp anonim de la adresa ftp://vega.unitbv.ro/pub/microp. Orice fel de observaţie referitoare la acest îndrumar poate fi făcută prin e-mail nicula@vega.unitbv.ro.

Cuprins

Partea I

Microprocesorul Z80

Lucrarea 1

Prezentarea machetei cu microprocesor Z80

Această lucrare prezintă microprocesorul Z80 şi sistemul *MPF1-B Microprofessor*. Sînt prezentate funcţiile tastelor, facilităţile oferite de programul monitor, harta memoriei şi adresele porturilor de comandă.

1.1 Microprocesorul Z80

Microprocesorul Z80 este un procesor pe 8 biţi (unitatea logico-aritmetică acceptă operanzi reprezentaţi pe 8 biţi). Magistrala de date este de 8 biţi iar magistrala de adrese de 16 biţi (spaţiul de memorie este de $2^{16} = 64KB$). Setul de instrucţiuni conţine 158 de instrucţiuni.

Microprocesorul Z80 are următoarele caracteristici:

- generează semnalul de refresh pentru memoria DRAM;
- are un pin pentru primirea unei întreruperi nemascabile;
- implementează un mecanism de tratare a întreruperilor vectorizate;
- conţine un set dublu de registre;
- setul de instrucţiuni conţine instrucţiuni pentru adresarea indexată a memoriei;
- setul de instrucţiuni conţine instrucţiuni pentru prelucrarea unor blocuri de date din locaţii adiacente de memorie.

Din familia Z80 fac parte următoarele circuite specializate:

- controler de port paralel Z80-PIO;
- controler de port serial Z80-SIO;
- controler DMA;
- circuit timer Z80-CTC.

Z80 poate funcţiona şi cu porturi de la alte familii de microprocesoare:

- I8255 - interfaţă paralelă cu trei porturi;
- I8251 - interfaţă serială cu două porturi.

Structura internă a procesorului Z80 este prezentată în figura 1.1.

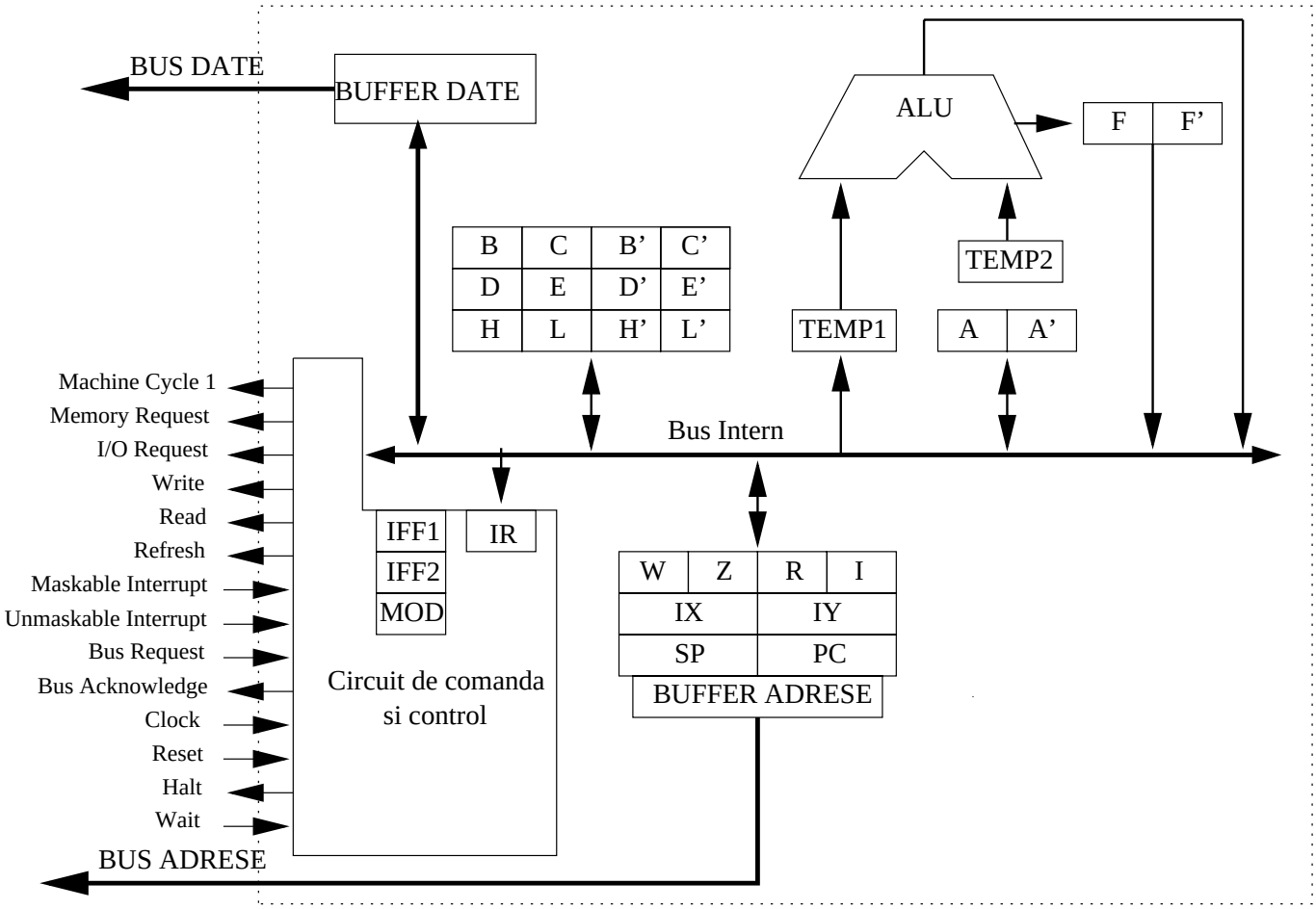

Figura 1.1: Structura internă a microprocesorului Z80.

1.1.1 Registrele procesorului

Registre specializate

Registrele din această categorie au roluri bine determinate în cadrul procesorului, avînd şi unele sarcini implicite:

- PC - *Program Counter* (16 biţi) utilizat pentru memorarea adresei instrucţiunii care urmează a fi executată;
- SP - *Stack Pointer* (16 biţi) utilizat pentru memorarea adresei vîrfului stivei. Stiva creşte spre adrese mici;

- IR - *Instruction Register* (8 biţi) registru invizibil pentru programator, folosit pentru memorarea codului instrucţiunii curente;
- A - *Accumulator* (8 biţi) utilizat ca registru implicit în multe operaţii aritmetice şi logice, A' - registru secundar;
- F - *Flag Register* (8 biţi) utilizat pentru memorarea indicatorilor de stare. Structura cuvîntului de stare este prezentată în tabelul 1.1:

7	6	5	4	3	2	1	0
S	Z	X	H	X	P	N	C

Tabelul 1.1: Structura registrului de stare şi indicatori.

  Semnificaţia biţilor din cuvîntul de stare este următoarea:

  - S - *Sign* este setat dacă rezultatul unei operaţii aritmetice este negativ;
  - Z - *Zero* este setat dacă rezultatul unei operaţii aritmetice este zero;
  - H - *Half Carry* este setat dacă există semitransport, de la bitul 3 la bitul 4 (se foloseşte la corecţia zecimală DAA);
  - P/V - *Parity/Overflow* este setat dacă numărul de biţi de valoare 1 din cuvînt este par (pentru paritate), sau dacă există o depăşire de domeniu (pentru operaţii aritmetice);
  - N - *Name of the last operation* este setat dacă ultima operaţie a fost adunare (indicator folosit pentru instrucţiunea DAA);
  - C - *Carry* este setat dacă există transport de la bitul cel mai semnificativ (MSB);
  - X - bit a cărui valoare nu contează.

- R - *Refresh Register* (7 biţi) utilizat pentru memorarea adresei de refresh;
- I - *Interrupt Register* (8 biţi) utilizat pentru identificarea sursei care a cauzat întreruperea (utilizat în modul 2 de întreruperi, IM2).

Registre de uz general

Registrele din această categorie au rolul de a păstra datele în imediata vecinătate a *ALU* pentru a putea fi accesate şi prelucrate rapid:

- B, C - registre generale de cîte 8 biţi care se pot accesa prin intermediul unor instrucţiuni ca un registru dublu de 16 biţi numit BC;
- D, E - registre generale de cîte 8 biţi care se pot accesa prin intermediul unor instrucţiuni ca un registru dublu de 16 biţi numit DE;
- H, L - registre generale de cîte 8 biţi care se pot accesa prin intermediul unor instrucţiuni ca un registru dublu de 16 biţi numit HL. Aceste registre se folosesc, în anumite instrucţiuni, pentru adresare indirectă;
- IX, IY - registre index de cîte 16 biţi utilizate la adresare indexată;

- B', C', D', E', H', L' - registre secundare al căror conţinut se poate interschimba cu conţinutul registrelor B, C, D, E, H, L prin executarea instrucţiunii EXX.

Registre de manevră

Registrele din această categorie sînt invizibile pentru programator, fiind folosite de către procesor ca locaţii de memorie temporare:

- W, Z - registre de cîte 8 biţi care se pot accesa şi ca un registru dublu de 16 biţi numit WZ, utilizat de către procesor pentru memorarea adreselor salturilor necondiţionate;
- T - *Temporar Register* utilizat de către procesor pentru diverse instrucţiuni;
- DB - *Data Buffer* registru tampon bidirecţional de 8 biţi utilizat la interfaţa dintre procesor şi lumea exterioară;
- AB - *Address Buffer* registru de 16 biţi care izolează magistrala de adrese internă de cea externă.

1.1.2 Circuite de comandă şi control

Circuitele de comandă şi control au funcţia de a genera în exterior semnalele de comandă de la procesor şi de a gestiona modul de lucru cu întreruperile. Alături de registrul IR în care se memorează codul instrucţiunii care se execută, circuitele de comandă mai conţin:

- IFF1, IFF2 - bistabile de validare/inhibare a întreruperilor;
- MOD - registru de 2 biţi utilizat pentru memorarea modului de întrerupere activ la un moment dat: IM0, IM1, IM2.

Semnalele de intrare şi ieşire ale circuitelor de comandă şi control sînt prezentate în figura 1.1.

1.1.3 Unitatea aritmetică şi logică

Unitatea aritmetică şi logică este de 8 biţi. Operaţiile aritmetice executate sînt: adunare, scădere, incrementare şi decrementare. Operaţiile logice executate sînt: OR, AND, XOR, NOT, comparare (la nivel de bit). Procesorul poate executa şi operaţii cu un singur operand, deplasări şi rotiri. Odată cu generarea rezultatului operaţiei aritmetice/logice se setează şi indicatorii din registrul de indicatori F.

1.1.4 Ciclurile procesorului

Definirea celor trei ciclii ce caracterizează funcţionarea procesorului este prezentată în figura 1.2. Aceşti ciclii sînt:

Ciclul de ceas (*Clock Cycle - CC*) are durata perioadei semnalului de ceas.

Ciclul maşină (*Machine Cycle - MC*) este format din unul sau mai mulţi ciclii de ceas. Pe durata unui ciclu maşină se desfăşoară o activitate intermediară, bine definită, cu o finalitate clară.

Ciclul instrucţiune (*Instruction Cycle - IC*) este format din unul sau mai mulţi ciclii maşină. Pe durata unui ciclu instrucţiune se desfăşoară toate acţiunile legate de execuţia unei instrucţiuni.

Ciclurile maşină ale procesorului Z80 sînt:

- citirea codului operaţiei (*Fetch - M1*);
- citirea unui operand din memorie (*Read*);
- scrierea unui operand în memorie (*Write*);
- citirea unui operand dintr-un port (*In*);
- scrierea unui operand într-un port (*Out*);
- ciclul intern;
- acceptarea cererii de întrerupere;
- acceptarea cererii de magistrale.

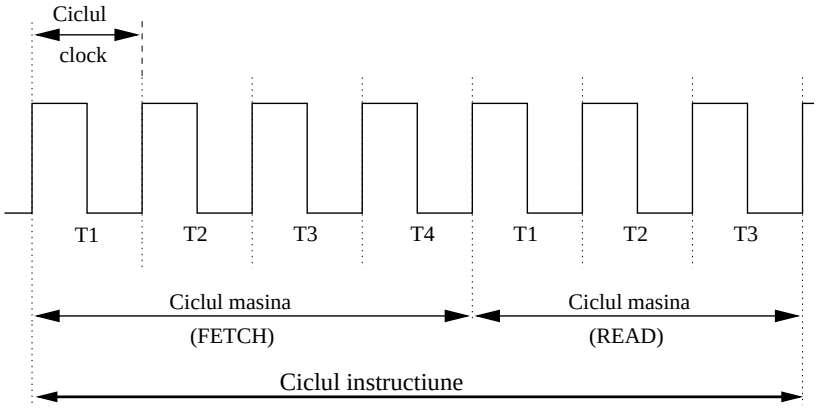

Figura 1.2: Ciclul instrucţiune pentru o operaţie de citire din memorie.

1.2 Macheta de laborator MPF1-B Microprofessor

1.2.1 Descriere hardware

Macheta didactică *MPF1-B Microprofessor* conţine un sistem cu microprocesor Z80. Frecvenţa semnalului de ceas este de 1.79 MHz. Sistemul dispune de două tipuri de memorie: o memorie ROM de capacitate 6 KB, în care sînt înscrise programele producătorului, şi o memorie RAM de capacitate 2 KB. Sistemul mai are posibilitatea de extindere a memoriei, avînd rezervat pentru aceasta un spaţiu de 8 KB. Zona de memorie ROM se întinde între adresele 0000H - 17FFH. Sistemul dispune de un program monitor care este înscris în EPROM-ul U6 şi ocupă spaţiul 0000H - 0FFFH. De la adresa 0800H este stocat un program de transfer între un calculator compatibil IBM PC şi macheta de laborator MPF1-B. Memoria RAM este de tip static (deci nu are nevoie de reîmprospătarea informaţiei) şi ocupă zona 1800H - 1FFFH. Conform

specificaţiilor tehnice ale MPF1-B, 80 de octeţi din memoria RAM (1FAFH - 1FFFH) sînt folosiţi de programul monitor. Se recomandă ca în acest spaţiu să nu se efectueze înscrieri. Spaţiul rezervat pentru extindere (2000H - 3FFFH) poate fi ocupat cu un alt EPROM sau cu o memorie RAM de tip static.

Harta memoriei poate fi configurată cu ajutorul unor jumperi (JP3, JP4, JP5) aflaţi pe machetă, conform tabelului 1.2.

U6	U7	JP3	JP4	JP5
4K (0000H - 0FFFH)	4K (2000H - 2FFFH)	2 - 3	2 - 1	2 - 3
4K (0000H - 0FFFH)	8K (2000H - 3FFFH)	2 - 3	2 - 3	2 - 1
6K (0000H - 1FFFH)	8K (2000H - 3FFFH)	2 - 1	2 - 3	2 - 1

Tabelul 1.2: Configurarea memoriei.

În figura 1.3 este prezentată harta memoriei sistemului MPF1-B.

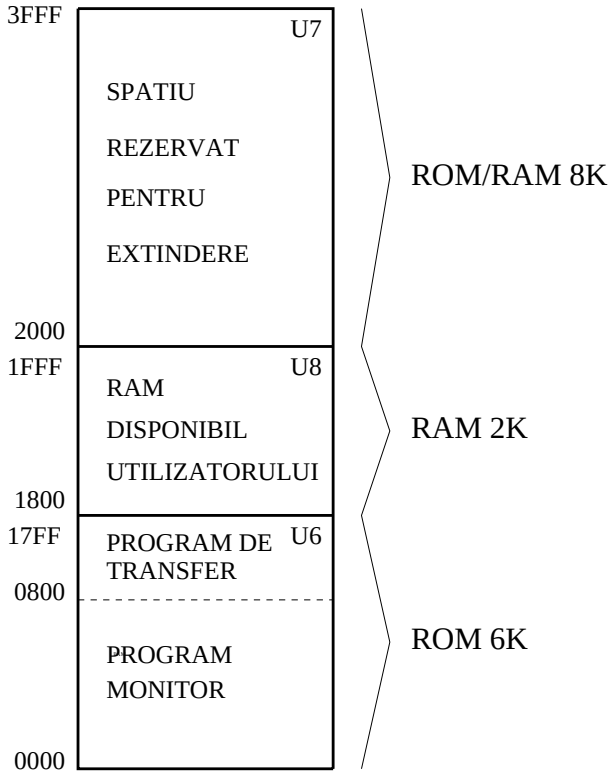

Figura 1.3: Harta memoriei sistemului MPF1-B.

Sistemul dispune de trei circuite specializate care folosesc porturi de intrare/ieşire. Interfaţa paralelă PIO dispune de două porturi paralele (16 biţi) şi ocupă următoarele adrese:

- 80H - portul A;
- 81H - portul B;
- 82H - controlul portului A;

- 80H - controlul portului B.

Circuitul counter/timer CTC ocupă următoarele adrese:

- 40H - canalul 0;
- 41H - canalul 1;
- 42H - canalul 2;
- 43H - registrul de control.

Al treilea circuit de care dispune sistemul este interfaţa paralelă programabilă 8255 din familia Intel. Aceasta are un total de 24 linii paralele care sînt folosite la scanarea tastaturii şi la controlul afişajului. Adresele ocupate de acest circuit sînt:

- 00H - portul A;
- 01H - portul B;
- 02H - portul C;
- 03H - registrul de control.

Portul A are biţii conectaţi astfel:

- bitul 7 - la intrarea de citire de la un dispozitiv de stocare magnetică;
- bitul 6 - la tasta *User Key* activă în stare *low*;
- biţii 5-0 - la şase rînduri ale tastaturii.

Portul B controlează cele 7 segmente şi punctele afişajului. Toţi biţii de ieşire sînt activi în stare *high*.

Portul C are biţii conectaţi astfel:

- bitul 7 - la ieşirea de scriere a unui dispozitiv de stocare pe bandă magnetică;
- bitul 6 - la tasta [MONI];
- biţii 5-0 - la şase coloane ale tastaturii şi ale afişajului. Bitul 0 corespunde primei cifre de la dreapta, iar bitul 5 primei cifre de la stînga. Toţi biţii sînt activi în stare *high*.

Portul C mai este conectat şi la difuzorul de pe machetă şi la led-ul *TONE-OUT*. Led-ul este aprins cînd ieşirea este în 0 logic. Afişajul este format din 6 celule cu 7 segmente. Tastatura conţine 36 taste din care 19 corespund anumitor funcţii, 16 reprezintă cifrele sistemului de numărare hexazecimal şi o tastă pe care poate fi definită de către utilizator. Ceasul sistemului este generat cu ajutorul unui oscilator cu cristal de cuarţ cu frecvenţa de $3,58MHz$. Acest semnal este divizat şi apoi aplicat la intrarea de ceas a microprocesorului Z80.

Iniţializarea sistemului se poate face în două moduri. Primul mod este punerea sub tensiune, caz în care au loc următoarele acţiuni:

1. Bistabilul de activare a întreruperilor este resetat ($IFF = 0$);
2. Registrul de întreruperi este resetat ($I = 0$);
3. Modul de întreruperi este setat cu valoarea zero ($MOD = 0$);

4. Registrul PC este încărcat cu valoarea 1800H;
5. Registrul SP este încărcat cu valoarea 1F9FH;
6. Se elimină punctele de oprire setate de utilizator;
7. Adresa rutinei de tratare a întreruperilor este setată la valoarea 0066H (această adresă se memorează la adresele 1FFEH şi 1FFFH);
8. Semnătura machetei, `uPF-1`, va defila pe afişaj.

Al doilea mod de iniţializare este prin activarea tastei de reset [RS]. În acest caz, au loc acţiunile de la punctele 1-5. Rutina de tratare a întreruperilor şi punctele de oprire rămîn neafectate, iar semnătura machetei nu mai defilează pe afişaj.

1.2.2 Descriere software

Sistemul dispune de un program monitor care permite introducerea programelor de la tastatură, verificarea şi rularea lor pas cu pas.

Tastele şi semnificaţia acestora este prezentată în tabelul 1.3.

Operaţii de bază

Iniţializarea sistemului se face prin apăsarea tastei de reset [RS]. Examinarea şi modificarea datelor din memorie se realizează folosind tastele adrese [ADDR] şi date [DATA]. Pentru verificarea unei date din memorie se tastează [ADDR] şi adresa la care se găseşte data respectivă. La apăsarea tastei [ADDR] vor apărea 4 puncte în dreptul primelor 4 cifre din stînga afişajului (cifrele care reprezintă adresa locaţiei de memorie). Cele 4 puncte semnifică faptul că este activ cîmpul de introducere a adresei. Cifrele care vor fi introduse, vor reprezenta adresa locaţiei de memorie ce va fi accesată. Primele două cifre din dreapta vor reprezenta data stocată la adresa afişată în stînga. Pentru vizualizarea datei de la adresa următoare se va tasta [+], iar pentru vizualizarea datei de la adresa anterioară se va tasta [-]. Pentru modificarea datei de la adresa afişată trebuie ca punctele să se afle în cîmpul de date. Acest lucru se obţine apăsînd tasta [DATA]. În momentul în care punctele se află în cîmpul de date, se pot modifica datele. Utilizatorul poate modifica numai datele dintre adresele 1800H şi 1FFFH (zona memoriei RAM). De asemenea, utilizatorul trebuie să evite modificarea datelor stocate în zona 1FA0H şi 1FFFH, deoarece această zonă este folosită de programul monitor.

Examinarea şi modificarea datelor stocate în registre se face cu tastele [REG] şi [DATA]. Pentru verificarea conţinutului unui registru se apasă tasta [REG] urmată de tasta pe care este înscris numele registrului (tastele de introducere a datelor). În partea dreptă se afişează numele registrului iar în partea stîngă datele memorate. Registrele de 8 biţi sînt grupate cîte două astfel: AF, BC, DE, HL şi AF', BC', DE', HL', reprezentate cu puncte lîngă nume. Registrul I şi bistabilul de validare a întreruperilor sînt grupate împreună. Pentru modificarea datelor, se apasă tasta [DATA], după care apar două puncte în zona corespunzătoare celui de-al doilea registru (poziţiile 3-4). După ce se modifică valoarea memorată, se apasă tasta [+] şi se trece la poziţiile 1-2, corespunzătoare primului registru.

Registrul de indicatori are o reprezentare particulară. Sistemul MPF1-B decodifică acest registru şi îl afişează în grupuri de 4 biţi astfel:

Tastă	*Semnificaţie*
[RS]	(ReSet) Semnal de iniţializare a sistemului
[ADDR]	(ADDRess) Introducerea unei valori în cîmpul de adresă
[REG]	(REGister) Selectarea unui registru al cărui conţinut este vizualizat
[DATA]	(DATA) Introducerea unei valori în cîmpul de date
[PC]	Tranferă controlul programului la valoarea curentă a registrului PC
[+]	Incrementare valoare afişată (dată sau adresă)
[-]	Decrementare valoare afişată (dată sau adresă)
[STEP]	(STEP) Rularea unei instrucţiuni din programul utilizatorului
[SBR]	(Set Break Point) Stabileşte un punct de oprire în programul utilizatorului
[CBR]	(Clear Brak Point) Anulează punctul de oprire din programul utilizatorului
[MONI]	(MONItor) Terminarea programului curent şi redarea controlului programului monitor
[GO]	(GO) Comandă de lansare în execuţie a programului aflat în memorie la adresa afişată pe display
[INS]	(INSert) Inserarea unui byte în memorie
[DEL]	(DELete) Ştergerea unui byte din memorie
[MOVE]	(MOVE) Copierea unui bloc de date dintr-un loc în altul
[RELA]	(RELAtive) Calcularea şi inserarea unei adrese relative pentru instrucţiuni de salt
[TAPE WR]	(TAPE WRite) Scriere pe bandă magnetică (ieşire serială)
[TAPE RD]	(TAPE ReaD) Citire de pe bandă magnetică (intrare serială)
[INTR]	(INTeRrupt) Semnal conectat la pinul de întreruperi mascabile al procesorului
[USER KEY]	Tastă a cărei semnificaţie poate fi modificată de către utilizator

Tabelul 1.3: Semnificaţia tastelor sistemului MPF1-B.

- grupul S, Z, H - [S Z x H];
- grupul P/V, N, C - [x P/V N C].

La apăsarea tastei [PC] în registrul PC se înscrie cea mai mică adresă din RAM (1800H).

Verificarea şi rularea programelor pas cu pas

Adresa de început a programului încărcat în memorie este stabilită prin apăsarea tastei [ADDR]. După apariţia adresei pe partea din stînga a afişajului, lansarea în execuţie a programului se face prin apăsarea tastei [GO]. Tasta [STEP] este similară tastei [GO], cu deosebirea că în acest caz procesorul va executa o singură instrucţiune, după care controlul este redat programului monitor. Programul monitor afişează noua valoare a registrului PC. Utilizatorul poate modifica şi verifica registrele sau conţinutul memoriei la fiecare pas. Tastele [GO] şi [STEP] sînt funcţionale numai cînd afişajul este în format standard: *Adresă-Dată*. Rularea unui program de dimensiuni mari pas cu pas consumă mult timp. Pentru evitarea acestui lucru se poate fixa un punct de oprire în program cu ajutorul tastei [SBR]. Astfel, programul se va executa

pînă la acest punct, după care se dă controlul programului monitor. În acest moment se pot vizualiza şi modifica date din memorie. Fixarea punctului de oprire se face apăsînd tasta [SBR], atunci cînd este afişată adresa la care se doreşte fixarea punctului de oprire. Într-un program se poate fixa un singur punct de oprire, care este simbolizat prin afişarea punctelor atît în zona de adrese, cît şi în zona de date a afişajului. Nu este permisă fixarea unui punct de oprire în zona memoriei ROM. Dacă o instrucţiune are mai mulţi octeţi, punctul de oprire trebuie fixat la primul octet al instrucţiunii, altfel vor apărea erori.

Eliminarea punctului de oprire se face apăsînd în orice moment tasta [CBR]. După apăsarea acestei taste, pe afişaj va apărea: [F.F.F.F.-F.F.]. În cazul în care un program se blochează sau intră în buclă infinită, se poate apăsa tasta [MONI] pentru ca sistemul să dea din nou controlul programului monitor. Pe display se va afişa conţinutul registrului PC. De asemenea, după execuţia unei instrucţiuni HALT, se poate apăsa tasta [MONI] pentru a reda controlul programului monitor, iar PC va lua valoarea adresei instrucţiunii următoare.

Funcţii pentru simplificarea utilizării machetei

Transferarea unui bloc de memorie se face cu tasta [MOVE]. Operaţia permite mutarea unui bloc de date dintr-o zonă în alta a memoriei. Succesiunea tastelor care trebuie apăsate pentru a muta blocul cuprins între adresele 1800H şi 18FFH la adresa 1810H este prezentată în tabelul 1.4.

Tastă apăsată	*Afişaj*	*Comentarii*
[MOVE]	`x.x.x.x.-s`	S semnifică adresa de început a blocului de date (Start)
[1][8][0][0]	`1.8.0.0.-s`	Adresa de început a blocului a fost fixată la adresa 1800H
[+]	`x.x.x.x.-e`	E semnifică adresa de sfîrşit a blocului de date (End)
[1][8][F][F]	`1.8.F.F.-e`	Adresa de sfîrşit a blocului a fost fixată la adresa 18FFH
[+]	`x.x.x.x.-d`	D semnifică adresa de destinaţie (Destination)
[1][8][1][0]	`1.8.1.0.-d`	Adresa de destinaţie a blocului a fost fixată la adresa 1810H
[GO]	`1810-x.x.`	Comanda executivă pentru mutarea blocului de date

Tabelul 1.4: Mutarea unui bloc de date în memorie.

Ştergerea unui byte de date se face cu ajutorul tastei [DEL]. Tasta este funcţională numai cînd afişajul este în formatul standard *Adresă-Dată*. Tasta se apasă în momentul în care este afişată data de la adresa de unde se doreşte ştergerea. Toate datele de deasupra sînt translatate în jos cu o poziţie (înspre adresele mici), iar la adresa 1DFFH se încarcă valoarea 00H. Zona de memorie în care se pot face ştergeri este 1800H şi 1DFFH.

Inserarea unui byte de date se face cu ajutorul tastei [INS]. Tasta este funcţională numai cînd afişajul este în format standard *Adresă-Dată*. Tasta se apasă în momentul în care este afişată adresa la care se doreşte inserarea datei respective. Succesiunea tastelor care trebuie apăsate pentru a insera un byte în memorie este prezentată în tabelul 1.5. Conţinutul locaţiilor de memorie înainte şi după operaţia de inserare este prezentat în tabelul 1.6.

Zona de memorie în care se pot face inserări este între 1800H şi 1DFFH. La inserare, baitul de la adresa 1DFFH se pierde.

Tastă apăsată	*Afişaj*	*Comentarii*
[ADDR][1802]	`1.8.0.2.-22`	Stabileşte adresa după care se doreşte inserarea unui byte
[INS]	`1803-0.0.`	La apăsarea tastei de inserare se înscrie 00 la adresa următoare
[3][3]	`1803-3.3.`	Se introduce valoare baitului inserat (în acest caz 33)

Tabelul 1.5: Inserarea unui byte în memorie.

Adrese	*Date înainte de inserare*	*Date după inserare*
1800	00	00
1801	11	11
1802	22	22
1803 inserare 33H	44	33
1804	55	44
1805	66	55

Tabelul 1.6: Conţinutul memoriei la inserarea unui byte.

Calculul adresei relative, necesare instrucţiunilor JR şi DJNZ, se poate face cu ajutorul tastei [RELA]. Se consideră cazul unei instrucţiuni JR aflată în memorie la adresa 1800H. Saltul trebuie făcut la adresa 1804H. Succesiunea tastelor care trebuie apăsate este prezentată în tabelul 1.7. Instrucţiunea JR de la adresa 1800H are doi baiţi. După execuţia acesteia, registrul PC are valoarea 1802H. Rezultă că pentru a face saltul la adresa 1804H, PC ar trebui însumat cu 02H.

Întreruperea unui program

Întreruperile nemascabile sînt folosite numai de programul monitor. Pinul 16 al procesorului (INT) este conectat la tasta [INTR]. La execuţia codului din programul monitor de la adresa 0038H controlul este transferat rutinei care începe la adresa stocată în locaţiile 1FFEH şi 1FFFH. În timpul acestui proces, starea procesorului este afectată. Conţinutul original de la adresele 1FFEH şi 1FFFH este 0066H. Instrucţiunea de la adresa 0038H este executată în următoarele situaţii:

1. Este acceptată o întrerupere de mod 1;
2. Se execută instrucţiunea RST 38H (cod FFH);
3. Magistrala de date este trecută în 1 logic. Dacă o întrerupere de mod 0 este acceptată fără vector de întrerupere, se execută instrucţiunea RST 38H;
4. Cînd un program încearcă să sară la o adresă inexistentă.

Dacă nu se modifică adresa memorată în locaţiile 1FFEH şi 1FFFH, efectul executării instrucţiunii de la adresa 0038H este acelaşi cu cel produs de apăsarea tastei [MONI], sau de atingerea

Tastă apăsată	*Afişaj*	*Comentarii*
[RELA]	`x.x.x.x.-s`	Se introduce adresa de start, adică adresa la care se află instrucţiunea JR
[1][8][0][0]	`1.8.0.0.-s`	Adresa instrucţiunii de salt este, în acest caz, 1800H
[+]	`x.x.x.x.-d`	Se introduce adresa de destinaţie, adică adresa la care se face saltul
[1][8][0][4]	`1.8.0.4.-d`	Adresa la care se face saltul este, în acest caz, 1804H
[GO]	`1801-0.2.`	Sistemul MPF1-B prelucrează valorile introduse şi introduce numărul relativ cu care se face saltul la adresa ce urmează instrucţiunii JR. În acest caz, de la 1801H la 1804H sînt 2 baiţi (JR ocupă 2 octeţi de la adresele 1800H şi 1801H)

Tabelul 1.7: Calculul unei adrese relative de salt.

unui punct de oprire în program. Utilizatorul poate să-şi definească propria rutină de servire modificînd conţinutul locaţiilor 1FFEH şi 1FFFH.

Instrucţiunea RST 30H are acelaşi efect cu cel al unui punct de oprire. Se numeşte *software break* deoarece nu implică partea hardware. De obicei, este folosită la sfîrsitul unei aplicaţii utilizator. De asemenea, se utilizează atunci cînd se doreşte stabilirea mai multor puncte de oprire într-un program.

La fiecare oprire a aplicaţiei utilizator, programul monitor verifică valoarea memorată în registrul SP. Dacă stiva programului utilizatorului se suprapune peste cea de sistem, se semnalizează o eroare prin apariţia pe afişaj a mesajului: [SYS-SP].

1.2.3 Program de transfer între PC şi MPF1-B

Editarea programului sursă

Fişierul sursă se poate edita cu orice editor de texte (NCEDIT, EDIT). Se recomandă folosirea editorului CWE.EXE deoarece acesta conţine meniuri specifice editării unui cod sursă Z80 şi oferă posibilitatea consultării interactive a unei documentaţii referitoare la setul de instrucţiuni şi directivele de asamblare Z80.

Codul sursă trebuie să înceapă obligatoriu cu directiva *ORG* care specifică adresa de memorie unde se va plasa programul transferat pe machetă. Zona de memorie RAM este 1800H - 1FFFH. Se recomandă încărcarea programului începînd cu adresa 1800H.

Prelucrarea programului sursă

Asamblarea fişierului sursă <*filename.s*> se face cu comanda:

asm800 <filename.s> -s asm816 -l-o <filename.o>

Rezultatul asamblării constă în obţinerea fişierelor obiect <*filename.o*> şi listing <*filename.l*>.

Link-editarea fişierului obiect <*filename.o*> se face cu comanda:

mlink <filename.o> -e 00000 -o <filename.bin>

Rezultatul link-editării constă în obţinerea fişierului binar *<filename.bin>*.

Conversia fişierului binar *<filename.bin>* în format Intel HEX se face cu comanda:

mload <filename.bin> -i-o <filename.hex>

Rezultatul conversiei constă în obţinerea fişierului în format Intel hex *<filename.hex>*. Acest fişier va fi încărcat în memoria RAM de pe machetă, la adresa specificată prin directiva ORG.

Prelucrarea automată a fişierului sursă, pînă la obţinerea formatului hex, poate fi făcută prin lansarea în execuţie a fişierului batch XASM.BAT cu comanda:

xasm < filename.s >

Ca rezultat, se obţin fişierele:

- *<filename.o>* - fişier obiect obţinut în urma asamblării;
- *<filename.l>* - fişier listing ce conţine adresele şi codul programului;
- *<filename.bin>* - fişier binar obţinut în urma link-editării;
- *<filename.hex>* - fişier în format Intel Hex care conţine imaginea programului în memorie.

Transferul programului de la PC la machetă

Transferul programului între PC şi macheta cu Z80 se face prin legătură serială între portul calculatorului PC COM1 şi un pin al portului paralel 8255 al machetei. Etapele transferării unui program între PC şi machetă sînt detaliate în continuare.

1. Se lansează în execuţie programul de recepţie de pe macheta cu Z80. Programul se află în memoria ROM la adresa 0800H. Aşteptarea transferului de la PC este semnalizată pe machetă prin apariţia pe afişaj a patru linii orizontale (_ _ _ _).
2. Se lansează în execuţie programul de transmisie de pe PC cu comanda:

   comm <filename.hex>

   Începerea transmisiunii este semnalizată prin apariţia pe afişajul machetei a patru linii verticale (| | | |).

   Dacă transferul s-a efectuat fără erori, pe afişajul machetei va apărea mesajul `Good`. Altfel v-a apărea un mesaj de eroare.

Execuţia programului utilizatorului

Se introduce adresa de început a programului utilizatorului prin apăsarea tastei [ADDR]. Lansarea în execuţie se realizeză prin apăsarea tastei [GO].

1.3 Experimente

I. Urmăriţi pe schema machetei blocul de selecţie a memoriei şi explicaţi conţinutul tabelului 1.2.

II. Încărcaţi următorul program în memoria RAM a machetei:

Adresă	Cod maşină	Instrucţiuni Z80
1800	3E00	LD A, 0
1802	3C	INC A
1803	47	LB B, A
1804	04	INC B
1805	48	LD C, B
1806	FB	EI

Justificaţi codul maşină plecînd de la instrucţiunile scrise în limbaj de asamblare.

III. Rulaţi programul. Ce valori vor avea registrele A, B, C, la terminarea programului?

IV. Executaţi programul pas cu pas. La fiecare pas, vizualizaţi conţinutul registrelor A, B, C.

V. Modificaţi programul astfel încît, după rulare, registrul C să conţină valoarea 6.

VI. Editaţi codul pe un PC, utilizînd editorul *CWE.EXE*. Asamblaţi programul şi transferaţi-l pe machetă. Rulaţi programul transferat.

Lucrarea 2

Vederea programatorului asupra procesorului Z80

Această lucrare prezintă setul de instrucţiuni al microprocesorului Z80 şi modul de utilizare a machetei de laborator. Se vor exersa programe simple, care pot fi asamblate manual. Introducerea programelor se va face atît manual cît şi prin transferul codului asamblat de la calculatorul PC unde este conectată macheta.

2.1 Moduri de adresare

Modul în care operanzii sînt aduşi pentru a fi procesaţi de către microprocesor, calea pe care i se comunică microprocesorului adresa la care se află stocat operandul căutat, se numeşte *mod de adresare*. Codul unei instrucţiuni trebuie să conţină şi informaţii asupra a ceea ce are de făcut instrucţiunea respectivă, nu numai adresa datelor implicate de aceasta. Adresarea registrelor, a celulelor de memorie şi a dispozitivelor de intrare/ieşire diferă esenţial. Microprocesorul Z80 are următoarele registre simpli (de 8 biţi): A, B, C, D, E, H, L. Adresele celor şapte registre pot fi codificate pe 3 biţi. Spaţiul de memorie adresabilă este $2^{16} = 64K$. Pentru a adresa o celulă de memorie, este necesară o adresă de 16 biţi. Pentru adresarea unui dispozitiv de intrare/ieşire este necesară o adresă de 8 biţi.

Microprocesorul Z80 are 6 moduri de adresare, prezentate în continuare.

- *Adresare imediată.* Acestă adresare este folosită ori de cîte ori corpul unei instrucţiuni înglobează şi data care reprezintă obiectul acelei instrucţiuni.

```
Limbaj de asamblare   Cod maşină
  LD A, 55H            3E55H       ;Încarcă registrul A
                                   ;cu valoarea 55H. Data (55H) este un
                                   ;octet de sine stătător în cîmpul de
                                   ;doi octeţi ai instrucţiunii.
  LD DE, 6AF5H         11F56AH     ;Încarcă registrul dublu DE cu
                                   ;valoarea hexazecimală 6AF5H.
                                   ;Data este un cuvînt de doi octeţi
                                   ;în cîmpul celor trei ai instrucţiunii.
  AND 8EH              E68EH       ;Se execută o operaţie logică între
```

```
                                    ;conţinutul registrului acumulator
                                    ;şi numărul 8EH.
```

- *Adresare implicită.* Această adresare este folosită dacă toate informaţiile necesare pentru localizarea datei care reprezintă obiectul unei instrucţiuni sînt încorporate în codul acesteia.

```
Limbaj de asamblare   Cod maşină
LD C, D                4AH          ;Transferă conţinutul registrului D în
                                    ;registrul C. Atît adresa iniţială a
                                    ;datei cît şi adresa ei de destinaţie
                                    ;sînt specificate în octetul de cod
ADD HL, BC             09H          ;Conţinutul registrului dublu HL este
                                    ;adunat cu cel al registrului dublu BC,
                                    ;ambii operanzi se specifică implicit,
                                    ;în octetul de cod.
LD DE, 6AF5H           11F56A       ;Operandul sursă este adresat imediat,
                                    ;iar operandul destinaţie este adresat
                                    ;implicit, ca fiind în registrul DE.
```

- *Adresare indirectă.* Această adresare este folosită dacă adresa unui operand este conţinută într-un registru sau o locaţie de memorie. Adresarea indirectă prin memorie se foloseşte într-un singur caz: atunci cînd Z80 acceptă o întrerupere în modul 2, adresa de început a subrutinei de tratare a întreruperii este determinată folosind această tehnică.

```
Limbaj de asamblare   Cod maşină
ADC (HL)               8EH          ;Adună la acumulatorul numărul stocat
                                    ;în memorie, la adresa conţinută în
                                    ;registrul dublu HL, iar apoi este
                                    ;adăugată şi valoarea bitului de
                                    ;transport.
PUSH DE                D5H          ;Conţinutul registrelor D şi E este
                                    ;salvat în memorie la adresa următoare
                                    ;din stivă, începînd cu registrul D.
                                    ;Adresa la care se face transferul este
                                    ;conţinută în indicatorul de stivă SP,
                                    ;al cărui conţinut este decrementat
                                    ;după fiecare transfer.
LD A, (BC)             0AH          ;Se încarcă conţinutul registrului A,
                                    ;de la adresa conţinută în registrul
                                    ;dublu BC.
```

- *Adresare directă.* Această adresare este folosită dacă în corpul unei instrucţiuni apare adresa efectivă a operandului ce constituie obiectul instrucţiunii. Adresarea directă se foloseşte la instrucţiunile de transfer între registre şi memorie, precum şi la instrucţiunile de salt.

```
Limbaj de asamblare   Cod maşină
LD (1784H), A          328417H      ;Transferă conţinutul acumulatorului
```

```
                                    ;în memorie, la adresa 1784H. Adresarea
                                    ;folosită este implicită pentru
                                    ;determinarea sursei (reg. A) şi directă
                                    ;pentru specificarea destinaţiei (1784H).
OUT (80H), A          D380H         ;Transferă conţinutul acumulatorului
                                    ;la dispozitivul de ieşire
                                    ;de la adresa 80H. Adresa sursei este
                                    ;specificată implicit iar cea a
                                    ;destinaţiei (portul cu adresa 80H)
                                    ;direct.
CALL 7778H            CD7877H       ;Apel de subrutină. Conţinutul registrului
                                    ;PC este salvat în stiva adresată de
                                    ;indicatorul de stivă SP, după care se
                                    ;execută un salt la adresa de început a
                                    ;subrutinei (7778H).
```

- *Adresare relativă.* Această adresare este folosită dacă adresa locaţiei de memorie în care se găseşte operandul se obţine adăugînd o valoare la conţinutul curent al registrului PC.

```
Limbaj de asamblare   Cod maşină
JR +05H                1803H        ;Execută un salt la adresa
                                    ;care se calculează însumînd valoarea
                                    ;curentă a registrului PC cu
                                    ;valoarea (deplasamentul) inclusă în
                                    ;cîmpul instrucţiunii.
```

- *Adresare indexată.* Această adresare este folosită dacă adresa unui operand se obţine adăugînd un deplasament (indice) la un registru de bază (index). Z80 are două registre de 16 biţi, IX şi IY, care pot fi folosite ca registre de bază. Tehnica de adresare indexată este eficientă în cazul în care datele sînt organizate într-un tabel. Considerînd că fiecare element din tabel ocupă un octet şi că adresa de început (adresa de bază) a tabelului se încarcă în registrul index IX sau IY, atunci regăsirea unei date dorite se poate face specificîndu-i doar indicele.

```
Limbaj de asamblare   Cod maşină
LD E, (IX+25H)         DD5E25H      ;Transferă un octet din memorie în
                                    ;registrul E. Adreasa celulei de memorie
                                    ;sursă se obţine însumînd conţinutul
                                    ;registrului de bază IX şi valoarea
                                    ;deplasamentului +25H din cîmpul
                                    ;instrucţiunii.
RES 7, (IY+69H)        FDCB69BEH    ;Şterge (înscrie 0) bitul
                                    ;cel mai semnificativ al octetului din
                                    ;memorie, octet a cărui adresă se obţine
                                    ;însumînd conţinutul registrului de bază
                                    ;(index) IY şi valoarea numerică (indice)
                                    ;69H conţinută în cîmpul instrucţiunii.
                                    ;Instrucţiunea ocupă 4 octeţi,
                                    ;din care trei octeţi sînt rezervaţi
                                    ;pentru cod.
```

2.2 Instrucţiuni de transfer

Majoritatea operaţiilor de tranfer de date se realizează cu ajutorul instrucţiunii LD. Operaţiile permit ca operanzii să fie de 8 sau 16 biţii. Procesorul poate să execute două instrucţiuni pentru interschimbarea datelor din registre: EX şi EXX. Pentru operaţii cu stiva se pot folosi instrucţiunile PUSH şi POP, iar pentru transferul blocurilor de date, instrucţiunile LDx şi LDxR.

Instrucţiunile de transfer pot avea unul, doi sau nici un operand. De asemenea, aceste instrucţiuni mai pot fi clasificate după sensul de transmitere a datelor şi tipul operanzilor:

```
  Tipul transferului              Exemple
registru <- registru          LD A, B     ; LD   HL, BC
registru <- memorie           LD A, (HL)  ; POP  AF
registru <- val. imediata     LD A, 25H   ; LD   HL, 125BH
memorie  <- registru          LD (HL), A  ; PUSH BC
memorie  <- memorie           LDD         ; LDIR
memorie  <- val. imediata     LD (HL), 5BH
```

Modul de stocare a unei date pe 16 biţi în memorie este "little endian" (octetul mai puţin semnificativ este stocat la adresa mai mică).

```
Adresa    Limbaj maşinã   Limbaj de asamblare   Comentarii
                           ORG 1800H             ;directivã
                                                 ;necesarã dacã programul
                                                 ;se transferã pe machetã
 1800      3E88           LD   A, 88H
 1802      011000         LD   BC, 10H          ;contorul operaţiei LDIR
 1805      112018         LD   DE, 1820H        ;adresa destinaţiei
 1808      216600         LD   HL, 0066H        ;adresa sursei
 180B      C5             PUSH BC               ;se salveazã BC în stivã
 180C      EDB0           LDIR
 180E      C1             POP  BC               ;se reface registrul BC
 180F      77             LD   (HL), A
 1810      FF             RST  38H              ;întoarcere la programul
                                                 ;monitor
```

Instrucţiunea RST 38H este necesară la sfîrşitul codului deoarece, după terminarea programului, controlul sistemului este preluat de programul monitor. În acest fel, se pot verifica datele obţinute.

Dacă se vizualizează un registru care conţine o dată pe 16 biţi (adresă) şi se apasă tasta [ADDR], sistemul va afişa conţinutul memoriei de la adresa memorată în tregistru. Reîntoarcerea în programul utilizatorului se va face apăsînd tasta [PC]. Instrucţiunea LDIR, fiind o instrucţiune repetitivă, va fi executată în mai mulţi paşi, astfel încît utilizatorul poate urmări ce se întîmplă la fiecare etapă a instrucţiunii.

2.3 Instrucţiuni aritmetice şi logice

Operaţiile aritmetice pot avea operanzi pe 8 sau 16 biţi. Operaţiile aritmetice cu date de 8 biţi au ca prim operand obligatoriu registrul acumulator (A), iar rezultatul este stocat tot în registrul A. Operaţiile cu date pe 16 biţi au ca surse unul din registrele HL, IX sau IY. Instrucţiunile logice sînt operaţii pe 8 biţi şi au ce registru sursă întotdeauna acumulatorul. Operaţiile aritmetice şi logice afectează indicatorii în funcţie de rezultatele acestora:

- *Carry* este setat în urma operaţiilor cu semn sau fără semn dacă rezultatul este un număr ce nu poate fi reprezentat pe 8 biţi. Indicatorul este setat şi atunci cînd se generează împrumuturi la operaţia de scădere. Acest indicator poate fi folosit drept condiţie la instrucţiunile de salt sau ca element de legătură la adunarea numerelor mari, ce se reprezintă pe 24 sau 32 de biţi.
- *Parity/Overflow* este setat la depăşirea domeniului de reprezentare a numerelor cu semn în complement faţă de 2 (între -128 şi +127) la operaţii aritmetice, sau în cazul în care rezultatul unei operaţii logice are un număr par de biţi 1.
- *Zero* este setat atunci cînd acumulatorul devine zero în urma unei operaţii aritmetice sau logice.
- *Sign* este setat atunci cînd cel mai semnificativ bit al acumulatorului este 1 (reprezentînd faptul că numărul din acumulator este interpretat drept un număr negativ).

```
Adresa   Limbaj maşină   Limbaj de asamblare   Comentarii
                          ORG  1800H
 1800     AF              XOR  A                ;A=0, Z=1
 1801     3D              DEC  A                ;A=0FFH, C=1
 1802     06FF            LD   B, 0FFH
 1804     90              SUB  A                ;A=0, Z=1
 1805     05              DEC  B                ;B=0FEH
 1806     88              ADC  B                ;A=FFH
 1807     0E0F            LD   C, 0FH
 1809     A1              AND  C                ;A=0FH, P/V=0
 180A     06F5            LD   B, 0F5H
 180C     80              ADD  B                ;A=4, C=1
 180D     FF              RST  38H
                          END
```

2.4 Instrucţiuni de salt şi de ciclare

Instrucţiunile de salt se pot clasifica, după adresa la care se face saltul, în salturi:

- *Absolute* - caz în care argumentul instrucţiunii este chiar adresa de memorie a destinaţiei.
- *Relative* - se calculează un deplasament (pozitiv sau negativ) faţă de adresa curentă. Aceste instrucţiuni se folosesc în cazul salturilor mici deoarece ocupă mai puţin spaţiu în memorie (2 octeţi, faţă de 3 cît ocupă un salt absolut).

Acest tip de instrucţiuni se mai pot clasifica şi după felul în care se face saltul:

- *Condiţionate* - se efectuează în funcţie de valorile unor indicatori, dacă aceştia respectă sau nu condiţia de salt.
- *Necondiţionate.*

Uneori, în programe este nevoie ca anumite părţi din cod să fie executate de mai multe ori. Pentru realizarea acestui lucru se stabileşte mai întîi un contor, în care se va încărca numărul de execuţii ale buclei, şi apoi se foloseşte o instrucţiune de salt condiţionat. În programele mai complexe, pot să apară chiar şi bucle imbricate.

În continuare, este prezentat un program care poate fi folosit la calcularea sumei dintre operanzii aflaţi într-un bloc de memorie (1900-190FH). Rezultatul va fi memorat în registrul DE.

```
Adresa   Limbaj     Eticheta   Limbaj de          Comentarii
         maşină                 asamblare
 1800     0E10                   LD  C,10         ;C-contor
 1802     AF                     XOR A            ;A=0
 1803     210019                 LD  HL, 1900H    ;adresa de început a
                                                  ;datelor; HL pointer
 1806     57                     LD  D, A         ;registrul D va memora
                                                  ;transporturile, D=0
 1807     86         XXX:        ADD A, (HL)
 1808     23                     INC HL           ;se obţine următoarea
                                                  ;dată
 1809     3001                   JR  NC, YYY      ;dacă nu s-a generat
                                                  ;transport, se efectuează
                                                  ;un salt relativ la
                                                  ;adresa YYY
 180B     14                     INC D            ;dacă s-a generat
                                                  ;transport, registrul D
                                                  ;se incrementează
 180C     0D         YYY:        DEC C            ;decrementare contor
 180D     20F8                   JR  NZ, XXX      ;repetă pînă se adună
                                                  ;toate datele
 180F     5F                     LD  E, A         ;se încarcă A în E,
                                                  ;rezultatul este în DE
 1810     FF                     RST 38H          ;întoarcere în programul
                                                  ;monitor
```

2.5 Stiva

În cadrul proiectării programelor, stiva este o secţiune a memoriei care un singur port pentru intrări şi ieşiri. Datele sînt înscrise şi citite în stivă cu ajutorul acestui port. Prima dată care se salvează în stivă se plasează la baza ei (*bottom of stack*). Data înscrisă cel mai recent în stivă este plasată în vîrful acesteia (*top of stack*). Din aceste motive, stiva este considerată o memorie LIFO (Last-In First-Out). Pentru a defini o stivă la începutul zonei de memorie

RAM, cea mai mare adresă este incrementată cu 1 şi după aceea este stocată în registrul stivă (SP) al microprocesorului. Următorul program ilustrează operaţiile cu stiva:

```
Limbaj de asamblare       Comentarii
 LD  SP, 1FAFH             ;secţiunea RAM cu adrese mai mici sau egale cu
                           ;1FAFH este consideratã stivă
 DEC SP                    ;SP este decrementat, deci adresa de bază a
                           ;stivei este 1FAEH
 LD  (SP), H               ;încarcă conţinutul registrului H în memorie
                           ;(RAM), la adresa 1FAEH
 DEC SP                    ;decrementare SP
 LD  (SP), L               ;plaseaza conţinutul registrului L în vîrful
                           ;stivei
 DEC SP
 LD  (SP), A               ;registrul A este stocat în vîrful stivei
 DEC SP
 LD  (SP), F               ;plasează conţinutul registrului F în vîrful
                           ;stivei
 ...
 LD  C, (SP)               ;extrage un octet din vîrful stivei, acesta este
                           ;stocat în registrul C
 INC SP                    ;SP incrementat cu 1. SP este deplasat spre
                           ;vîrful stivei
 LD  B, (SP)               ;extrage un octet din vîrful stivei
 INC SP
 LD  E, (SP)               ;încarcă în E octetul din vîrful stivei
 INC SP
 LD  D,(SP)                ;încarcă în D octetul din vîrful stivei.
                           ;Acesta dată a fost prima salvată în stivă
 INC SP                    ;SP are valoarea iniţială
```

În cadrul operaţiilor cu stiva, din programului anterior prezentat, datele pot fi stocate în memoria RAM folosind SP ca şi pointer. SP este decrementat cu 1 ori de cîte ori un octet este salvat în stivă, deci dimensiunea stivei creşte. Similar, SP este incrementat cu 1 ori de cîte ori un octet este citit din stivă, deci dimensiunea stivei scade. Stiva poate fi folosită pentru a stoca temporar adrese sau date pe 16 biţi. Microprocesorul Z80 dispune de instrucţiuni care permit salvarea/recuperarea regiştrilor dubli în/din stivă (*PUSH/POP*). În timpul acestor operaţii SP este decrementat/incrementat cu 2. Următorul program este echivalent cu cel prezentat anterior.

```
Limbaj de asamblare       Comentarii
 LD   SP, 1FAFH            ;identică cu prima instrucţiune
 PUSH HL                   ;identică cu instrucţiunile 2, 3, 4, 5
 PUSH AF                   ;identică cu instrucţiunile 6, 7, 8, 9
 POP  BC                   ;identică cu instrucţiunile 10, 11, 12, 13
 POP  DE                   ;identică cu instrucţiunile 14, 15, 16, 17
```

În cadrul unui program, este foarte important ca numărul instrucţiunilor de tip *PUSH* să fie egal cu cel al instrucţiunilor de tip *POP*.

2.6 Subrutine

Programele de tip aritmetic (adunări, scăderi, înmulţiri sau împărţiri), de control a tastaturii şi a afişajului sau altele, sînt des folosite ca părţi ale unor aplicaţii de dimensiuni mari. Pentru a economisi memorie şi a reduce posibilitatea de apariţie a erorilor, subrutinele sînt des folosite în cadrul programelor. Pentru manipularea subrutinelor se folosesc instrucţiunile *CALL* şi *RET*. Subrutinele pot fi executate necondiţionat, în funcţie de anumite condiţii sau indicatori.

În cazul apelului unei subrutine dintr-un program principal (*CALL*), se execută operaţiile prezentate în exemplul următor:

```
CALL   1A38H            ;apelul subrutinei de la adresa 1A38H
```

Apelul de procedură este echivalent cu:

```
PUSH   PC               ;salvează contorul program PC în stivă
JP     1A38H            ;salt la adresa 1A38H şi continuă execuţia
```

Spre deosebire de instrucţiunle de salt, după executarea unei subrutine, controlul programului este transferat instrucţiunii care urmează după apelul subrutinei. Instrucţiunea *RET* nu are nevoie de nici un operand, este codificată pe un octet şi are acelaşi efect ca şi instrucţiunea POP PC.

```
RET                     ;reîntoarcere la programul principal şi continuă
                        ;execuţia
```

Instrucţiunea de revenire din procedură este echivalentă cu:

```
POP PC                  ;se reface conţinutul PC, din stivă, după care
                        ;programul se execută conform valorii PC-ului
```

În figura 2.1 este prezentată forma generală a unui program care conţine apeluri de subrutine.

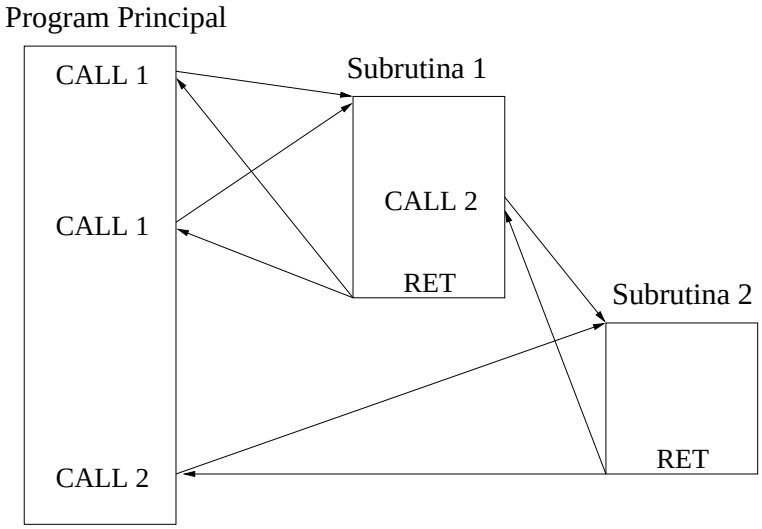

Figura 2.1: Apeluri de subrutine.

În cazul unui apel de subrutină din programul principal, trebuie considerate următoarele observaţii:

- Registrele care nu trebuie să fie afectate de către subrutină vor fi salvate în stivă de către programul apelant, înainte de apelul subrutinei.
- Modalitatea prin care rezultatele, obţinute în urma execuţiei subrutinei, se transferă în programul principal este stabilită de programator.

2.7 Experimente

I. Scrieţi un program care să încarce în registrele microprocesorului următoarele valori, folosind instrucţiuni de transfer pe 16 biţi: B=12, C=34, D=56, E=78, H=9, L=A. Asamblaţi-l şi apoi verificaţi-l pe macheta MPF1-B.

II. Transferaţi pe machetă codul executabil de la exemplul secţiunii 2.2. Executaţi programul. Verificaţi dacă s-au copiat cei 16 octeţi de la adresa 0066H la adresa 1820H. Rulaţi programul pas cu pas şi urmăriţi acţiunile fiecărei instrucţiuni. Verificaţi conţinutul registrelor afectate înainte şi după instrucţiuni.

III. Scrieţi un program în limbaj de asamblare care să şteargă conţinutul memoriei între adresele 1850H - 186FH. Rulaţi-l pe machetă şi verificaţi rezultatele.

IV. Scrieţi un program în limbaj de asamblare care setează conţinutul memoriei între adresele 1840H - 184FH cu următoarele valori: 0, 1, ..., F. Asamblaţi-l şi apoi verificaţi-l pe macheta MPF1-B.

V. Următorul program este folosit pentru a aduna un operand de 16 biţi aflat în memorie la adresele 1A00H - 1A01H cu conţinutul registrului dublu DE. Rezultatul va fi stocat în registrul dublu HL.

```
ORG 1800H
LD  A, (1A00H)
ADD A, E
LD  L, A
LD  A, (1A01H)
ADC A, D
LD  H, A
RST 38H
```

Rulaţi programul pe machetă şi urmăriţi rezultatele. Modificaţi programul de mai sus pentru o operaţie de scădere.

VI. Următorele linii de cod pot fi folosite pentru a împărţi 256 de octeţi din memorie în 16 blocuri. Adresa de început este 1900H.

```
       LD    HL, 19FFH
       LD    C,  0FH
LOOP2: LD    B,  10H
LOOP1: LD    (HL), C
       DEC   HL
       DJNZ  LOOP1
       DEC   C
```

```
JP    NZ, LOOP2
RST   38H
```

Setaţi valoarea fiecărui bloc de date în modul următor: 1H pentru blocul 1, 2H pentru blocul 2, ... , 0FH pentru blocul 16.

Lucrarea 3

Afişajul şi tastatura MPF1-B

3.1 Afişajul

Macheta *Microprofessor MPF1-B* conţine un display format din şase afişoare cu 7 segmente (plus punctul zecimal). Figura 3.1 prezintă asocierea dintre cele 7 segmente şi literele cu care sînt denumite acestea. Fiecare afişor conţine 8 LED-uri (7 asociate segmentelor şi unul asociat punctului zecimal). Cele 8 LED-uri sînt conectate cu anodul în comun.

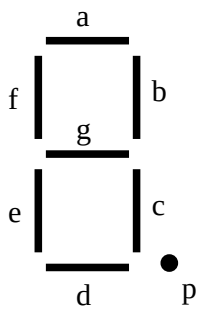

Figura 3.1: Denumirea celor 7 segmente ale unui afişor.

Comandarea concurentă a celor şase afişaje ar necesita un număr de 48 semnale, determinat astfel: (6 afişaje) × (8 semnale de date) = 48 semnale.

Comandarea secvenţială a celor şase afişaje ar necesita un număr de 14 semnale, determinat astfel: (6 semnale de selecţie a afişajului) + (8 semnale de date) = 14 semnale.

Ţinînd cont de inerţia ochiului uman, comandarea secvenţială a afişajelor cu o frecvenţă mare (cel puţin de 40 de ori pe secundă), poate crea impresia că afişajele sînt aprinse simultan. Liniile de date pentru segmente sînt notate cu Sa, Sb, Sc, Sd, Se, Sf, Sg şi Sp. Liniile de selecţie a afişajului comandat la un moment dat sînt notate cu D0, D1, D2, D3, D4, D5.

Pentru a comanda afişajul machetei sînt folosite porturile A şi B ale circuitului port paralel I8255, aşa cum se este prezentat în figura 3.3. Portul B (liniile PB0 - PB7) este folosit ca port de ieşire pentru liniile de date ale afişoarelor (Sa, Sb, Sc, Sd, Se, Sf, Sg). Portul C (liniile PC0 - PC5) este folosit ca port de ieşire pentru semnalele de comandă (D0, D1, D2, D3, D4, D5). Prin portul C se selectează afişorul care se luminează. Prin portul B se specifică segmentele luminate din cadrul afişajului selectat prin portul C. Toate segmentele sînt controlate de semnale active în $'1'$.

3.2 Tastatura

Deoarece reacţia unui calculator este mult mai rapidă decît cea a utilizatorului, tastatura trebuie scanată repetat pînă în momentul în care se detectează o tastă apăsată. O tastă oscilează pentru un timp scurt în momentul în care este apăsată sau eliberată. În figura 3.2 este prezentată diagrama răspunsului, în timp, în cazul operaţiilor de apăsare, respectiv de eliberare, a unei taste.

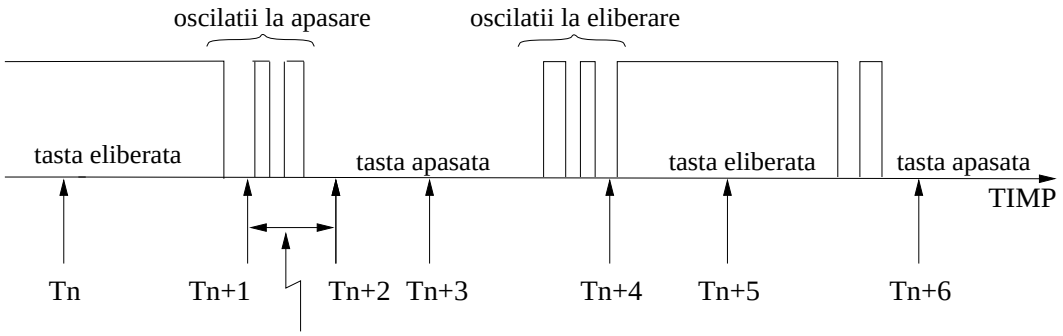

Figura 3.2: Răspuns în timp la scanarea tastaturii.

Datorită oscilaţiilor mecanice, dacă viteza de scanare a tastaturii este prea mare,apăsarea unei taste poate fi interpretată ca două sau mai multe apăsări succesive. Pentru a evita aceast lucru, perioada de scanare trebuie să fie mai mare decît perioada oscilaţiilor. Perioada oscilaţiilor nu este mai mare de 10 ms. Perioada de scanare trebuie aleasă între 10 ms şi 50 ms. În figura 3.2, săgeţile indică momentele de timp cînd este examinată tastatura. La momentul T_{n+2}, programul monitor detectează o tastă apăsată şi identifică codul acesteia. La momentul T_{n+3} tasta este, din nou, găsită apăsată. Deoarece tasta a fost detectată apăsată în timpul precedentei scanări, monitorul nu consideră această acţiune ca fiind o nouă apăsare (tasta nu a fost eliberată în acest interval). Doar dacă tasta este detectată ca fiind eliberată la momentul T_{n+4} sau T_{n+5}, atunci se consideră o nouă apăsare a acesteia. Un program care obţine date de la tastatură în acest mod, este lipsit de erori. Practic, nu contează intervalul de timp cît o tastă este apăsată.

Structura tastaturii, prezentată în figura 3.3, constă dintr-un număr de linii dispuse în formă matricială. În fiecare nod de intersecţie este poziţionată o tastă. Cele 6 linii orizontale şi cele 6 linii verticale formează, 36 de puncte de contact pentru tastatură. În momentul în care este acţionată o tastă, se va crea un contact electric între o linie şi o coloană a matricei. Cele şase linii orizontale (PA0 - PA5), sînt conectate la portul de intrare A al circuitului port paralel I8255. Dacă nici o tastă nu este apăsată, atunci cele 6 linii sînt conectate la tensiunea de alimentare (+5V) prin 6 rezistoare. Coloanele matricii sînt conectate la portul de ieşire C (PC0 - PC5), care la rîndul lui este conectat şi la afişaj.

Microprocesorul selectează cea mai din drepta coloană cu ajutorul liniei PC0. Tensiunile celor 6 linii ale matricei sînt evaluate secvenţial. La începutul procesului de scanare a tastaturii, un numărător este setat la zero, portul C va avea valoarea "11000001", deci PC5 - PC0 se vor găsi în starea logică "000001". În timpul cît se scanează tastatura, PC6 şi PC7 trebuie să fie în stare 1, deoarece PC6 este conectat la semnalul BREAK, iar PC7 la ieşirea de difuzor. Tensiunile liniilor tastaturii se citesc succesiv. Dacă o tastă este apăsată (pe linia respectivă se detectează o tensiune nulă), ea poate fi identificată cu ajutorul poziţiei liniei în cadrul portului. Dacă nici o tastă din prima coloană nu este apăsată, atunci microprocesorul va forţa pe portul C următoarea valoare a numărătorului (11000010), selectînd a doua coloană.

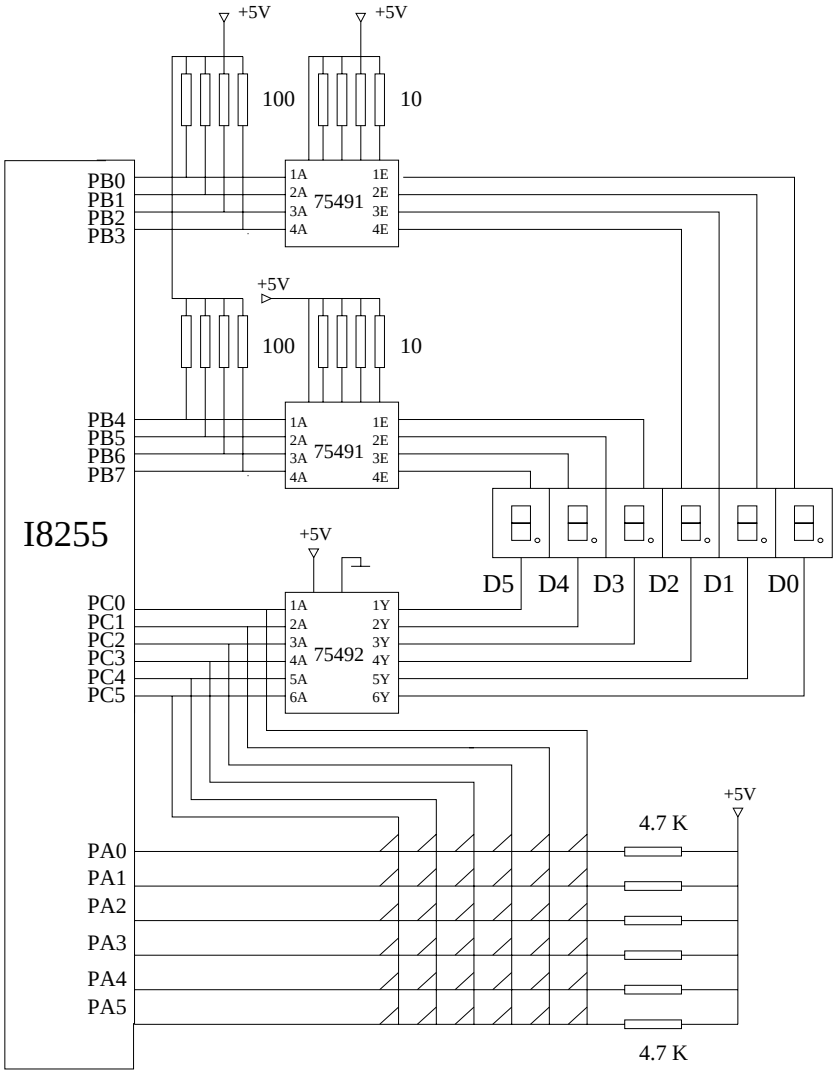

Figura 3.3: Structura display-ului şi a tastaturii machetei MPF1-B.

Procesul de scanare al tastaturii se desfăşoară succesiv din partea dreaptă spre stînga, şi de sus în jos. Fiecare tastă este codată: de cîte ori o tastă examinată este găsită neapăsată, se incrementează valoarea numărătorului. În momentul în care se identifică o tastă apăsată, codul poziţiei acelei taste este chiar valoarea numărătorului.

În tabelele 3.1 şi 3.2 sînt descrise codul poziţiei şi codul intern al fiecărei taste.

3.3 Subrutinele programului monitor

Programul monitor conţine 8 subrutine ce pot fi apelate şi din programele utilizatorului.

Adresele, descrierea şi parametrii subrutinelor SCAN1, SCAN, HEX7, HEX7SG, RAMCHK şi TONE sînt prezentate în tabelele 3.3, 3.4, 3.5, 3.6, 3.7, 3.8.

1E SBR	18 CBR	12 ′0′	0C ′1′	06 ′2′	00 ′3′
1F ′ − ′	19 PC	13 ′4′	0D ′5′	07 ′6′	01 ′7′
20 DATA	1A REG	14 ′8′	0E ′9′	0B ′A′	02 ′B′
21 ′ + ′	1B ADDR	15 ′C′	0F ′E′	09 ′E′	03 ′F′
22 INS	1C DEL	16 GO	10 STEP	0A	04
23 MOVE	1D RELA	17 TPWR	11 TPRD	0B	05

Tabelul 3.1: Codul poziţiei tastelor.

15 SBR	1A CBR	00 ′0′	01 ′1′	02 ′2′	03 ′3′
11 ′ − ′	18 PC	04 ′4′	05 ′5′	06 ′6′	07 ′7′
14 DATA	1B REG	08 ′8′	09 ′9′	0A ′A′	0B ′B′
10 ′ + ′	19 ADDR	0C ′C′	0D ′E′	0E ′E′	0F ′F′
16 INS	17 DEL	12 GO	13 STEP	22	20
1C MOVE	1D RELA	1E TPWR	1F TPRD	23	21

Tabelul 3.2: Codul intern al tastelor.

3.4 Exemple

Exemplul 1:

Afişarea mesajului `HELP US` pînă cînd se apasă tasta *STEP*.

```
        ORG  1800H
        LD   IX, HELP
DISP:   CALL SCAN
        CP   13H                ;codul intern al tastei STEP
        JR   NZ, DISP
        HALT

        ORG  2000H
HELP:   DEFB 0AEH               ; "S"
        DEFB 0B5H               ; "U"
```

SCAN1	
Adresă	0624H
Funcţie	Scanează tastatura şi afişajul timp de 1 ciclu, de la dreapta la stînga. Timpul de execuţie este de 9.97ms.
Intrare	IX este un pointer la buffer-ul de afişare.
Ieşire	(1) Indicatorul carry este setat dacă nu s-a apăsat nici o tastă; (2) Dacă a fost apăsată o tastă, indicatorul carry este resetat şi codul poziţiei tastei este memorat în registrul A.
Registre afectate	AF, AF', BC, BC', DE'.
Observaţii	(1) Sînt necesari 6 octeţi pentru memorarea celor 6 pattern-uri; (2) IX este un pointer spre cuvîntul ce corespunde rîndului din dreapta. IX+5 indică cuvîntul ce corespunde rîndului din stînga.

Tabelul 3.3: Subrutina SCAN1.

SCAN	
Adresă	05FEH
Funcţie	Similară cu cea a rutinei SCAN1 cu 2 excepţii: (1) SCAN1 scanează un ciclu, pe cînd SCAN scanează pînă se apasă o tastă; (2) SCAN1 întoarce poziţia tastei apăsate, în timp ce SCAN întoarce codul tastei apăsate.
Intrare	IX este un pointer la buffer-ul de afişare.
Ieşire	Registrul A conţine codul intern al tastei apăsate.
Registre afectate	AF, AF', B, BC', DE', HL.

Tabelul 3.4: Subrutina SCAN.

```
        DEFB 01FH           ; "P"
        DEFB 085H           ; "L"
        DEFB 08FH           ; "E"
        DEFB 037H           ; "H"

SCAN    EQU  05FEH
        END
```

Exemplul 2:

Afişarea cu intermitenţă a mesajului `HELP US`, folosind rutina SCAN1. Fiecare pattern este afişat timp de 500 ms prin executarea rutinei SCAN de 50 de ori. Valoarea registrului B determină frecvenţa de afişare.

```
        ORG  1800H
        LD   HL, BLANK
        PUSH HL
```

HEX7	
Adresă	0689H
Funcţie	Converteşte o cifră în baza 16 în formatul de afişare cu 7 segmente.
Intrare	Cei mai puţin semnificativi 4 biţi ai registrului A conţin cifra, exprimată în baza 16.
Ieşire	Rezultatul este memorat în registrul A.
Registre afectate	AF.

Tabelul 3.5: Subrutina HEX7.

HEX7SG	
Adresă	0678H
Funcţie	Converteşte două cifre din baza 16 în formatul de afişare cu 7 segmente.
Intrare	Cei mai puţin semnificativi 4 biţi ai registrului A conţin prima cifră, iar cei mai semnificativi 4 biţi ai registrului A conţin a doua cifră.
Ieşire	Primul pattern de afişat este memorat la adresa din registrul HL, iar al doilea este memorat la adresa următoare (conţinutul registrului HL, plus 1).
Registre afectate	AF, HL.

Tabelul 3.6: Subrutina HEX7SG.

```
         LD   IX, HELP
LOOP:    EX   (SP), IX
         LD   B, 50
HALFSEC: CALL SCAN1
         DJNZ HALFSEC
         JR   LOOP

         ORG  1820H
HELP:    DEFB 0AEH              ; "S"
         DEFB 0B5H              ; "U"
         DEFB 01FH              ; "P"
         DEFB 085H              ; "L"
         DEFB 08FH              ; "E"
         DEFB 037H              ; "H"
BLANK:   DEFB 0
         DEFB 0
         DEFB 0
         DEFB 0
         DEFB 0
         DEFB 0
```

RAMCHK	
Adresă	05F6H
Funcţie	Verifică dacă o anumită adresă este în RAM.
Intrare	Adresa care trebuie memorată este stocată în HL.
Ieşire	Dacă adresa este în RAM, indicatorul de zero este setat, altfel este resetat.
Registre afectate	AF.

Tabelul 3.7: Subrutina RAMCHK.

TONE	
Adresă	05E4H
Funcţie	Generează un sunet.
Intrare	Registrul C controlează frecvenţa sunetului. Perioada este aproximativ egală cu $(44 + C \cdot 13) \cdot 2 \cdot 0.56\mu s$, iar frecvenţa este $200/(10 + 3 \cdot C)kHz$.
Registre afectate	AF.

Tabelul 3.8: Subrutina TONE.

```
SCAN1    EQU  0624H
         END
```

Exemplul 3:

Afişarea codului intern al tastei apăsate.

```
         ORG  1800H
         LD   IX, OUTBUF
LOOP:    CALL SCAN
         LD   HL, OUTBUF
         CALL HEX7SEG
         JR   LOOP

         ORG  1900H
OUTBUF:  DEFB 0
         DEFB 0
         DEFB 0
         DEFB 0
         DEFB 0
         DEFB 0

SCAN     EQU  05FEH
HEX7SG   EQU  0678H
         END
```

Pentru a afişa codul poziţiei tastei apăsate, programul trebuie modificat după cum urmează:

```
         ORG  1800H
         LD   IX, OUTBUF
LOOP:    CALL SCAN1
         JR   C, LOOP
         LD   HL, OUTBUF
         CALL HEX7SEG
         JR   LOOP
```

Exemplul 4:

Se convertesc trei octeţi din memorie în formatul şapte segmente. Rezultatul este stocat în memorie la adresele 1903H - 1908H, după care este afişat.

```
         ORG  1800H
         LD   DE, BYTE0
         LD   HL, OUTBUF
         LD   B, 3
LOOP:    LD   A, (DE)
         CALL HEX7SEG
         INC  DE
         DJNZ LOOP

         LD   IX, OUTBUF
         CALL SCAN
         HALT

         ORG  1900H
BYTE0:   DEFB 10H
         DEFB 32H
         DEFB 45H

OUTBUF:  DEFS 6

SCAN     EQU  05FEH
HEX7SEG  EQU  0678H
         END
```

Cei trei octeţi de date sînt stocaţi la adresele 1900H - 1902H.

3.5 Experimente

I. Transferaţi codul executabil de la *Exemplul 1* pe machetă, după care executaţi programul. Încărcaţi în memorie la adresa 1808H valoarea 1AH. Apăsînd tasta *CBR*, mesajul nu va mai apărea pe afişaj. De ce? Setaţi conţinutul memoriei între adresele 1820H - 1822H cu valorile 3FH, BDH, 85H. Ce se va afişa pe display? Scrieţi un program care să afişeze `SYS-SP` pînă cînd se apasă tasta *PC*.

II. Transferaţi codul executabil de la *Exemplul 2* pe machetă, după care executaţi programul. Setaţi conţinutul locaţiei de memorie 180BH cu valoarea 01. Ce se va afişa pe display? Dar pentru valoarea 05?

III. Pentru *Exemplul 3* setaţi conţinutul zonei de memorie 1900H - 1905H la valoarea FFH. Ce se va afişa?

IV. Modificaţi programul de la *Exemplul 4*, astfel încît să se afişeze secvenţa `333446`.

Lucrarea 4

Aplicaţii cu circuitul Z80-PIO

Această lucrare prezintă modul de lucru al circuitului port paralel Z80-PIO, comanda cu acest circuit a unor dispozitive aflate pe placa de aplicaţii, precum şi cîteva programe care să exemplifice această problematică.

4.1 Interfaţa paralelă programabilă Z80-PIO

4.1.1 Arhitectura internă a circuitului Z80-PIO

Z80-PIO (Parallel Input/Output - engl.) este o interfaţă paralelă programabilă prevăzută cu o unitate de comandă şi două porturi paralele de 8 biţi de date şi 2 semnale de conversaţie (Ready şi Strob) cu ajutorul cărora se controlează transferul de date. Cele două porturi furnizează o interfaţă compatibilă TTL între procesor şi dispozitivele periferice. Porturile, denumite A şi B, pot fi programate ca porturi de intrare sau porturi de ieşire, la nivel de octet sau de bit. Portul A poate fi programat şi pentru a lucra bidirecţional. În funcţie de indicatorii de stare ai echipamentelor periferice, se pot genera întreruperi programabile. Figura 4.1 prezintă arhitectura internă a circuitului Z80-PIO. Simbolul bloc asociat acestui circuit este prezentat în figura 4.2.

Pinii circuitului au următoarea semnificaţie:

Semnale generale:

CLK (System Clock) - Semnal de ceas comun tuturor circuitelor din sistemul cu microprocesor Z80.

/RESET (Reset) - Semnal de iniţializare.

Magistrala de date:

D0-D7 (System Data Bus) - Magistrală bidirecţionlă conectată la magistrala de date a procesorului.

Semnalele de control (primite de la procesor):

SEL.PB/NPA (Port B or A Select) [intrare, High=B, Low=A] - Semnal de selecţie a portului accesat în timpul unui transfer de date între procesor şi PIO. Pentru această selecţie se foloseşte bitul A0 al magistralei de adrese a procesorului.

SEL.CTRL/NDATA (Control or Data Select) [intrare, High=C, Low=D] - Semnal care de-

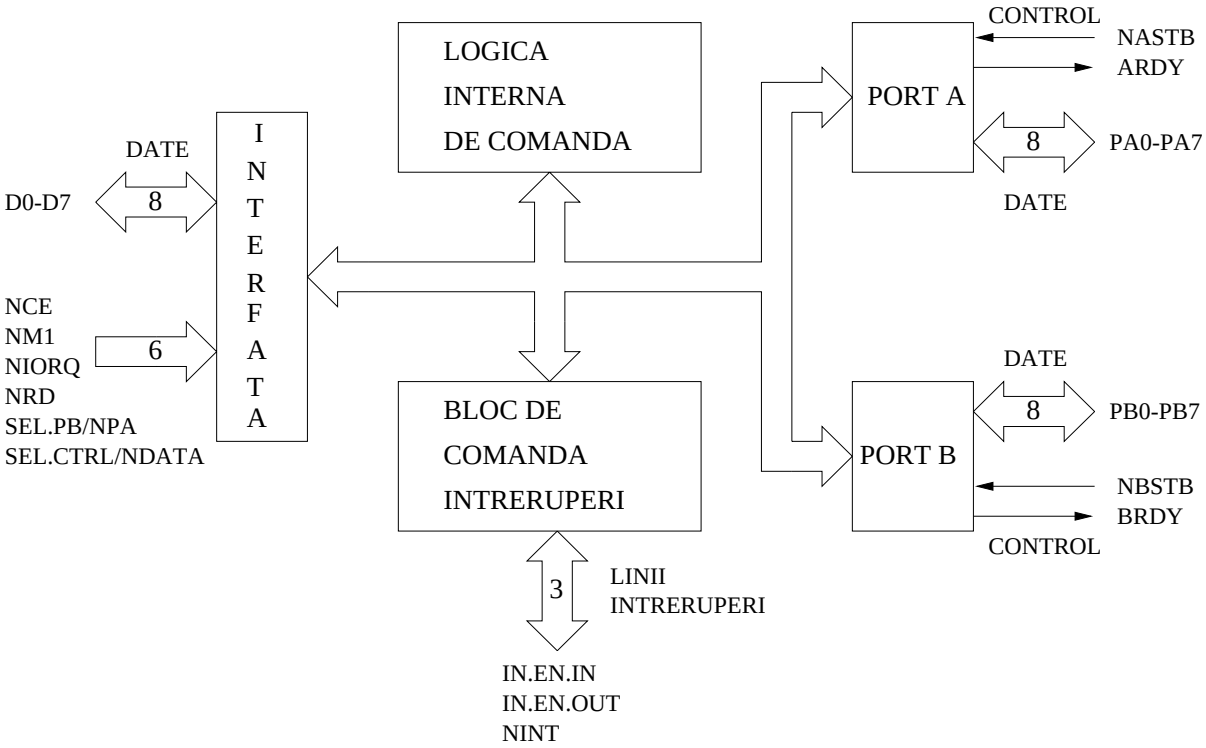

Figura 4.1: Arhitectura circuitului Z80-PIO.

fineşte tipul de date care se transferă între procesor şi PIO. Dacă semnalul este în starea 1, cuvîntul este interpretat ca o comandă, altfel ca o dată. Pentru această selecţie se foloseşte bitul A1 al magistralei de adrese a procesorului.

/CE (Chip Enable) [intrare, activă în stare Low] - Semnal de validare a circuitului PIO. Se obţine prin decodificarea magistralei de adrese.

/M1 (Machine Cycle 1) [intrare de la procesor, activă în stare Low] - Semnal de la procesor utilizat ca impuls de sincronizare pentru a controla mai multe operaţii interne ale circuitului PIO. Cînd semnalele */M1* şi */RD* sînt active simultan, procesorul încarcă o dată din memorie. Semnalul */M1* mai are încă două funcţii în cadrul circuitului PIO: sincronizează logica de întreruperi din PIO şi iniţializează circuitul PIO în momentul apariţiei semnalului */M1*, fără ca unul din semnalele */RD* sau */IORQ* să fie active.

/IORQ (Input/Output Request) [intrare de la procesor, activ în stare Low] - Semnal de la procesor utilizat împreună cu *SEL.PB/NPA*, *SEL.CTRL/NDATA*, */CE* şi */RD* pentru a transfera comenzi şi date între procesor şi circuitul PIO. Cînd */CE*, */RD* şi */IORQ* sînt active portul adresat de *SEL.PB/NPA* scrie date în procesor (operaţie de citire). Cînd */RD* nu este activ portul adresat de *SEL.PB/NPA* este înscris cu date sau informaţii de control de la procesor, în funcţie de starea semnalului *SEL.CTRL/NDATA*.

Dacă */IORQ* şi */M1* sînt simultan active procesorul anunţă acceptarea unei întreruperi. Portul care a cerut întreruperea pune în mod automat vectorul lui de întrerupere pe magistrala de date a procesorului, în cazul în care dispozitivul periferic care a cerut întreruperea are prioritatea cea mai mare.

/RD (Read Cycle Status) [intrare de la procesor, activ în stare Low] - Dacă semnalul */RD* este activ sau o operaţie de intrare/ieşire este în curs de desfăşurare,*/RD* este folosit împreună cu semnalele SEL.PB/NPA, SEL.CTRL/NDATA, /CE şi /IORQ pentru a se transfera date de la

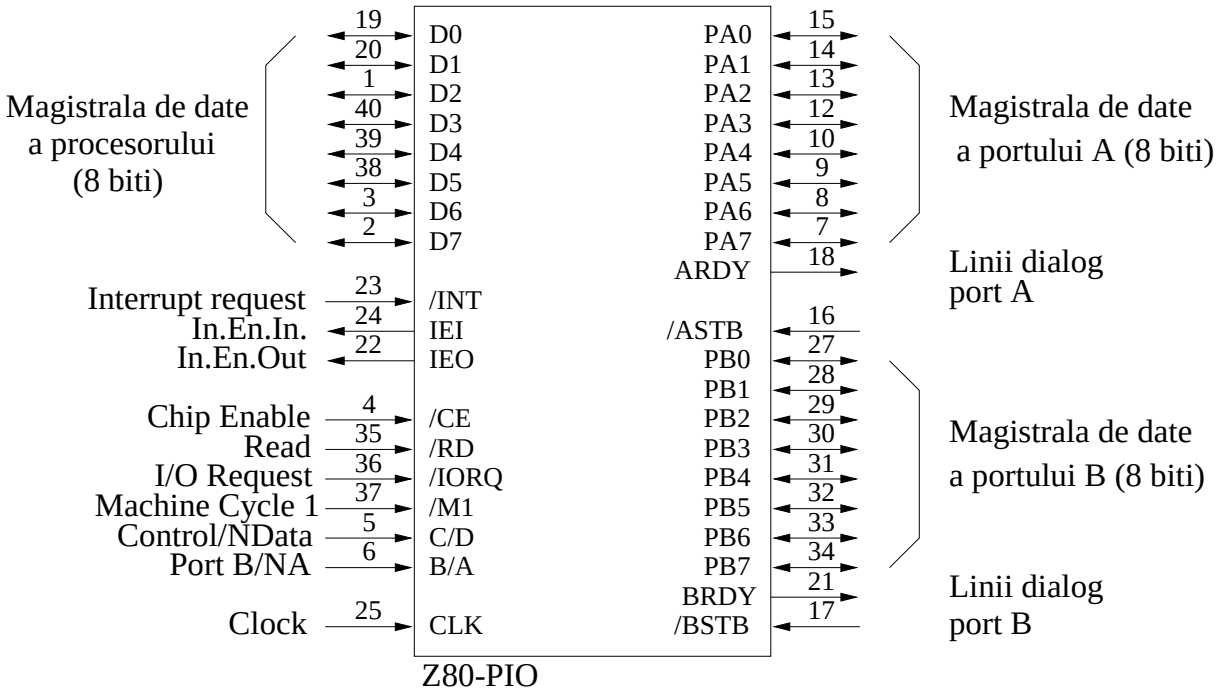

Figura 4.2: Simbolul bloc al circuitului Z80-PIO.

circuitul PIO spre procesor.

Semnalele de întrerupere:

IN.EN.IN (IEI) (Interrupt Enable In) [intrare, activ în stare High] - Semnal de validare a întreruperilor. Este folosit pentru a forma un lanţ de priorităţi la cererile de întrerupere, cînd se utilizează mai multe dispozitive periferice comandate prin întreruperi. Starea logică 1 a aceastei linii semnifică faptul că nici un alt dispozitiv cu prioritate mai mare nu este servit de procesor printr-o rutină de tratare a întreruperii.

IN.EN.OUT (IEO) (Interrupt Enable Out) [ieşire, activ în stare High] - Semnal de validare a întreruperilor. Este cel de-al doilea semnal necesar pentru a forma lanţul de priorităţi la întreruperi. Starea logică a acestui semnal este 1 numai dacă şi IEI este în 1 logic şi dacă procesorul nu deserveşte o întrerupere de la acest circuit PIO. Altfel acest semnal blochează cererile de întrerupere pentru dispozitivele mai puţin prioritare, în timp ce un dispozitiv cu prioritate mai mare este deservit de procesor printr-o rutină specifică.

/INT (Interrupt Request) [ieşire, activ în stare Low] - Cerere de întrerupere adresată procesorului.

Semnalele porturilor:

PA0-PA7 (Port A Bus) [intrări/ieşiri, active în stare High, 3-state] - Magistrală bidirecţională de date. Pe această magistrală se realizează transferul de date între portul A al PIO şi dispozitivul periferic.

/ASTB (Port A Strobe Pulse From Peripheral Device) [intrare, activ în stare Low] - Semnificaţia semnalului depinde de modul de funcţionare ales pentru portul A după cum urmează:

Mod de ieşire: frontul crescător al semnalului emis de către dispozitivul periferic semnalează că acesta a primit data furnizată de circuitul PIO.

Mod de intrare: pulsul emis de către dispozitivul periferic semnalează că acesta este gata să scrie date de la portul A. Datele sînt încărcate în PIO numai cînd acest semnal este activ.

Mod bidirecţional: cînd semnalul este activ datele din registrul de ieşire al portului A sînt transferate pe magistrala de date a portului A (bidirecţională). Frontul crescător al semnalului emis de către dispozitivul periferic semnifică faptul că acesta a primit data furnizată de circuitul PIO.

Mod bit: semnalul este dezactivat intern (întrucît nu este necesar).

ARDY (Register A Ready) [ieşire, activ în stare High] - Semnificaţia semnalului depinde de modul de funcţionare selectat după cum urmează:

Mod de ieşire: semnalul activ indică faptul că registrul de ieşire al portului A a fost încărcat şi datele sînt valide pentru citire.

Mod de intrare: semnalul este activ cînd registrul de intrare al portului A este gol şi poate să preia datele de la dispozitivul periferic.

Mod bidirecţional: semnalul este activ cînd datele pentru dispozitivul periferic sînt disponibile în registrul de ieşire al portului A. În acest mod datele nu sînt puse pe magistrala de date a portului A numai cînd semnalul /ASTB este activ.

Mod bit: semnalul este dezactivat intern (întrucît nu este necesar).

PB0-PB7 (Port B Bus) [intrări/ieşiri, active în stare High, 3-state] - Magistrală bidirecţională de date. Pe această magistrală se realizează transferul de date între portul B al PIO şi dispozitivul periferic. Portul B poate furniza pe fiecare linie 1.5 mA la 1.5V pentru a comanda tranzistoare montate în conexiune Darlington.

em /BSTB (Port A Strobe Pulse From Peripheral Device) (intrare, activ în stareLow) - Similar cu semnalul */ASTB*, cu excepţia faptului că în modul bidirecţional semnalul încarcă datele de la dispozitivul periferic în registrul de intrare al portului A.

BRDY (Register B Ready) [ieşire, activ în stare High] - Similar cu semnalul *ARDY*, cu excepţia faptului că în mod bidirecţional semnalul este în 1 logic cînd registrul de intrare al portului A este gol şi poate să preia datele de la dispozitivul periferic.

4.1.2 Modurile de lucru ale circuitului Z80-PIO

Modul 0, de ieşire Ambele porturi (A sau B) pot fi programate în acest mod. Un ciclu de ieşire este întotdeauna pornit de execuţia unei instrucţiuni de ieşire de către procesor. La semnalul */WR* furnizat de către procesor datele de pe magistrala de date sînt înscrise în registrul de ieşire al portului PIO selectat. Impulsul de scriere poziţionează indicatorul *READY* după frontul descrescător al semnalului de ceas, indicînd astfel disponibilitatea informaţiei pentru dispozitivul periferic. Linia *READY* rămîne activă pînă cînd PIO recepţionează de la dispozitivul periferic semnalul de *STROB*, care semnifică faptul că perifericul a preluat infomarţia. Frontul crescător al semnalului de STROB generează o întrerupere */INT*, dacă bistabilul de activare a întreruperilor a fost setat şi dacă dispozitivul periferic are prioritatea cea mai mare.

Modul 1, de intrare Ambele porturi (A sau B) pot fi programate în acest mod, fiecare avînd un registru de intrare adresabil de către procesor. Data de la echipamentul periferic este înscrisă în registrul de intrare al porturilor PIO pe frontul descrescător al semnalului de *STROB*. Frontul crescător al aceluiaşi semnal activează */INT*, dacă bistabilul de activare a întreruperilor a fost setat şi dacă echipamentul periferic respectiv are prioritatea cea mai mare. Următorul front descrescător (activ) al semnalului de ceas inactivează semnalul *READY*, prin care echipamentul periferic este anunţat că portul de intrare conţine o informaţie care nu a fost preluată de către procesor, şi deci nu mai poate fi încărcat cu altă informaţie pînă la citirea celei existente de către procesor. După frontul crescător al semnalului */RD* de la procesor, pe următorul front descrescător al semnalului de ceas, se reactivează semnalul *READY*.

Modul 2, bidirecţional Acest mod reprezintă o combinaţie a modurilor 0 şi 1. Portul A va fi port bidirecţional. Liniile de coversaţie ale portului A se folosesc drept linii de dialog pentru ieşire, iar liniile de conversaţie ale portului B se folosesc drept linii de dialog pentru intrare. Portul B va fi programat, în acest caz, în modul bit (care nu necesită linii de dialog). Dacă apare o întrerupere va fi folosit vectorul de întrerupere al portului A în cazul unei operaţii de ieşire sau cel al portului B în cazul unei operaţii de intrare. Datele de ieşire sînt disponibile perifericului numai cînd semnalul */ASTB* este în 0 logic.

Modul 3, bit Ambele porturi pot fi programate în acest mod. Nu se folosesc semnale de dialog. O operaţie normală de scriere poate avea loc în orice moment. Un semnal de întrerupere se generează dacă starea unei intrări sau starea tuturor intrărilor se modifică. Condiţiile de generare a întreruperii sînt definite în timpul programării circuitului. Nivelul activ poate fi ales 1 sau 0 logic, iar condiţia logică este fie pentru o intrare activă (SAU logic), fie pentru toate intrările active (SI logic). Dacă portul A este programat în mod bidirecţional, portul B nu mai are un semnal de întrerupere şi pentru acest motiv va trebui să fie interogat. Acest mod se foloseşte pentru aplicaţiile în care se generează semnale de comandă sau se monitorizează stări.

4.1.3 Structura unui port

Portul are un registru de intrare şi unul de ieşire, în felul acesta putînd funcţiona în orice mod. Conţinutul acestor registre se modifică numai atunci cînd sînt înscrise, în rest ele păstrînd datele scrise în ele. Portul mai conţine un registru de comandă a modului de lucru (astfel fiecare port se poate programa independent de celălalt - cu excepţia modului 2 de lucru), logică de comandă a registrului mască (utilizat în modul 3 de lucru) şi logică de comandă a liniilor de dialog, fiind astfel capabil să controleze un sistem de întreruperi ierarhizate. În figura 4.3 este prezentată structura unui port al circuitului PIO.

4.1.4 Blocul de comandă a întreruperilor

Blocul de comandă a întreruperilor se ocupă de întregul protocol de întreruperi spre procesor. Poziţia fizică a unui dispozitiv într-un lanţ de priorităţi determină prioritatea lui. Fiecare circuit Z80-PIO are două semnale (*IEI* şi *IEO*) pentru a forma un lanţ de priorităţi, aşa cum se prezintă în figura 5.4. Dispozitivul care este cel mai apropiat de procesor are prioritatea cea mai mare. În cadrul unui circuit PIO, întreruperile portului A au prioritate mai mare decît cele ale portului B. În modurile de intrare, ieşire şi bidirecţional o cerere de întrerupere se poate genera

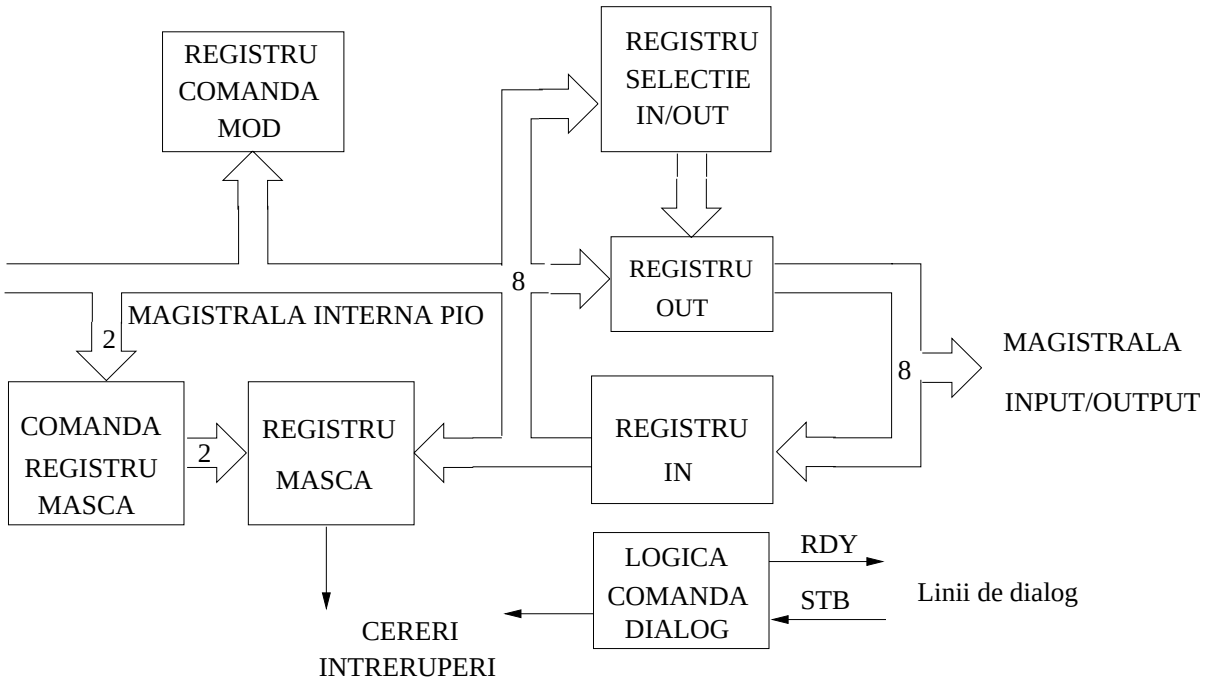

Figura 4.3: Structura unui port al circuitului Z80-PIO.

oricînd perifericul cere transferul unui nou octet. Circuitul PIO permite controlul complet al întreruperilor ierarhizate. Astfel, dispozitivele cu prioritate mai mică nu le pot întrerupe pe cele cu prioritate mai mare, ale căror subrutine de deservire nu au fost executate de procesor. Cele cu prioritate mai mare pot întrerupe deservirea celor mai puţin prioritare.

Dacă procesorul aflat în modul 2 de întrerupere acceptă o întrerupere, circuitul PIO care a cerut întreruperea trebuie să furnizeze unităţii centrale *un vector de întrerupere*. Acest vector indică o locaţie de memorie unde se află adresa rutinei de servire a întreruperii. Cei 8 biţi furnizaţi de dispozitivul care a cerut întreruperea reprezintă cei mai puţin semnificativi 8 biţi ai indicatorului, în timp ce registrul I din procesor asigură cei mai semnificativi 8 biţi.

Fiecare port are un vector de întrerupere independent. Cel mai puţin semnificativ bit al vectorului este fixat în mod automat în 0 în interiorul circuitului PIO, pentru că indicatorul trebuie să identifice două locaţii de memorie succesive, pentru a forma o adresă completă de 16 biţi. Spre deosebire de alte periferice din sistemul Z80, circuitul PIO nu acceptă întreruperi imediat după programare, ci aşteaptă până cînd */M1* este în 0 logic (de exemplu în timpul aducerii unui cod de operaţie).

Circuitul PIO decodifică instrucţiunea de revenire din întrerupere RETI direct de pe magistrala de date a sistemului, astfel încît fiecare circuit PIO din sistem ştie în orice moment dacă este deservit de procesor printr-o rutină de tratare a întreruperii, nefiind astfel necesară nici o comunicaţie în plus cu procesorul.

4.1.5 Iniţializarea circuitului Z80-PIO

Circuitul Z80-PIO intră în mod automat în starea iniţială (de reset) cînd este pus sub tensiune. În acest caz au loc următoarele acţiuni:

- Ambele registre de mascare a porturilor sînt iniţializate pentru a inhiba toţi biţii de date ai porturilor;
- Liniile de date ale magistralelor porturilor trec în starea de înaltă impedanţă şi semnalele de conversaţie sînt inactivate. Modul 1 este selectat automat;
- Registrele vectorilor de adresă nu sînt iniţializate;
- Ambele bistabile de validare a intreruperilor din port sînt initializate;
- Ambele registre de ieşire ale porturilor sînt iniţializate.

Circuitul PIO poate fi iniţializat aplicînd un semnal */M1* în absenţa unui semnal */RD* sau */IORQ*, rezultatul fiind iniţializarea circuitului imediat după ce */M1* devine inactiv. După ce intră în starea iniţială, circuitul PIO rămîne în această stare pînă la primirea unui cuvînt de control de la procesor.

4.1.6 Programarea circuitului Z80-PIO

Programarea unui port în modurile 0, 1 sau 2 necesită două cuvinte pentru fiecare port. Al treilea cuvînt este trimis numai atunci cînd se doreşte validarea/invalidarea întreruperilor.

Primul cuvînt este *cuvîntul de selectare a modului de operare*. Structura cuvîntului este următoarea:

D7	D6	D5	D4	D3	D2	D1	D0
M1	M2	X	X	1	1	1	1

unde:

- D0-D3 identifică cuvîntul de selectare a modului de operare;
- D4, D5 nu contează;
- D6, D7 determină modul de operare după cum urmează:

M0	*M1*	*MOD*
0	0	ieşire
0	1	intrare
1	0	bidirecţional
1	1	bit

Al doilea cuvînt este *vectorul de întrerupere*, cuvînt care trebuie furnizat de circuitul PIO care a cerut întreruperea, în cazul în care aceasta a fost acceptată. Structura cuvîntului este următoarea:

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

unde:

- D0 identifică vectorul de întrerupere;
- D1-D7 reprezintă vectorul de întrerupere fixat de utilizator.

Programarea unui port în modul 3 necesită un cuvînt de control, vector de întrerupere (dacă întreruperile sînt activate) şi încă trei cuvinte care vor fi descrise în continuare.

Cuvîntul registrului de control intrare/ieşire defineşte care linii ale portului sînt intrări şi care sînt ieşiri. Structura cuvîntului este următoarea:

D7	D6	D5	D4	D3	D2	D1	D0

unde:

- un bit 0 defineşte o ieşire;
- un bit 1 defineşte o intrare.

În modul 3 semnalele conversaţionale nu sînt folosite. Întreruperile sînt generate ca funcţii logice aplicate liniilor considerate intrări.

Cuvîntul de control al întreruperii fixează condiţiile şi nivelele logice necesare generării semnalului de întrerupere. Structura cuvîntului este următoarea:

D7	D6	D5	D4	D3	D2	D1	D0
I3	I2	I1	I0	0	1	1	1

unde:

- D3 - D0 identifică cuvîntul de control al întreruperii
- D4 = 0 - nu urmează cuvănt mască, 1 - urmează cuvînt mască
- D5 = 0 - semnale active în stare Low, 1 - semnale active în stare High
- D6 = 0 - întrerupere la funcţia SAU logic, 1 - întrerupere la funcţia ŞI logic
- D7 = 0 - dezactivare întreruperi, 1 - activare întreruperi.

Cuvîntul mască permite ca orice bit nefolosit din port să fie mascat. Dacă se doreşte acest lucru, atunci bitul D4 din cuvîntul de control al întreruperii trebuie setat, iar următorul cuvînt scris în port trebuie să fie cuvîntul mască. Structura cuvîntului este următoarea:

D7	D6	D5	D4	D3	D2	D1	D0

unde un bit este monitorizat dacă este definit ca ieşire, iar bitul mască este pus în 0 logic.

Pentru invalidarea întreruperilor unui port se poate folosi *cuvîntul de dezactivare întreruperi*. Se poate utiliza fără a schimba restul cuvîntului de control al întreruperilor şi în acest mod conţinutul bistabilului de validare a întreruperilor. Structura cuvîntului este următoarea:

D7	D6	D5	D4	D3	D2	D1	D0
I	X	X	X	0	0	1	1

unde:

- D3-D0 identifică cuvîntul de dezactivare întreruperi
- D4-D6 pot lua orice valori
- D7 = 0 - invalidare întreruperi, 1 - validare întreruperi.

Dacă apare o cerere de întrerupere, în timp ce procesorul înscrie cuvîntul de dezactivare a întreruperilor în PIO (03H), poate să apară o problemă în sistem. Dacă întreruperile sînt validate de procesor, acesta va accepta întreruperea cerută de PIO. Totuşi în acest timp circuitul

PIO, primind cuvîntul de dezactivare a întreruperilor nu va trimite vectorul de întrerupere în timpul ciclului de recunoaştere a întreruperilor. Ca urmare, procesorul va prelua de pe magistrala de date, un vector eronat. Soluţia pentru evitarea acestor erori este să se dezactiveze întreruperile în procesor cu o instrucţiune *DI*, chiar înainte de dezactivarea întreruperilor circuitului PIO, şi să se valideze din nou întreruperile cu o instrucţiune *EI*, după aceea. Aceasta determină procesorul să ignore eventualele cereri de întrerupere de la circuitul PIO în timpul dezactivării lui.

4.1.7 Întrebări

I. Descrieţi modurile de funcţionare ale circuitului PIO.

II. Daţi un exemplu de programare al circuitului PIO în modul 3 cu portul B ca port de ieşire.

Răspuns:

```
LD  A, 0FFH
OUT (83H), A
INC A
OUT (83H), A
```

III. Cum se programează circuitul Z80-PIO?

IV. Cum se poate realiza un transfer între PIO şi un periferic, fără întreruperi?

4.1.8 Aplicaţie: Comanda motorului de curent continuu

Programul va conţine trei părţi, prezentate în figura 4.4:

- programarea circuitului PIO,
- comanda motorului,
- procedura de întîrziere.

Programarea circuitului PIO: Portul B al circuitului este legat la portul de intrare al plăcii de aplicaţii. Acest port va trebui comandat pentru a trimite date plăcii. Deoarece nu este nevoie de semnalele de conversaţie ale portului, şi nici de un vector de întrerupere, portul B al circuitului PIO va fi programat în mod bit, cu numai două cuvinte de comandă.

Primul cuvînt, cuvîntul de selectare a modului de operare, va fi 0FFH. Cel de-al doilea cuvînt, cuvîntul registrului de control intrare/ieşire, va fi 00H (adică toţi biţii portului sînt biţi de ieşire).

Comanda motorului: Motorul de curent continuu de pe placa de aplicaţii poate fi comandat cu ajutorul biţilor 6 şi 7 ai portului de intrare, care este legat la portul B al circuitului PIO. Motorul poate fi pornit, caz în care accelerează pînă la viteza maximă, sau oprit. Semnificaţia biţilor de comandă este prezentată în tabelul 4.1.

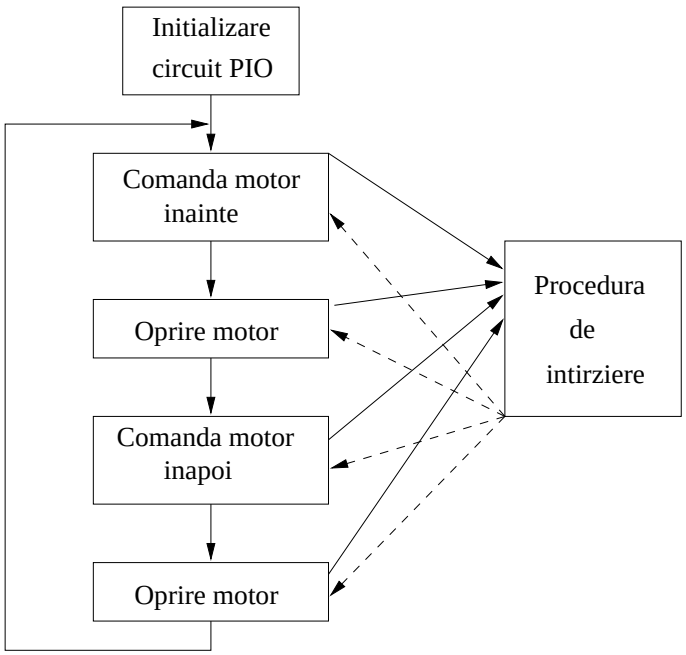

Figura 4.4: Etapele de programare

Bit 7	*Bit 6*	*Actiune*
0	0	Motor oprit
0	1	Mişcare înainte
1	0	Mişcare înapoi
1	1	Motor oprit

Tabelul 4.1: Semnificaţia biţiilor de comandă.

Procedura de întîrziere: Durata buclei de întîrziere se calculează cunoscînd frecvenţa de ceas a sistemului şi numărul de perioade de ceas necesare instrucţiunilor. Frecvenţa sistemului este 1.79 MHz, de unde rezultă că perioada ceasului este 0.5586 ms (în calcule se consideră 0.56 ms). În continuare se va programa o buclă de întîrziere de 1 s. Instrucţiunea `DEC` consumă 4 perioade de ceas, iar instrucţiunea `JP NZ, nn` consumă 10 perioade de ceas, deci este nevoie de $14 \times 255 = 3570$ perioade de ceas pentru a decrementa un registru care conţine data 0FFH pînă la 0. Deci sînt necesare $255 \times 3570 = 910350$ perioade de ceas pentru a decrementa un registru care conţine data 0FFFFH pînă la 0. Această operaţie durează $0.56ms \times 910350 = 0.509796s$. Prin urmare, realizînd această operaţie de două ori, se obţine o întîrziere de aproximativ 1 s.

Programul scris în limbaj de asamblare

```
****;  comanda motorului 1s înainte, 1s stop, 1s înapoi

1800                        ORG  1800H
1800    3E FF               LD   A,0FFH   ; programare circuit PIO
1802    D3 83               OUT  (83H),A  ; mod 3
```

```
1804    3C                  INC  A         ; A=0
1805    D3 83               OUT  (83H),A   ; B port de ieşire
1807    3E 40   START:      LD   A,40H     ; bit 6 = 1, mişcare înainte
1809    D3 81               OUT  (81H),A   ; trimis la port B
180B    CD 24 18            CALL DELAY     ; apel procedură întîrziere (1 s)
180E    AF                  XOR  A         ; A=0
180F    D3 81               OUT  (81H),A   ; oprire motor
1811    CD 24 18            CALL DELAY     ; repaus 1 s
1814    3E 80               LD   A,80H     ; bit 7 = 1, mişcare înapoi
1816    D3 81               OUT  (81H),A
1818    CD 24 18            CALL DELAY     ; timp de 1 s
181B    AF                  XOR  A         ; A=0
181C    D3 81               OUT  (81H),A   ; oprire motor
181E    CD 24 18            CALL DELAY     ; repaus 1 s
1821    C3 07 18            JP   START     ; reia ciclul
1824    2E 02    DELAY:     LD   L,2
1826    01 FF FF LOOP2:     LD BC,0FFFFH   ; aproximativ 1/2 s
1829    0D       LOOP1:     DEC C
182A    C2 29 18            JP   NZ,LOOP1 ; repetă pînă C=0
182D    05                  DEC  B
182E    C2 29 18            JP   NZ,LOOP1 ; repetă pînă B=0
1831    2D                  DEC  L
1832    C2 26 18            JP   NZ,LOOP2 ; întîrzie încă 1/2 s
1835    C9                  RET           ; întoarcere în program
                            END
```

Înainte de a executa programul pe machetă, comutatorul SW2-2 trebuie fixat pe poziţia "MOTOR".

4.1.9 Experimente:

I. Comandaţi motorul de curent continuu de pe placa de aplicaţii conform graficului din figura 4.5.

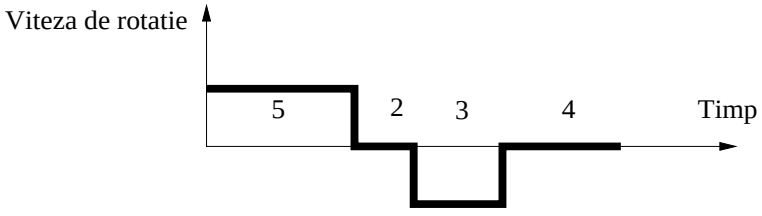

Figura 4.5: Grafic de funcţionare a motorului de curent continuu.

II. Cum se poate obţine comanda motorului de curent continuu astfel încît să se obţină o pantă de viteză? Realizaţi un program care să comande motorul conform graficului din figura 4.6. Figura prezintă atit profilul de viteză ideal cît şi cel real pentru motorul de curent continuu.

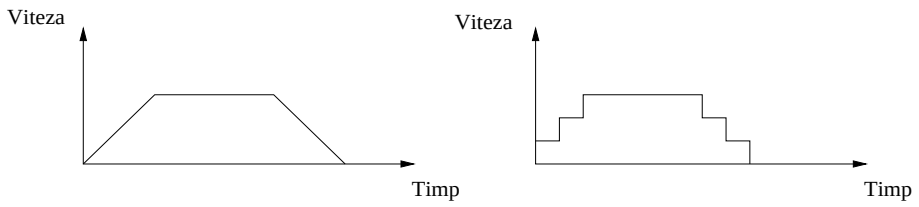

Figura 4.6: Graficul ideal (a) şi real (b) în regim accelerat/decelerat.

Lucrarea 5

Aplicaţii cu circuitul Z80-CTC

Această lucrare prezintă circuitul numărător/temporizator Z80-CTC şi modul de utilizare a acestuia.

5.1 Prezentare generală şi arhitectura internă a circuitului Z80-CTC

Z80-CTC (Counter/Timer Circuit - engl.) este un circuit cu patru canale ce pot funcţiona în mod numărător sau temporizator. Acest circuit poate fi folosit pentru o gamă largă de aplicaţii de numărare: numărare de evenimente, cronometrări de intervale de timp, generare de întreruperi şi generarea unui semnal de ceas. Cele patru canale sînt programabile independent în două moduri de lucru. Circuitul Z80-CTC se conectează direct (pin la pin) la circuitul microprocesor Z80-CPU. Fiecare canal se programează cu doi octeţi de comandă. Cînd se activează întreruperile, este necesar încă un octet suplimentar ce semnifică vectorul de întrerupere. După programare, circuitul numără descrescător pînă la zero. Apoi, se reîncarcă automat (dintr-un registru) şi reia procesul de numărare descrescătoare. Prin utilizarea circuitului CTC se pot elimina buclele de întîrziere implementate prin program. Lucrul cu întreruperile este simplificat, deoarece circuitului i se trimite un singur vector de întrerupere, iar acesta generează intern cîte un vector pentru fiecare canal. Semnalul de ceas monofazic este primit de la procesor. Figura 5.1 prezintă structura internă a circuitului Z80-CTC. Simbolul circuitului este prezentat în figura 5.2.

5.1.1 Semnificaţia pinilor circuitului Z80-CTC

Semnale generale:

CLK (System Clock) - Semnal de ceas comun tuturor circuitelor din sistemul cu microprocesor Z80.

/RESET (Reset) - Semnal de iniţializare. Activarea acestui semnal conduce la terminarea tuturor acţiunilor de numărare descrescătoare şi dezactivarea tuturor întreruperilor. Biţii de întrerupere din registrele de control ale canalelor sînt resetaţi. Ieşirile de întreruperi şi *ZC/TO* devin inactive. *IEO* ia valoarea lui *IEI*. Magistrala de date este trecută în starea de înaltă impedanţă.

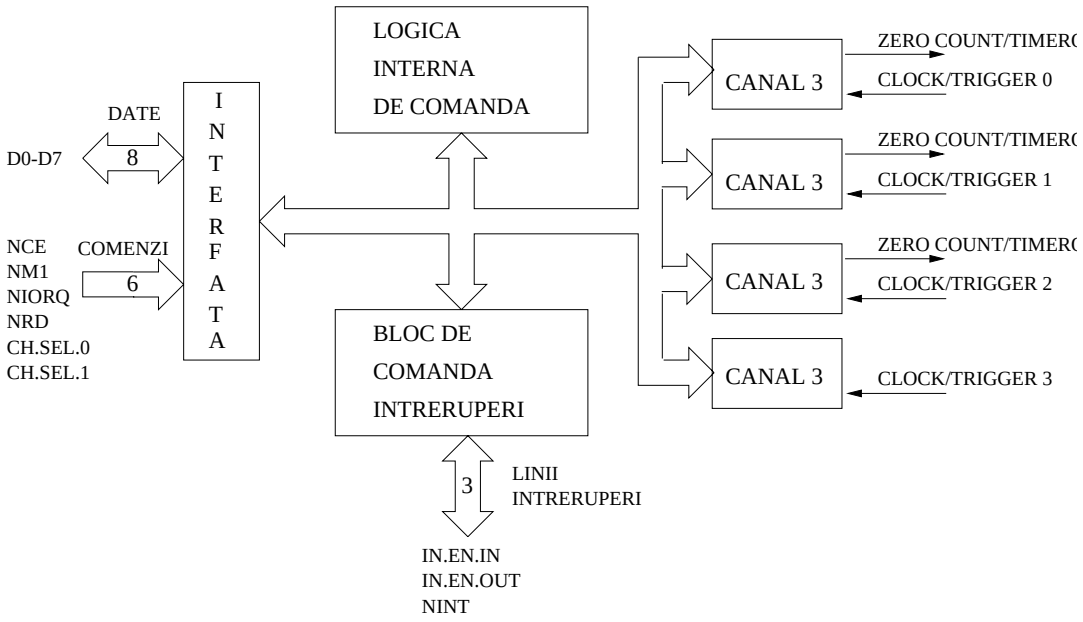

Figura 5.1: Structura internă a circuitului Z80-CTC.

Magistrala de date:

D0-D7 (System Data Bus) - Magistrală bidirecţională, conectată la magistrala de date a procesorului.

Semnalele de control (primite de la procesor):

CS0-CS1 (Channel Select) - Semnale care formează o adresă de doi biţi cu care se selectează unul din cele patru canale ale circuitului pentru o operaţie de scriere sau citire. De obicei, aceşti pini se leagă la pinii *A0-A1* ai magistralei de adrese a procesorului. Modul de selecţie a canalelor este prezentat în tabelul 5.1.

	CS1	*CS0*
Ch0	0	0
Ch1	0	1
Ch2	1	0
Ch3	1	1

Tabelul 5.1: Selecţia canalelor cu biţii CS0 şi CS1.

/CE (Chip Enable) - Semnal de validare a chip-ului. Semnalul este activat cînd circuitul acceptă cuvinte de control, vectori de întrerupere sau constante de timp de pe magistrala de date, în timpul unui ciclu de scriere la dispozitivele de intrare/ieşire sau cînd se transmite procesorului conţinutul unui numărător în timpul unui ciclu de citire de la dispozitivele de intrare/ieşire. În majoritatea aplicaţiilor, acest semnal este decodificat din cei mai puţin semnificativi opt biţi ai magistralei de adrese pentru oricare din cele patru adrese de intrare/ieşire care sînt asociate celor patru canale ale circuitului.

/M1 (Machine Cycle 1) - Semnal provenit de la pinul */M1* al procesorului. Cînd */M1* şi */IORQ* sînt active, procesorul Z80 acceptă o întrerupere. Apoi, dacă are prioritatea cea mai mare şi dacă unul din canale a cerut o întrerupere (prin activarea semnalului */INT*), circuitul CTC

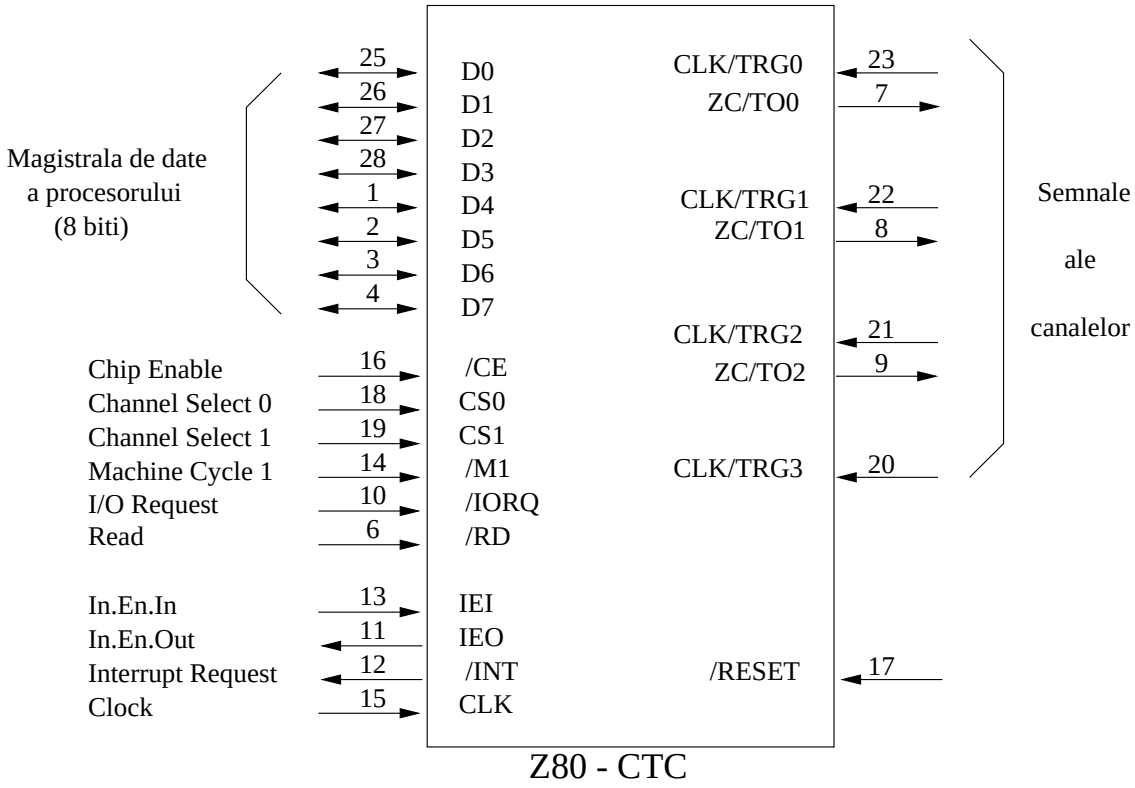

Figura 5.2: Simbolul circuitului Z80-CTC.

plasează vectorul de întrerupere pe magistrala de date.

/IORQ (Input/Output Request) - Semnal provenit de la pinul */IORQ* al procesorului. Semnalul este folosit în conjuncţie cu */CE* şi */RD* pentru a transfera date şi cuvinte de control pentru canale între procesor şi CTC. În timpul unui ciclu de scriere, semnalele */IORQ* şi */CE* trebuie să fie active, iar semnalul */RD* trebuie să fie inactiv. Circuitul CTC nu primeşte un semnal specific de scriere, ci îşi generează unul intern din inversul unui semnal */RD*. Într-un ciclu de citire, */IORQ*, */RD* şi */CE* trebuie să fie active pentru ca procesorul să poată citi conţinutul unui numărător.

/RD - (Read Cycle Status) - Semnal provenit de la pinul */RD* al procesorului. Semnalul este folosit în conjuncţie cu */IORQ* şi */CE* pentru a transfera date şi cuvinte de control între CTC şi procesor.

Semnale de întrerupere:

IN.EN.IN (IEI) - (Interrupt Enable In) - Un semnal cu valoare logică 1 pe această linie semnifică faptul că nici un alt dispozitiv periferic cu prioritate mai mare în lanţul de întreruperi nu este deservit de către procesor.

IN.EN.OUT (IEO) - (Interrupt Enable Out) - Semnal folosit în conjuncţie cu *IEI* pentru a forma un sistem de întreruperi ierarhizat. Linia este în 1 logic numai dacă linia *IEI* este în aceeaşi stare şi procesorul nu deserveşte o întrerupere de la unul din canalele circuitului. Semnalul blochează dispozitivele cu prioritate mai mică pentru ca acestea să nu poată întrerupe un dispozitiv cu prioritate mai mare în timp ce este deservit de procesor.

/INT - (Interrupt Request) - Semnal activ cînd numărătorul unui canal al circuitului CTC, programat să activeze semnalul de întrerupere, a ajuns la zero.

Semnalele canalelor:

CLK/TRG0-CLK/TRG3 - (External Clock/Timer Trigger) - Patru semnale ce corespund celor patru canale ale circuitului. În mod *numărător*, fiecare front activ pe acest pin decrementează numărătorul. În mod *timer*, un front activ al semnalului porneşte timerul. Utilizatorul poate selecta frontul activ fie crescător, fie descrescător.

ZC/TO0-ZC/TO2 - (Zero Count/Timeout) - Trei semnale ce corespund canalelor 0-2 ale circuitului. În ambele moduri, ieşirea prezintă un impuls cu valoare 1 logic cînd numărătorul ajunge la zero.

Funcţiile semnalelor de la pinii circuitului Z80-CTC sînt prezentate centralizat în tabelul 5.2.

Nume	*Funcţie*	*Tip*
Semnale generale		
CLK	Tact sistem	Intrare
/RESET	Reset sistem	Intrare
	Magistrală de date	
D0-D7	Bus date	Bidir. 3-stări
Semnale de control		
CS0-CS2	Selecţie canal	Intrare
/CE	Validare chip	Intrare
/M1	Ciclu maşină 1	Intrare
/IORQ	Cerere I/O	Intrare
/RD	Citire	Intrare
Semnale de întrerupere		
IEI	Activare întreruperi	Intrare
IEO	Inactivare întreruperi	Ieşire
/INT	Cerere de întrerupere	Intrare
Semnale ale canalelor		
CLK/TRG0-CLK/TRG3	Ceas extern	Intrare
ZC/TO0-ZC/TO2	Sfîrşit numărare	Intrare

Tabelul 5.2: Funcţiile pinilor circuitului Z80-CTC.

5.1.2 Structura unui canal

Un canal este compus din două registre de cîte opt biţi, două număratoare şi logică de control. Un registru este folosit pentru a memora constanta de timp, iar celălalt pentru a memora modul de lucru şi parametrii canalului. Un numărător descrescător pe 8 biţi poate fi citit de către procesor. Un alt numărător de opt biţi implementează un divizor de frecvenţă al semnalului de ceas. Figura 5.3 prezintă structura unui canal al circuitului Z80-CTC.

Registrul de control al canalului este înscris de către procesor pentru a selecta modul de lucru şi parametrii canalului. În circuit sînt patru asemenea registre, corespunzînd celor patru canale Selectarea registrului în care se scrie se face cu pinii *CS1* şi *CS0*.

Divizorul este folosit numai în modul *timer*. Acesta este un dispozitiv de numărare pe 8 biţi care este programat de procesor prin registrul de control al canalului, el divizînd semnalul de

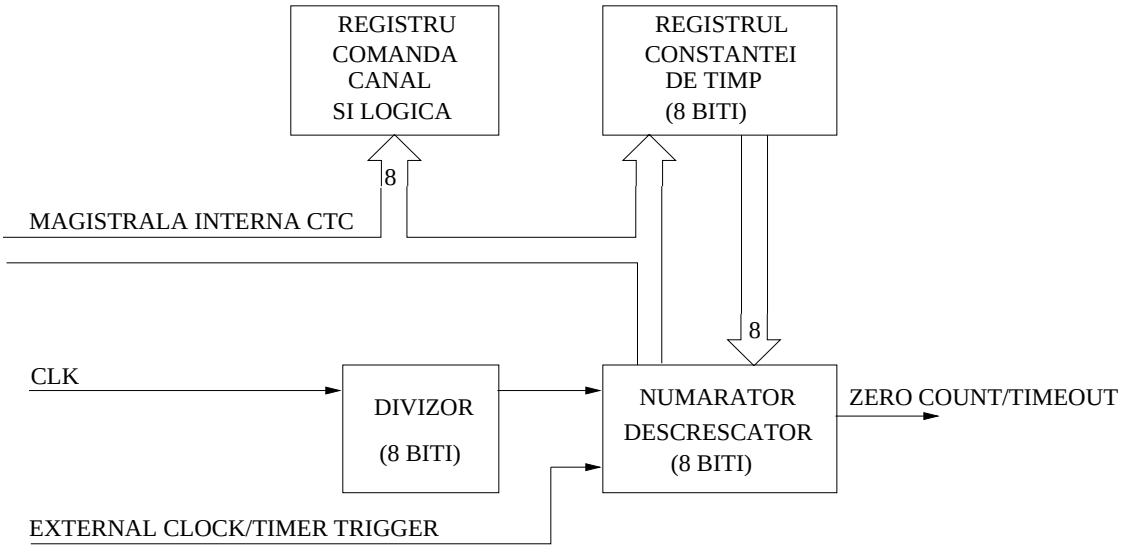

Figura 5.3: Schema bloc a unui canal a circuitului Z80-CTC.

intrare (ceasul sistemului). Ieşirea divizorului este folosită ca intrare de ceas pentru numărătorul descrescător. Registrul constantei de timp este un registru de 8 biţi folosit în ambele moduri de funcţionare. Acest registru este înscris imediat după registrul de control al canalului. Constanta de timp este o valoare întreagă între 1 şi 256 (256 este codificat cu 8 biţi de zero). Această constantă este automat încărcată în numărătorul descrescător atunci cînd canalul este iniţializat sau de fiecare dată cînd numărătorul ajunge la zero.

Numărătorul descrescător este un dispozitiv de numărare pe 8 biţi folosit în ambele moduri de funcţionare. Înainte de fiecare ciclu de numărare el este încărcat cu valoarea conţinută în registrul constantei de timp. Numărătorul este decrementat pe frontul activ al ceasului extern în modul numărător sau pe cel al ieşirii de ceas dată de divizor. Valoarea conţinută în numărător poate fi citită de către procesor în orice moment printr-o operaţie de citire de la adresa portului ce a fost asociat canalului respectiv. Canalele 0, 1 şi 2 pot fi programate ca atunci cînd ajung la zero să genereze o întrerupere. Datorită limitărilor de pini, canalul 3 nu are această posibilitate. Canalul 3 poate fi folosit numai în aplicaţiile care nu trebuie să genereze semnal de întrerupere.

5.2 Modurile de lucru ale circuitului Z80-CTC

La punerea sub tensiune, starea circuitului Z80-CTC este necunoscută. Prin activarea semnalului */RESET* se aduce circuitul într-o stare iniţială, cunoscută. Pentru a putea folosi un canal pentru numărare, acesta trebuie programat cu un cuvînt de control şi o constantă de timp. Dacă un canal a fost programat să activeze semnalul de întrerupere, trebuie programat şi un vector de întrerupere. După programarea unui canal prin trimiterea cuvintelor de control, acesta va începe să funcţioneze conform modului programat: *numărător* sau *timer*.

Modul numărător

În modul *numărător* circuitul numără fronturile active ale intrării de ceas extern *CLK/TRG*. Circuitul numărător este încărcat cu constanta de timp şi la fiecare eveniment extern este decrementat pînă cînd ajunge la zero. Numărătoarele 0, 1 şi 2 pot fi programate să genereze o întrerupere în acel moment. În acelaşi timp, sînt încărcate automat cu valoarea conţinută

în registrul constantei de timp, fără să se întrerupă procesul de numărare. Dacă în registrul constantei de timp se înscrie o nouă valoare în timp ce numărătorul funcţionează, se termină mai întîi numărătoarea curentă şi abia apoi se va încărca noua valoare în numărător.

Modul timer

În modul *timer* circuitul generează semnale cu perioada multiplu de perioada ceasului sistem. Pentru a realiza acest lucru sînt folosite divizoarele aferente fiecărui canal. Divizarea ceasului sistem se face în două etape: prima în divizor (cu 16 sau 256), iar a doua în numărător (cu valoarea înscrisă în registrul constantei de timp). Şi în acest mod numărătoarele 0, 1 şi 2 pot genera o întrerupere atunci cînd ajung la zero. Se obţine un semnal cu perioada:

$$t_C \times D \times C_T$$

unde:
t_C este perioada ceasului sistem,
D este factorul de divizare programat,
C_T este constanta de timp programată.

Circuitul poate fi programat să numere imediat după ce a fost iniţializat (numărătoarea porneşte odată cu ciclul procesor ce urmează celui în care a fost înscris registrul constantei de timp) sau la frontul activ al semnalului de triggerare *CLK/TRG* (numărătoarea începe la al doilea front activ al semnalului de trigerare, după ce a fost încărcată constanta de timp).

5.3 Blocul de comandă a întreruperilor

Blocul de comandă a înteruperilor asigură interfaţarea întreruperilor circuitului CTC cu sistemul de întreruperi ierarhizate al procesorului Z80. Semnalele cu care se asigură corelarea cu celelalte dispozitive periferice sînt *IEI* şi *IEO*. În timp ce o cerere de întrerupere a circuitului CTC este deservită de procesor, blocul de comandă a întreruperilor ţine semnalul *IEO* în 0 logic, inhibînd astfel întreruperile dispozitivelor mai puţin prioritare. Cînd *IEI* devine 0, blocul de comandă a întreruperilor poate genera o întrerupere. Figura 5.4 prezintă strucura unui lanţ de întreruperi cu priorităţi ierarhizate.

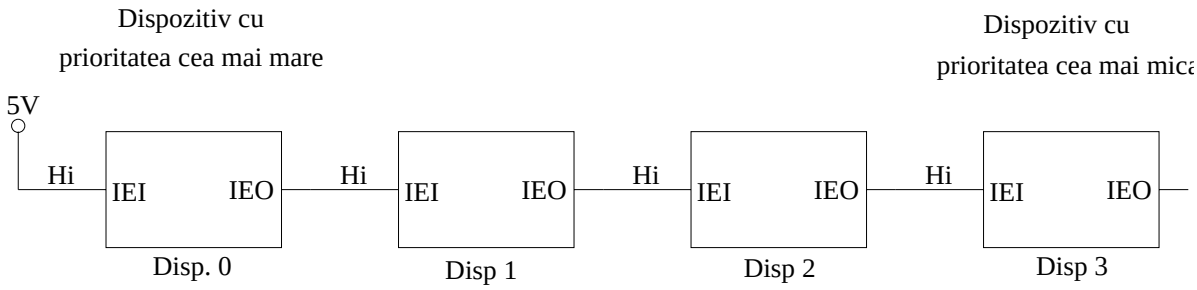

Figura 5.4: Lanţ de dispozitive cu priorităţi ierarhizate.

Dacă un canal este programat să genereze o întrerupere, blocul de comandă a întreruperilor pune linia *IEO* în 0 logic atunci cînd numărătorul canalului respectiv ajunge la zero. Simultan, se activează semnalul */INT*. La răspunsul procesorului (*/M1* şi */IORQ*), acest bloc pune pe magistrala de date vectorul de întrerupere corespunzător canalului care a generat întreruperea. Totodată, acest bloc arbitrează priorităţile întreruperilor în circuitul CTC. Sistemul este identic cu cel al procesorului Z80, canalul 0 avînd prioritatea cea mai ridicată. Blocul de comandă

a întreruperilor monitorizează magistrala de date şi decodifică instrucţiunea de întoarcere din întrerupere (RETI). Dacă o întrerupere este în aşteptare, blocul ţine semnalul *IEO* în 0 logic. Cînd apare instrucţiunea RETI (2 octeţi), dispozitivul comută linia *IEO* în 1 logic pe durata unui ciclu maşină (*M1*) ca să se asigure că dispozitivele cu prioritate mai mică vor decodifica întreaga instrucţiune RETI şi se vor iniţializa corespunzător.

5.4 Iniţializarea circuitului Z80-CTC

Circuitul Z80-CTC are două moduri de iniţializare: *hardware* şi *software*.

Iniţializarea *hardware* încheie toate operaţiile de numărare. În acest caz, au loc următoarele acţiuni:

- dezactivarea tuturor întreruperilor circuitului CTC şi resetarea biţilor de întrerupere din registrele de control ale canalelor;
- inactivarea semnalelor *ZC/TO* şi *INT*;
- semnalul *IEO* ia valoarea semnalului *IEI*;
- magistrala de date este trecută în starea de înaltă impedanţă.

Toate canalele trebuie să fie complet reprogramate după acest tip de iniţializare.

Iniţializarea *software* este controlată de bitul 1 din registrul de control al canalului. Cînd un canal este iniţializat software se opreşte numărarea. În momentul iniţializării software, ceilalţi biţi din cuvîntul de control modifică parametrii canalului. După o astfel de iniţializare trebuie înscrisă o nouă constantă de timp în registrul corespunzător al aceluiaşi canal.

5.5 Programarea circuitului Z80-CTC

Fiecare canal al circuitului trebuie programat înainte de funcţionare. Programarea constă în înscrierea a două cuvinte de control la adresa portului ce corespunde canalului dorit. Primul cuvînt de control selectează modul de operare şi parametrii canalului. Al doilea cuvînt reprezintă constanta de timp, care este o dată binară cu valoare între 1 şi 256. Constanta de timp trebuie să fie precedată de cuvîntul de control al canalului. După iniţializare, canalele pot fi reprogramate la orice moment. Dacă pentru un canal sînt activate întreruperile, atunci mai este nevoie de un cuvînt de comandă ce reprezintă vectorul de întrerupere. Este necesar un singur vector de întrerupere, deoarece circuitul generează intern vectori diferiţi pentru fiecare canal. Structura cuvîntului de control pentru un canal este următoarea:

D7	D6	D5	D4	D3	D2	D1	1

Biţii au următoarea semnificaţie:

- D0 = 1 identificator pentru cuvîntul de control;
- D1 = 0 - continuarea funcţionării, 1 - iniţializare software;

- D2 = 0 - nu urmează constanta de timp, 1 - urmează constanta de timp;
- D3 = 0 - triggerare automată după încărcarea constantei de timp, 1 - triggerare externă cu semnalul CLK/TRG;
- D4 = 0 - frontul activ este descrescător, 1 - frontul activ este crescător;
- D5 = 0 - factorul de divizare are valoarea 16, 1 - factorul de divizare are valoarea 256;
- D6 = 0 - funcţionare în mod 'timer', 1 - funcţionare în mod 'numărător';
- D7 = 0 - dezactivare întreruperi, 1 - activare întreruperi.

Întreruperile pot fi programate în orice mod de funcţionare şi pot fi activate sau dezactivate în orice moment. Reprogramarea frontului activ al semnalului *CLK/TRG* în timpul funcţionării este echivalentă cu inserarea unui front activ în semnal. Dacă numărătorul aşteaptă un eveniment în timpul reprogramării (în ambele moduri), aceasta nu va interveni în procesul de numărare. Odată pornit, numărătorul funcţionează neîntrerupt pînă este oprit prin iniţializare. Numărătorul nu poate funcţiona fără o constantă de timp. Aceasta se înscrie în registrul corespunzător în urma unui cuvînt de control care are bitul 2 setat. Structura cuvîntului de cod care stabileşte constanta de timp este următoarea: Cei 8 biţi codifică un număr binar între 1 şi 256.

CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

Valoarea 0 pe toţi cei 8 biţi semnifică o constantă de timp egală cu 256. Dacă Z80-CTC are una sau mai multe întreruperi activate, atunci trebuie să i se furnizeze un vector de întrerupere. Din acest cuvînt trebuie programaţi numai cei mai semnificativi 5 biţi, deoarece ceilalţi 3 biţi sînt completaţi de circuitul CTC. Structura cuvîntului de stabilire a vectorului de întrerupere este următoarea:

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	CS1	CS0	0

unde:

- D0 = 0 identificatorul vectorului de întrerupere,
- CS1-CS0 reprezintă semnalele de selecţie a canalelor (automat introduşi de circuitul CTC),
- V7-V3 cei 5 biţi programaţi de către utilizator.

5.5.1 Realizarea unui ceas folosind circuitul Z80-CTC

Realizarea ceasului implică trei etape:

- programarea circuitului Z80-CTC,
- scrierea rutinei de servire a întreruperii,
- actualizarea şi afişarea orei (numerele binare vor fi convertite în cod BCD pentru a uşura citirea orei.

Programul este conceput astfel încît să se înceapă măsurarea timpului de la o ora prestabilită. Această ora poate fi stabilită modificînd valorile iniţiale din buffer-ul de timp.

Programarea circuitului Z80-CTC

Pentru acest program este nevoie de un singur canal al circuitului CTC, folosit în modul timer. Ceasul timer-ului va fi ceasul sistemului. Problema principală va fi măsurarea secundei. Vom considera factorul de divizare 256. Ramîne problema stabilirii constantei de timp.

$256 \times 0.56\mu s = 0.00014336s$

$1/0.00014336 = 6975.446428571$

$6975.446428571/218 = 31.99746068152$

Deci, $32 \times 218 \times 256 \times 0.56\mu s = 1.00007936s$, o aproximare destul de bună a secundei. Eroarea este de $0.08ms$ la o secundă, adică aproape 8 secunde la 24 ore. Prin urmare programarea circuitului CTC se va face cu trei cuvinte astfel:

cuvîntul de control canalului 101101012 = 0B5H

- b7=1 întreruperi activate,
- b6=0 mod timer,
- b5=1 factor de divizare 256,
- b4=1 numărare la front crescător,
- b3=0 folosirea ceasului intern,
- b2=1 urmează constanta de timp,
- b1=0 nu se resetează numărătorul,
- b0=1 identificarea cuvîntului de control

constanta de timp 21810 = 0DAH Înmulţirea cu al treilea factor - 32 - se va face în rutină de servire a întreruperii, imcrementînd contorul secundelor numai cînd contorul întreruperilor ajunge la 32.

vectorul de întrerupere 0A8H În prealabil registrul I va fi încărcat cu valoarea 18H, deoarece memoria disponibilă utilizatorului se află între adresele 1800-1F9FH.

Actualizarea şi afişarea orei

În rutina de servire a întreruperii se va testa condiţia de scurgere a unei secunde (contorul întreruperilor are valoarea 32). După trecerea unei secunde se va incrementa contorul secundelor şi se va testa depăşirea valorii maxime pentru secunde, minute şi ore (60, 60 respecţiv 12 sau 24). După incrementarea unui contor, se va face şi ajustarea zecimală a numerelor. În cazul atingerii valorii maxime pentru un contor, acesta va lua valoarea zero şi va fi incrementat contorul unităţii de măsură superioare (dacă este posibil). Afişarea orei se face convertind mai întîi numerele din buffer-ul de afişare în formatul de afişare cu 7 segmente, după care se apelează procedura SCAN. Cifrele care reprezintă orele, minutele şi secundele vor fi despărţite de puncte zecimale. În figura 5.5 este prezentată schema logică a programului.

5.5.2 Întrebări:

I. Care sînt condiţiile de scriere/citire pentru circuitul CTC-Z80?

Răspuns

Din punct de vedere hardware condiţiile pentru scrierea/citirea circuitului CTC-Z80 sînt următoarele:

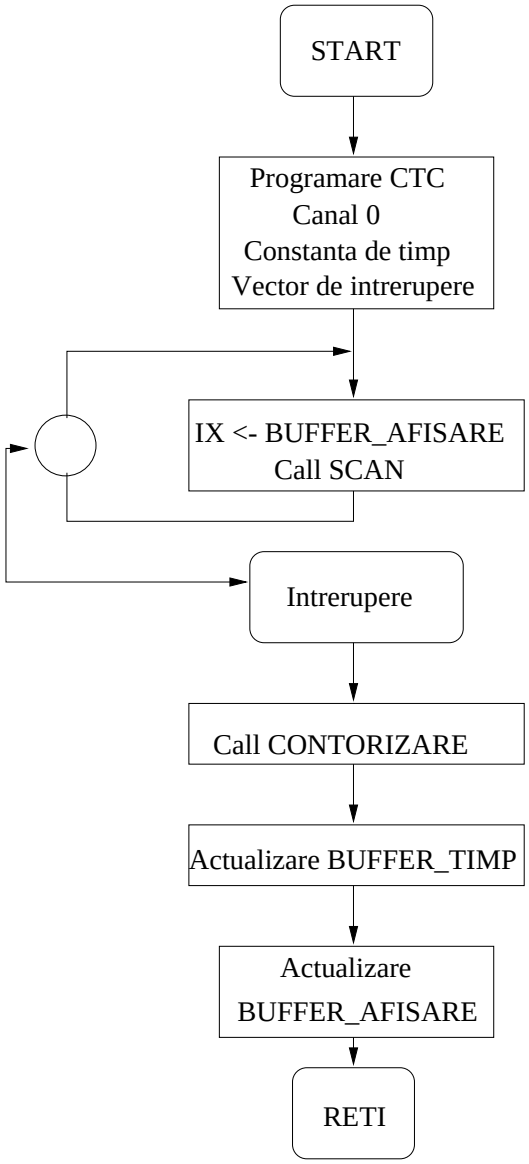

Figura 5.5: Schema logică a programului.

- pentru scriere: semnalele /IORQ şi /CE active, /RD inactiv. Circuitul CTC-Z80 generează intern semnalul de citire din inversul semnalului /RD.

- pentru citire: /IORQ, /CE, /RD active.

II. Care este structura unui canal? Pot fi folosite toate cele patru canale în sistem de întreruperii?

Răspuns

Fiecare canal este compus din două registre de 8 biţi, două numărătoare şi logică de control. Un numărător este folosit ca numărătoar descrescător iar altul ca divizor. Registrele folosesc pentru memorarea constantei de timp şi pentru setarea modului de lucru.

Datorită limitării numărului de pinii, canalul 3 nu se poate folosi în sistemul de întreruperi.

III. Cînd se activează semnalul de întrerupere */INT*?

Răspuns

Semnalul de întreruperi este activat de fiecare dată cînd numărătorul canalului respectiv ajunge la zero.

IV. Care sînt modurile de lucru ale CTC-Z80 şi care sînt condiţiile de programare?

Răspuns

Modurile de lucru ale CTC-Z80 sînt:

- mod numărător şi
- mod timer.

Programarea circuitului pentru canalul 0 se face în următorul mod:

```
CTC0 EQU 40H

LD  A,18H
LD  I,A
LD  A,10110101B
OUT (CTC0),A
LD  A,020H
OUT (CTC0),A
LD  A,0A8H
OUT (CTC0),A
IM  2
EI
```

Adresele celor 4 canale sînt 40H, 41H, 42H şi 43H. Cuvîntul de control fiind 10110101B, canalul 0 este programat în mod timer, factorul de divizare este 256. CTC va genera o întrerupere după 8192 (256×32) tacte, constanta de timp fiind 020H.

5.5.3 Experimente

I. Programaţi canalului 2 al circuitului CTC-Z80 în mod numărător?

II. Studiaţi următorul program ce implementează un ceas, conform schemei logice prezentate în figura 5.5. Numărarea secundelor se face cu ajutorul circuitului CTC.

```
*****; Program care implementează un ceas
*****; Pentru măsurarea unei secunde se foloseşte circuitul CTC

ORG             1800H
CTC0    EQU     40H   ; adresa portului la care se găseşte canalul 0
SCAN    EQU     05FEH ; adresa rutinei de afişare
HEX7SG  EQU     0678H ; adresa rutinei de conversie pentru afişare

START:
        LD A,18H
        LD I,A

*****; se completează cu programarea CTC, canalul 0
```

```
....
*****;

MAIN:
        LD   IX,BUFFER_AFISARE
        CALL SCAN             ; apel procedură SCAN
        JR   MAIN

*****; rutina de contorizare
CONTORIZARE:
        LD   DE,BUFFER_TIMP
        LD   A,(DE)           ; se citeşte contorul întreruperilor din buffer
        INC  A                ; se numără întreruperile
        LD   (DE),A           ; se salvează contorul actualizat
        CP   20H              ; într-o secundă apar 32 (20H) întreruperi
        LD   B,4              ; contor şi indicator pentru scurgerea secundei
        RET  NZ
        XOR  A                ; A=0
        DEC  B                ; B=3 (contor)
        LD   (DE),A           ; resetarea contorului de întreruperi
        INC  DE
        LD   HL,VALORI_MAXIME; se încarcă adresa tabelei cu
                              ; valori maxime în HL
LOOP:
        LD   A,(DE)           ; se încarcă în A numărul de secunde,
                              ; minute sau ore
        INC  A
        DAA                   ; ajustare zecimală a acumulatorului
        LD   (DE),A           ; actualizare buffer de timp
        SUB  (HL)             ; verificarea depăşirii valorilor maxime
        RET  C                ; ieşire în caz de nedepăşire a limitelor
        LD   (DE),A
        INC  HL               ; dacă se depăşesc limitele, se creşte
                              ; unitatea următoare
        INC  DE               ; şi se aduce la 0 cea curentă
        DJNZ LOOP
        RET

*****; procedura de conversie a bufferului de afişare
CONVERSIE_7SEG:
        LD   HL,BUFFER_AFISARE ; se pregăteşte rutina de conversie
        LD   DE,SECUNDE
        LD   B,3                 ; contor
LOOP2:  LD   A,(DE)              ; conversie buffer de timp
        CALL HEX7SG
        INC  DE
        DJNZ LOOP2
        DEC  HL
```

```
        DEC  HL
        SET  6,(HL)            ; se setează punctul zecimal pt. ore
        DEC  HL
        DEC  HL
        SET  6,(HL)            ; se setează punctul zecimal pt. minute
        RET

*****; rutina de servire a întreruperii

ORG     18A8H
DEFW    INTRERUPERE    ; la această adresă se află adresa de întrerupere

*****; început a rutinei de întrerupere
INTRERUPERE:
*****; se salvează registrele care vor fi afectate în stivă

        CALL CONTORIZARE       ; apelul rutinei care măsoară timpul
        LD   A,B
        CP   4                 ; se verifică dacă a trecut 1 s
        CALL NZ,CONVERSIE_7SEG ; conversia buffer-ului de timp

*****; rutina pentru afişare

*****; restaurare registre, activare întreruperi, revenire din întreruperi

*****; tabela de valori maxime

*****; se completează cu valorile pentru secunde, minute şi ore
*****; valorile sînt hexazecimale pt. că şi cele cu care se compară sînt
*****; ajustate BCD pt. afişare

*****; buffer-ul de afişare
BUFFER_AFISARE:
        DEFS       6      ; pt. afişare este nevoie de spaţiu pt. 6 caractere
        END
```

III. Completaţi programul cu părţiile comentate.

IV. Ce se va afişa pe display dacă se modifică valorile din tabela de valori maxime?

V. Cîte tacte de ceas sînt necesare pentru a contoriza o secundă?

Partea II

Microprocesorul 8086

Lucrarea 6

Arhitectura şi organizarea microprocesorului 8086

Această lucrare prezintă aspectele fundamentale ale microprocesorului 8086. Se regăsesc aici detalii necesare atît programatorului cît şi proiectantului de sisteme cu microprocesor. Expunerea din această lucrare este un material de referinţă şi pentru lucrările de laborator ce vor urma.

6.1 Arhitectura microprocesorului

Arhitectura microprocesorului 8086 (figura 6.1) prezintă două mari unităţi funcţionale:

- unitatea de interfaţare cu magistrala externă (*Bus Interface Unit* = BIU) şi
- unitatea de execuţie a instrucţiunilor (*Execution Unit* = EU).

Aceste două unităţi operează asincron şi formează un mecanism unitar de aducere şi execuţie a unei instrucţiuni.

În esenţă, procesarea paralelă asigurată de BIU şi EU elimină timpul necesar pentru aducerea multor instrucţiuni prin suprapunerea etapei de aducere a unei instrucţiuni (*fetch*) cu etapa de execuţie a altei instrucţiuni. Ca o consecinţă, bus-ul sistem este utilizat mai eficient şi se obţine o creştere a performanţelor sistemului.

6.1.1 Unitatea de interfaţă cu busul

BIU asigură toate semnalele necesare desfăşurării ciclurilor de magistrală. Această unitate realizează legătura dintre microprocesor şi lumea exterioară. Sarcinile acestei unităţi sînt:

- aducerea instrucţiunilor din memorie şi plasarea acestora în coada de instrucţiuni;
- gestionarea cozii de instrucţiuni;
- realocarea adreselor;
- controlul semnalelor de comandă.

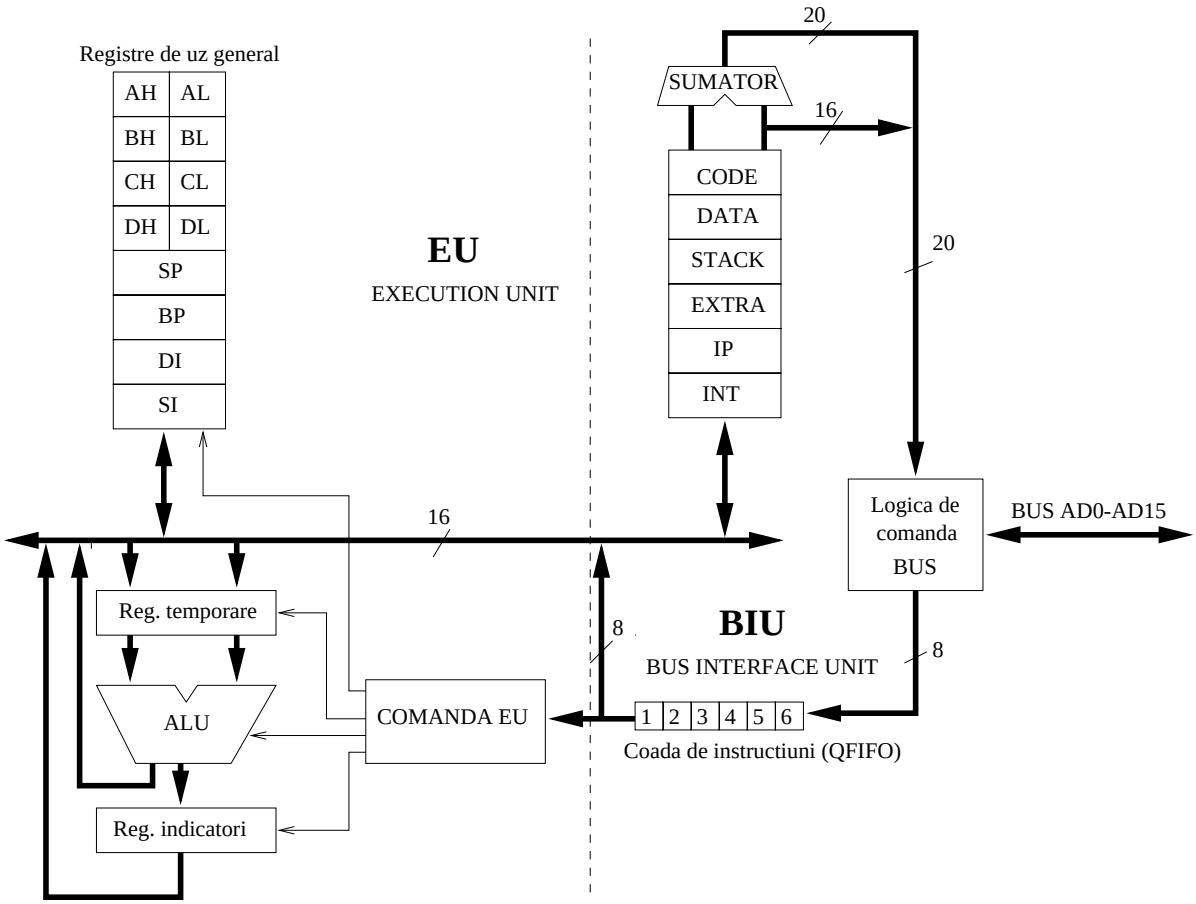

Figura 6.1: Arhitectura microprocesorului 8086.

Pentru implementarea acestor funcţii, *BIU* conţine:

- registre de comunicaţie internă;
- registre pentru segmentarea memoriei;
- sumator de adrese;
- registru indicator de program;
- memorie QFIFO pentru păstrarea instrucţiunilor în aşteptarea execuţiei;
- un bloc pentru controlul semnalelor de comandă.

Suportul fizic al *BIU* permite implementarea unei structuri "pipeline". Dacă la un moment dat în memoria QFIFO există cel puţin doi octeţi liberi şi dacă *EU* nu cere un acces la magistrală, atunci *BIU* face un acces la memorie, aducînd doi octeţi. Este posibilă astfel aducerea în avans (*prefetch*) a maximum şase octeţi din programul executat. Octeţii extraşi din memorie sînt introduşi, octet cu octet, în memoria QFIFO prin capătul de intrare. Conţinutul memoriei QFIFO este utilizat de *EU* prin capătul de ieşire, ceea ce determină deplasarea automată a octeţilor spre ieşire, cu o poziţie după citirea fiecărui octet. Dacă memoria QFIFO nu are la un moment dat nici un bloc liber şi nici *EU* nu cere un acces prin magistrala externă, atunci *BIU* nu iniţiază nici un ciclu al magistralei, ceea ce implică o stare de inactivitate (*idle state*)

a acesteia. Dacă *BIU* a iniţiat un ciclu iar *EU* cere un acces, va fi încheiat ciclul de *BIU* după care se va ceda accesul pentru *EU*.

Pentru generarea adresei de acces la o resursă externă, *BIU* trimite pe magistrala externă o adresă fizică de 20 biţi. Adresa fizică se formează prin însumarea conţinutul unui registru segment de 16 biţi, deplasat la stînga cu patru poziţii, cu un deplasament de 16 biţi, primit de la *EU*. De exemplu, dacă CS = 7100H, iar IP = 9002H, atunci următoarea instrucţiune va fi citită de la adresa: 71000H + 9002H = 7A002H. *BIU* generează şi semnale de comandă (scriere/citire memorie, scriere/citire port, etc.) necesare pentru desfăşurarea unui acces la o resursă externă.

6.1.2 Unitatea de execuţie

EU citeşte din capătul de ieşire al QFIFO octeţii care aparţin unei instrucţiuni. Instrucţiunea adusă în *EU* este decodificată de o unitate specială. Dacă instrucţiunea necesită acces la memorie, se generează către *BIU* un deplasament. Simultan, se transmit şi informaţii care identifică tipul accesului la magistrala externă (citire/scriere din/în memorie/port). După execuţia instrucţiunii, *EU* actualizează starea indicatorilor şi aşteaptă ca următoarea instrucţiune să fie disponibilă în memoria QFIFO. Datorită faptului că *BIU* foloseşte fiecare moment în care magistrala este liberă pentru a încărca memoria QFIFO, este foarte probabil ca în această memorie să se găsească instrucţiuni care urmează a fi executate. Astfel, viteza de prelucrare a informaţiilor de către microprocesorul 8086 este crescută pe baza unui hardware intern suplimentar şi nu prin intermediul creşterii frecvenţei de tact. După execuţia unei instrucţiuni de salt sau ramificaţie, memoria QFIFO este descărcată, pierzînd instrucţiunile care s-ar fi executat dacă n-ar fi fost executată instrucţiunea de salt. Rezultă o concluzie importantă: pentru ca programele să fie executate cît mai rapid este necesar să fie eliminate, pe cît posibil, instrucţiunile ce implică saltul la altă secvenţă de instrucţiuni.

6.1.3 Registrele microprocesorului

Procesorul 8086 are trei grupe de registre interne accesibile utilizatorului:

- *8 registre generale de date*,
- *4 registre segment*,
- *2 registre de stare şi control*.

Registrele generale de date sînt în număr de 8 şi sînt folosite pentru memorarea temporară a unor informaţii. Cele 8 registre fac parte din două categorii:

- *registre de date* şi
- *registre pointer şi index*.

Fiecare *registru de date* (tabelul 6.1) poate fi referit ca registru cuvînt (16 biţi) sau ca două registre semicuvînt (1 byte, 8 biţi). Sufixele H şi L desemnează partea superioară (High) sau inferioară (Low) a registrului de 16 biţi.

Denumire registru general de date	*16 biţi*	*8 biţi*	*Denumire în limba engleză*	*Semnificaţie*
A	AX	AH, AL	Accumulator	acumulator
B	BX	BH, BL	Base	bază
C	CX	CH, CL	Counter	numărător
D	DX	DH, DL	Data	date

Tabelul 6.1: Denumirile registrelor generale de date.

Registrul	*Operaţii*
AX	Înmulţire cuvînt, împărţire cuvînt, I/O cuvînt
AL	Înmulţire byte, împărţire byte, I/O byte, conversie aritmetică zecimală
AH	Înmulţire byte, împărţire byte
BX	Conversie
CX	Operaţii cu şiruri; cicluri
CL	Deplasare şi rotire variabilă
DX	Înmulţire cuvînt, împărţire cuvînt, I/O indirectă

Tabelul 6.2: Utilizarea implicită a registrelor generale de date.

Registrele de date pot fi utilizate pentru operaţii aritmetice şi logice, dar există instrucţiuni care presupun implicit utilizarea anumitor registre (tabelul 6.2).

Registrele indicator (pointer) şi index au funcţii dedicate. Cele două registre pointer se pot accesa numai ca registre de 16 biţi. Aceste registre sînt folosite pentru a păstra deplasamentul (offset) relativ la registrele de segment, în operaţiile de acces la memorie. *Registrul indicator de stivă* (Stack Pointer = SP) asigură accesul la segmentul de memorie definit ca stivă de SS şi indică deplasamentul ultimei locaţii de memorie care a fost implicată într-o operaţie cu stiva. După o asemenea operaţie, registrul SP este automat modificat (nefiind necesară intervenţia programatorului) indicînd în permanenţă vîrful stivei relativ la baza segmentului de stivă. *Registrul indicator de bază* (Base Pointer = BP) conţine un deplasament faţă de originea stivei şi se foloseşte pentru a accesa date din cadrul stivei.

Registrele index conţin deplasamentul faţă de originea segmentului de date definit de DS. *Registru index al sursei* (Source Index = SI) şi *registrul index al destinaţiei* (Destination Index = DI) sînt folosite implicit pentru calculul adresei operandului sursă, respectiv destinaţie, în instrucţiunile care apelează la adresarea indexată.

Spre deosebire de registrele de uz general, registrele pointer şi index, se accesează numai la nivel de cuvînt, fiind imposibil accesul la nivel de octet. Aceste registre pot fi implicate în operaţii aritmetice şi logice.

Registrele de segment oferă suport pentru implementarea unei memorii cu un spaţiu de adresare de 1 MB. Spaţiul de adresare este împărţit în segmente de cîte 64 KB, din care, la un moment dat, numai 4 segmente pot fi active. Segmentele active sînt definite prin registrele de segment:

- CS (*Code segment*) segmentul de cod/program,

- DS (*Data segment*) segmentul de date,
- SS (*Stack segment*) segmentul de stivă,
- ES (*Extra segment*) segmentul de date suplimentar.

Fiecare din aceste registre conţine adresa de început a segmentului de 64 KB pe care îl defineşte, adresă care poate fi modificată sub acţiunea programului rulat. Încărcarea registrului de cod CS este echivalentă cu transferul controlului programului în alt segment de memorie. Pentru cazuri excepţionale, în care este necesară referirea la un operand aflat în afara segmentului implicit, se poate include în instrucţiune un prefix care desemnează explicit segmentul în care se găseşte operandul.

Registrele de stare şi control constau din registrul indicator de instrucţiuni şi registrul de stare (flag-uri).

Indicatorul de instrucţiuni (Instruction Pointer = IP) este un registru de 16 biţi care identifică locaţia următoarei instrucţiuni ce va fi executată, în segmentul de cod curent. IP este similar unui registru contor program (Program Counter = PC). Spre deosebire de un registru PC care conţine adresa fizică a următoarei instrucţiuni, IP conţine deplasamentul de 16 biţi al următoarei instrucţiuni. Pentru determinarea adresei fizice, deplasamentul trebuie combinat cu conţinutul unui registrului segment de cod (CS), pentru a genera adresa fizică a instrucţiunii. Fizic, acest registru se află în BIU. La fiecare aducere a unei instrucţiuni din memorie, BIU actualizează valoarea registrului IP pentru a indica următoarea instrucţiune.

Registrul de stare şi indicatori este un registru de 16 biţi aflat în EU. Din cei 16 biţi, numai 9 biţi au o semnificaţie: 3 biţi de control şi 6 biţi de stare.

Structura registrului de stare şi indicatori este prezentată în tabelul 6.3.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	OF	DF	IF	TF	SF	ZF	*	AF	*	PF	*	CF

Tabelul 6.3: Structura registrului de stare şi indicatori.

Indicatorii de stare sînt modificaţi ca urmare a execuţiei unor anumite instrucţiuni (aritmetice, logice sau special destinate), iar conţinutul lor nu poate fi citit direct, ci numai testat indirect, prin transferul controlului programului la ramuri diferite, în funcţie de valoarea indicatorului testat.

Indicatorii de stare sînt:

Indicator de transport (Carry Flag = CF), care este setat după o operaţie aritmetică în care a apărut transport sau împrumut; în rest valoarea acestui indicator este "0";

Indicator de paritate (Parity Flag = PF), care este setat dacă rezultatul unei operaţii aritmetice conţine un număr par de biţi de valoare "1"; dacă numărul este impar, atunci PF=0;

Indicator de transport auxiliar (Auxiliary Carry Flag = AF), care este setat dacă după o operaţie aritmetică pe 16 biţi a apărut transport/împrumut în/din tetrada mai semnificativă a octetului mai puţin semnificativ dintr-un cuvînt (16 biţi);

Indicator de zero (Zero Flag = ZF), care este setat dacă în urma unei operaţii logice sau aritmetice toţi biţii rezultatului sînt egali cu zero;

Indicator de semn (Sign Flag = SF), care în urma unei operaţii logice sau aritmetice ia valoarea celui mai semnificativ bit (MSB); astfel, dacă rezultatul este negativ, atunci SF = MSB = 1. SF = 0 pentru restul situaţiilor;

Indicator de depăşire (Overflow Flag = OF), care este setat dacă, în cazul a doi operanzi de acelaşi semn, semnul rezultatului este diferit de cel al operanzilor, ceea ce este echivalent cu faptul că rezultatul este în afara domeniului de reprezentare.

Exemplul 1:

În urma operaţiei de adunare pe octet a numerelor fără semn $199|_{10} = C7H$ şi $90|_{10} = 5AH$, rezultatul operaţiei este $199|_{10} + 90|_{10} = C7H + 5AH = 21H = 33|_{10}$.

```
  1100 0111 +
  0101 1010
-----------
1 0010 0001
```

CF=1 (pentru că a apărut transport), PF=1 (pentru că rezultatul 21H = 0010 0001 are un număr par de biţi egali cu unu), AF=1 (pentru că a apărut un transport în tetrada mai semnificativă), ZF=0 (pentru că rezultatul este diferit de zero), SF = MSB = 0, OF = 0. Numerele fiind considerate fără semn, SF şi OF nu au nici o semnificaţie în acest caz.

Exemplul 2:

Considerînd numerele cu semn $-57|_{10} = C7H$ şi $+90|_{10} = 5AH$, rezultatul adunării şi valoarea biţilor de stare sînt aceleaşi cu deosebirea că CF şi AF nu au nici o semnificaţie, iar $SF = 0$ indică faptul că rezultatul este corect, deci nu s-a produs depăşire.

Indicatorii de control sînt:

Indicatorul de întrerupere (Interrupt Flag = IF), care este utilizat pentru determinarea modului în care microprocesorul 8086 reacţionează la cererile de întrerupere mascabilă aplicate pe pinul INT (IF = 1 - cererile de întrerupere sînt validate). Starea acestui indicator poate fi setată/resetată prin program;

Indicatorul de direcţie (Direction Flag = DF), care este utilizat în operaţiile cu şiruri de octeţi cînd este necesar ca şirul să fie definit prin adresa de bază şi prin sensul descrescător (DF=0) sau crescător (DF=1) al adreselor care definesc octeţii din şir faţă de adresa de bază;

Indicatorul de mod "pas cu pas" (Trap Flag = TF), care permite implementarea unui mod de funcţionare util în depanarea programelor. Astfel, dacă TF = 1, după execuţia fiecărei instrucţiuni se iniţiază o întrerupere care determină transferul controlului într-o zonă de memorie definită de utilizator, unde, sub acţiunea unui program adecvat, se poate face analiza stării interne a microprocesorului.

6.1.4 Structura memoriei

Spaţiul de adresare al microprocesorului 8086 este de 1 MB. Datele aflate la o adresă de memorie au 8 biţi. Un cuvînt de 16 biţi poate fi stocat în memorie la două adrese succesive. Ordinea de

stocare a celor doi baiţi aparţinînd unui cuvînt este cu cel mai semnificativ la adresa superioară ("little endian"). Spaţiul de adresare de 1 MB al microprocesorului 8086 necesită 20 de biţi de adresă. Pentru a rezolva accesibilitatea resurselor de 1 MB utilizînd cuvinte de cîte 16 biţi s-a adoptat segmentarea memoriei în pachete a cîte 2^{16} B = 64 KB. Memoria poate fi interpretată ca un număr oarecare de segmente avînd fiecare maximum 64 KB. Adresa de început a unui segment este divizibilă cu 16 (ultimii patru biţi sînt 0). Un program poate avea acces imediat numai în patru segmente:

- segmentul de cod/program,
- segmentul de date,
- segmentul de stivă,
- segmentul de date suplimentar.

Selecţia acestor patru segmente a fost motivul pentru care unitatea *BIU* a lui 8086 a fost prevăzută cu registre segment de 16 biţi.

Pentru generarea unei adrese fizice în spaţiul de 1 MB, conţinutul unui registru de segment este deplasat la stînga cu 4 poziţii şi adunat cu conţinutul unuia din registrele pointer sau index. Rămîne la latitudinea programatorului segmentarea memoriei disponibile apelînd la segmente disjuncte, parţial disjuncte sau suprapuse, neexistînd nici o restricţie în acest sens. Determinarea adresei fizice din adresa logică este realizată de *BIU* aşa ca în figura 6.2.

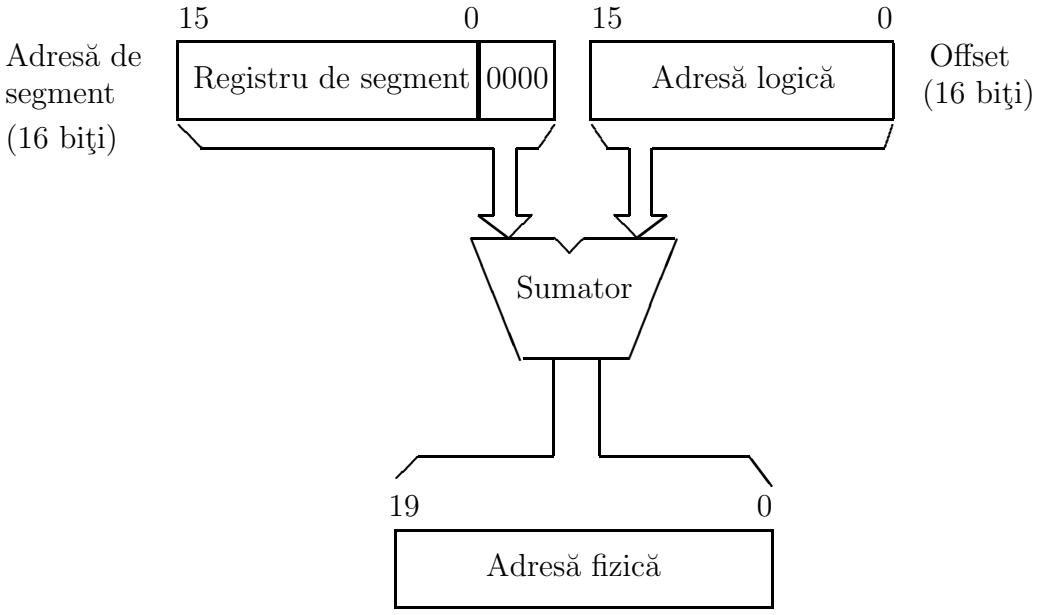

Figura 6.2: Determinarea adresei fizice.

BIU obţine adresele logice şi adresele de segment din diferite surse, în funcţie de tipul referinţei la memorie. Tabelul 6.4 prezintă sursele implicite pentru diferite tipuri de acces la memorie.

Segmentele de cod (CS) ale memoriei conţin instrucţiuni ale unuia sau mai multor programe. Conţinutul lui CS identifică locaţia de început a unei instrucţiuni în cadrul segmentului de cod activ. Pentru a face acces la locaţia unde se memorează instrucţiunea în segmentul de cod

Tipul referinţei la memorie	*Segment implicit*	*Segment alternativ*	*Deplasament (Offset)*
Fetch instrucţiune	CS	-	IP
Operaţie cu stiva	SS	-	SP
Accesare de variabilă	DS	CS, ES, SS	Adresă efectivă
Sursă de operaţie cu şiruri	DS	CS, ES, SS	SI
Destinaţie de operaţie cu şiruri	ES	-	DI
Registrul BP utilizat ca registru de bază	SS	CS, DS, ES	Adresă efectivă
Registrul BX utilizat ca registru de bază	DS	CS, ES, SS	Adresă efectivă

Tabelul 6.4: Sursele implicite pentru diferite tipuri de acces la memorie.

activ, 8086 trebuie să-şi poziţioneze toţi cei 20 biţi de adresă fizică. Pentru a realiza acest lucru, microprocesorul 8086 combină conţinutul registrului pointer de instrucţiuni (IP) cu valoarea din CS, rezultînd o adresă fizică ce va fi furnizată pe ieşirile de adresă ale microprocesorului. Segmentul de cod activ poate fi schimbat prin execuţia unei instrucţiuni care încarcă o nouă valoare în registrul CS. Din acest motiv, se poate utiliza pentru memorarea codului oricare din cele 16 segmente independente de memorie a cîte 64 KB.

Conţinutul registrului segment de date (DS) identifică locaţia de început a segmentului de date curent în memorie. Acesta este cel de-al doilea segment activ de 64 KB, care se constituie într-un spaţiu de memorie tip citire/scriere în care datele pot fi stocate. Majoritatea operanzilor instrucţiunilor sînt preluaţi din acest segment. Valorile din registrele index sursă (SI) sau destinaţie (DI) sînt combinate cu valoarea din (DS) pentru a forma o adresă fizică pe 20 de biţi (adresa operandului sursă sau destinaţie în segmentul de date).

Registrul segment stivă (SS) conţine adresa logică ce identifică locaţia de început a stivei curente în memorie. Este un segment de 64 KB în care sînt depuse valorile pointerului de instrucţiuni (IP), indicatorilor de stare şi ale altor registre (printr-o instrucţiune PUSH) la orice întrerupere hardware, întrerupere software sau execuţia unui apel de subrutină. După ce s-a încheiat execuţia subrutinei, starea originală a sistemului este restaurată prin executia instrucţiunii POP sau prin execuţia instrucţiunii RETURN. Următoarea locaţie în care va fi depus un cuvînt sau din care va fi preluat un cuvînt se obţine prin combinarea valorii curente din SS cu pointerul de stivă (SP). Stiva creşte spre valori de adrese descrescătoare. Cuvintele din stivă au 16 biţi (2 bytes).

Ultimul registru segment identifică cel de-al patrulea segment activ de 64 KB în spaţiul de adresare a memoriei. Acest spaţiu este denumit segment suplimentar (ES). Acest segment este folosit uzual pentru memorarea datelor. Instrucţiunile cu şiruri utilizează valoarea din ES cu conţinutul din DI drept un deplasament pentru a identifica adresa destinaţie.

6.1.5 Organizarea porturilor I/O

Spaţiul de adresare al porturilor I/O este disjunct faţă de spaţiul de adresare al memoriei. Adresarea porturilor se face cu instrucţiuni specifice (IN sau OUT).

Spaţiul de adresare al porturilor poate fi văzut ca fiind de dimensiune $64K \times 8$ biţi sau $32K \times 16$ biţi. Adresarea primelor 256 de porturi se poate face direct (adresa portului inclusă în codul

instrucţiunii). Toate porturile pot fi adresate indirect, adresa de 16 biţi fiind plasată în registrul implicit DX.

Instrucţiunile IN şi OUT transferă date între acumulator (AL pentru transferuri pe 8 biţi şi AX pentru transferuri pe 16 biţi) şi portul localizat în spaţiul I/O.

6.1.6 Modurile de adresare

Operanzii instrucţiunilor microprocesorului 8086 pot fi conţinuţi în registre, într-un cîmp al instrucţiunii, în memorie sau la porturile I/O. Adresele la care se găsesc operanzii în memorie sau la porturile I/O se pot determina în mai multe feluri.

Operanzii imediaţi sînt constante de 8 sau 16 biţi incluse în codul instrucţiunii. Accesarea acestora se face în faza de aducere a instrucţiunii în memorie (fetch) şi nu necesită un ciclu suplimentar de citire a memoriei.

Operanzi imediaţi (operandul este în corpul instrucţiunii)

```
CR   EQU  13
MOV  AL, CR         ; iniţializează registrul AL cu 13
MOV  AX, 0FFFFH     ; iniţializează registrul AX cu o dată imediată
```

Instrucţiunile cu *operanzi din registre* sînt executate mai rapid deoarece aceste operaţii nu fac acces la resurse din afara EU.

Operanzi în registre (corpul instrucţiunii conţine codul unui registru)

```
MOV  AL, BL         ; copiază conţinutul registrului BL în registrul AL
MOV  AX, BX         ; copiază conţinutul registrului BX în registrul AX
```

Spre deosebire de operanzii imediaţi şi de cei din registre, care sînt direct accesaţi de către EU, *operanzii din memorie* trebuie transferaţi între CPU şi memorie prin intermediul BIU şi al bus-ului extern. EU calculează *adresa efectivă* (Effective Address = EA) a operandului şi o transmite către BIU. EA este de tipul întreg reprezentat pe 16 biţi şi reprezintă offsetul operandului în segmentul în care se face adresarea. Pentru calcularea adresei efective, EU foloseşte informaţia din cel de-al doilea byte al instrucţiunii. Codul acestui byte este dedus de către compilator sau asamblor pe baza instrucţiunii scrise de către programator. În limbaj de asamblare se poate accesa memoria în toate modurile de adresare.

Adresare directă EA = $deplasament_{8|16}$

```
LOC1 DB 13          ; defineşte o variabilă de tip byte
                    ; şi o iniţializează cu 13 (baza 10)
MOV AL, LOC1        ; transferă în AL data de 8 biţi aflată
                    ; în memorie la adresa LOC1
```

Adresare indirectă la registru EA = $[BX|BP|SI|DI]$

```
MOV AX, [BX]        ; transferă în AX data de 16 biţi aflată
                    ; în memorie la adresa conţinută în registrul BX
```

Adresare la bază EA $= [BX|BP] + deplasament_{8|16}$

```
MOV AX, [BX+2]     ; transferă în AX data de 16 biţi aflată în
                   ; memorie la adresa obţinută prin însumarea
                   ; conţinutului registrului BX cu 2
```

Adresare indexată EA $= [SI|DI] + deplasament_{8|16}$

```
MOV AX, [SI+2]     ; transferă în AX data de 16 biţi aflată
                   ; în memorie la adresa obţinută prin însumarea
                   ; conţinutului registrului SI cu 2
```

Adresare la bază indexată EA $= [BX|BP] + [SI|DI] + deplasament_{8|16}$

```
MOV AX, [BX+SI+2] ; transferă în AX data de 16 biţi aflată
                   ; în memorie la adresa obţinută prin însumarea
                   ; conţinutului registrului BX cu conţinutul
                   ; registrului SI şi cu 2
```

Adresarea şirurilor EAsursă $= [SI]$, EAdestinaţie $= [DI]$

```
s1  DB 'şir1'      ; defineşte două variabile de tip
s2  DB 'şir2'      ; byte şi le iniţializează cu cîte patru caractere
MOV SI, offset s1 ; pregăteşte registrele implicite ale
MOV DI, offset s2 ; instrucţiunii care operează asupra
MOV CX, 4          ; şirurilor de caractere

REP MOVSB          ; instrucţiune care mută un şir de
                   ; caractere adresat implicit, de la DS:SI la ES:DI
```

Adresarea porturilor I/O Adresă $\text{Port}_8 = data_8$, Adresă $\text{Port}_{16} = [DX]$

```
IN AL, 0EAH        ; citeşte de la portul adresat direct un cuvînt de
                   ; 8 biţi şi îl transmite în registrul AL

MOV DX, 0ABCDH     ; pregăteşte în registrul DX adresa
                   ; portului reprezentată pe 16 biţi
IN AX, DX          ; citeşte de la portul a cărui adresă este
                   ; conţinută în registrul DX un cuvînt de 16 biţi
                   ; şi îl transmite în registrul AX
```

6.2 Conexiunile externe ale microprocesorului

Firma Intel a redus numărul de conexiuni externe, prezentînd microprocesorul 8086 într-o capsulă cu 40 de pini.

6.2.1 Modurile de lucru ale microprocesorului

Soluţia de încapsulare aleasă a impus multiplexarea în timp a funcţiilor mai multor pini (figura 6.3) cu implicaţii directe în realizarea circuitelor (schemelor) tipice cu acest microprocesor. Întrevăzînd faptul că 8086 va fi înglobat atît în configuraţii simple cît şi foarte complexe, firma Intel a prevăzut două moduri de lucru:

- modul *minim* şi
- modul *maxim*.

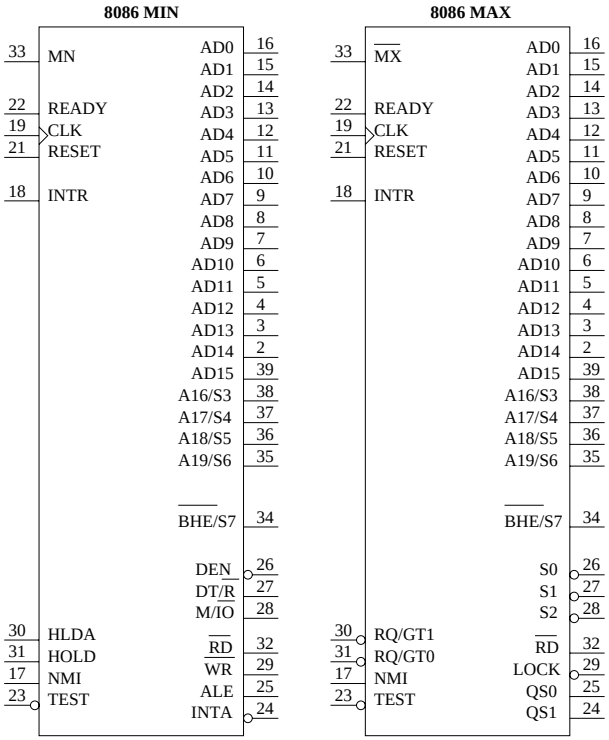

Figura 6.3: Configuraţia pinilor microprocesorului 8086.

Stabilirea modului de lucru se face prin impunerea unei stări logice pinului MN/MX, astfel:

- MN/MX = 1 pentru modul minim, care este potrivit sistemelor simple, cu un singur microprocesor;
- MN/MX = 0 pentru modul maxim, care se pretează la configuraţii multiprocesor.

În acord cu modul de lucru ales, destinaţia unor pini este diferită, ceea ce impune împărţirea pinilor în două categorii: *pini cu funcţii comune ambelor moduri de lucru* şi *pini cu funcţii specifice modurilor de lucru minim şi maxim*. Tabelul 6.5 prezintă semnificaţia pinilor microprocesorului 8086.

6.2.2 Modul minim

În modul minim, microprocesorul 8086 generează şi acceptă toate semnalele necesare pentru a realiza un sistem simplu, avînd totuşi facilităţi importante (lucru cu memoria sau cu porturile, în mod DMA sau prin întreruperi).

Semnale comune		
Nume	*Funcţie*	*Tip*
AD15-AD0	Bus adrese/date	Bidir. 3-Stări
A19/S6-A16/S3	Adrese/Stări	Ieşire, 3-Stări
NBHE/S7	Validare bus superior/stare	Ieşire, 3-Stări
MN/NMX	Control mod minim/maxim	Intrare
NRD	Control citire	Intrare, 3-Stări
NTEST	Wait pe control test	Intrare
READY	Control stare wait	Intrare
RESET	Reset sistem	Intrare
NMI	Cerere întrerupere nemascabilă	Intrare
INTR	Cerere întrerupere mascabilă	Intrare
CLK	Tact sistem	Intrare
VCC	+5V	Intrare
GND	Masa	Intrare
Semnalele modului minim (MN/NMX = VCC)		
Nume	*Funcţie*	*Tip*
HOLD	Cerere hold	Intrare
HLDA	Acceptare hold	Ieşire
NWR	Control scriere	Ieşire, 3-Stări
M/NIO	Control memorie/IO	Ieşire, 3-Stări
DT/NR	Emisie/recepţie date	Ieşire, 3-Stări
NDEN	Validare date	Ieşire, 3-Stări
ALE	Validare preluare adrese	Ieşire
NINTA	Recunoaştere întrerupere	Ieşire
Semnalele modului maxim (MN/NMX = GND)		
Nume	*Funcţie*	*Tip*
NRQ/NGT1,0	Control acces bus Cerere/cedare	Bidirecţional
NLOCK	Control lock prioritate bus	Ieşire, 3-Stări
NS2-S0	Stare ciclu de bus	Ieşire, 3-Stări
NQS1, NQS0	Starea cozii de instrucţiuni	Ieşire

Tabelul 6.5: Semnificaţia pinilor microprocesorului 8086.

S0	*S1*	*S2*	*Tip ciclu de bus*
0	0	0	Confirmare întrerupere (INTA)
0	0	1	Citire port I/O
0	1	0	Scriere port I/O
0	1	1	Stare de "halt"
1	0	0	Fetch instrucţiune
1	0	1	Citire memorie
1	1	0	Scriere memorie
1	1	1	Pasiv, nici un ciclu de bus

Tabelul 6.6: Tipul ciclului de bus, în funcţie de biţii de stare S0, S1 şi S2.

Magistrala de adrese şi date - AD0..AD15, A16..A19
Magistrala de adrese îndeplineşte, prin multiplexare în timp, două funcţii.

Magistrală de adrese de 20 biţi. În ciclurile I/O, A16=A17=A18=A19='0' ceea ce limitează la $2^{16} = 65536$ adresele disponibile pentru dispozitivele periferice.

Magistrală de date de 16 biţi (D15-D0). În acest caz, magistrala de date D0..D15 este bidirecţională.

Procesorul 8086 are posibilitatea de a trece liniile AD0..AD15, A16..A19 în starea de înaltă impedanţă (3-state).

Semnale de stare
Semnalele S0, S1 şi S2 prezintă informaţia referitoare la tipul ciclului de magistrală curent. Starea prezentată de aceşti biţi este validă la începutul fiecărui ciclu de bus. Tabelul 6.6 prezintă modul de interpretare a informaţiei furnizate de aceste semnale.

Prin multiplexare, liniile A16..A19 îndeplinesc şi o funcţie de stare. Prin prisma acestui dublu rol, denumirea corectă a magistralei este A16/S3..A19/S6. Informaţia de stare S3..S6 este prezentă simultan cu situaţia cînd informaţia de pe liniile de date este stabilă. Semnalele de pe magistrală au următoarea semnificaţie:

- S3 şi S4 formează un cod ce reprezintă registrul segment utilizat pentru generarea adresei din ciclul curent de magistrală (tabelul 6.7);
- S5 reprezintă valoarea flagului IF (Interrupt Flag) de autorizare a întreruperilor mascabile;
- S6 este neutilizat, fiind întotdeauna egal cu "0".

Semnale de control

Semnalele de control permit interconectarea cu memoria şi dispozitivele periferice. Semnificaţia biţilor de control este dependentă de tipul informaţiei existente pe magistrala D0..D15:

- adresă validă (ieşire);
- date de intrare, într-un ciclul de citire;

S4	*S3*	*Registru segment*
0	0	ES (segment de date auxiliar)
0	1	SS (segment de stivă)
1	0	CS sau nimic (I/O sau vector de întrerupere)
1	1	DS (segment de date)

Tabelul 6.7: Informaţia codificată cu biţii de stare S3 şi S4.

- date de ieşire, într-un ciclu de scriere.

ALE (Address Latch Enable) este un semnal activ în "1", care indică faptul că magistrala AD0..AD15 conţine o informaţie de adresă validă referitoare la ciclul aflat în execuţie. Întrucît această adresă este validă numai pe perioada cît ALE="1", ea trebuie memorată pe frontul căzător al semnalului ALE (în circuite "latch") pentru a fi folosită ulterior.

BHE (Bank High Enable) este un semnal activ în "0", folosit pentru a activa modulul de memorie legat de magistrala de date, în jumătatea mai semnificativă (D8..D15). De asemenea, BHE poate fi folosit ca semnal de stare S7.

M/IO (Memory/Input-Output) este un semnal care se formează pe baza tipului ciclului aflat în execuţie şi anume:

- M/IO="1" pentru acces la memorie;
- M/IO="0" pentru acces la port.

DT/R (Data Transmit/Receiver) indică dacă ciclul curent este unul de scriere, cînd datele sînt transmise de către microprocesor (DT/R="1"), sau unul de citire, cînd datele sînt citite de către microprocesor (DT/R="0"). Citirea şi scrierea datelor se referă atît la accesul la memorie cît şi la porturi. Semnalul DT/R poate fi folosit pentru schimbarea sensului de transfer al unui circuit bidirecţional de amplificare a magistralei de date.

RD (Read) este generat într-un ciclu de citire, fiind folosit pentru selectarea dispozitivelor logice, care oferă informaţia citită de către microprocesor.

WR (Write) este generat într-un ciclu de scriere semnalizînd că pe AD0..AD15 se află o dată validă, care trebuie înscrisă în memorie sau port. Semnalele RD şi WR pot fi prelungite nelimitat, oferind astfel posibilitatea ca memorii sau porturi mai lente să poată fi folosite cu microprocesorul 8086.

DEN (Data Enable) este generat atît în ciclurile de citire cît şi în cele de scriere, cînd este necesară marcarea momentului în care magistrala comună îndeplineşte funcţia de date (D0..D15). Semnalul DEN poate fi folosit pentru selecţia circuitelor 3-state de amplificare a magistralei de date.

READY este un semnal primit de microprocesor, la activarea căruia microprocesorul îşi prelungeşte ciclurile de magistrală şi implicit semnalele de comandă pentru resursele hardware lente.

Semnale de întrerupere

Întreruperea, în sens general, este devierea unui program principal prin comanda unui semnal exterior (întrerupere hardware), ca urmare a unei stări interne (întrerupere internă) sau ca

Registru	*Conţinut*
Registru de stare şi indicatori	0000H
IP	0000H
CS	FFFFH
DS	0000H
SS	0000H
ES	0000H
Coadă de instrucţiuni	Goală

Tabelul 6.8: Starea registrelor după resetare.

urmare a execuţiei unor instrucţiuni specifice (întrerupere software). Devierea este urmată de o tratare specifică şi revenirea în starea iniţială, în care a fost făcută întreruperea.

INTR (Interrupt) este un semnal prin care un dispozitiv periferic cere microprocesorului să îşi întrerupă temporar activitatea pentru a-l deservi. Acest semnal este testat de către microprocesor în ultima perioadă de ceas a fiecărui ciclu de aducere a instrucţiunii (fetch). Dacă semnalul are valoarea "1" şi întreruperile sînt validate, se iniţiază un ciclu de recunoaştere a întreruperii, cînd INTA devine activ ("0").

TEST poate determina, în anumite condiţii, suspendarea unui program. După execuţia instrucţiunii WAIT, microprocesorul 8086 testează starea liniei TEST. Dacă aceasta are valoarea "1" se trece magistrala în inactivitate şi se aşteaptă revenirea liniei TEST la valoarea "0". Programul este reluat din punctul de întrerupere. Prin utilizarea semnalului TEST se poate implementa un mod simplu de a sincroniza activitatea microprocesorului cu evenimente externe.

NMI (Non Maskable Interrupt) poate determina, pe frontul crescător, declanşarea unei întreruperi care nu poate fi invalidată. Activarea NMI determină necondiţionat execuţia unei rutine speciale de tratare a întreruperii. NMI poate fi folosit pentru iniţierea unor comenzi în cazul scăderii tensiunii de alimentare sau pentru tratarea unor erori de paritate la memorie (ca la calculatorul IBM PC).

RESET este inclus tot în categoria semnalelor de întrerupere. Activarea sa (pe valoare "0"), determină aducerea registrelor şi a flag-urilor interne într-o stare iniţială, urmînd ca la inactivarea acestuia (revenirea la valoarea "1") să fie reluată activitatea microprocesorului din această stare. După resetarea microprocesorului registrele sînt iniţializate în stările prezentate în tabelul 6.8.

Semnale de interfaţă DMA

HOLD poate fi adus în "1" de către un dispozitiv care intenţionează să preia controlul asupra magistralelor de adrese, date şi comenzi. La activarea semnalului, după terminarea ciclului curent, microprocesorul 8086 trece în starea de înaltă impedanţă semnalele: AD0 .. AD15, A16/S3 .. A19/S6, BHE, DEN, DTR, MI/0, RD, WR şi INTR. Totodată, această stare (*hold state*) este semnalizată în exterior prin HLDA="1" (Hold Acknowledge), perioadă în care *EU* utilizează informaţia din QFIFO, pînă cînd aceasta rămîne vidă.

6.2.3 Modul maxim

Dacă microprocesorul 8086 este pus să lucreze în modul maxim, (prin fixarea pinului MN/MX la "0") el generează semnale pentru implementarea structurilor multiprocesor/coprocesor. Prin acest termen se definesc acele structuri hardware care presupun existenţa mai multor microprocesoare în cadrul aceleiaşi configuraţii sistem, în condiţiile în care fiecare microprocesor îşi execută, în cea mai mare parte a timpului, propriul său program. De obicei, asemenea structuri conţin anumite resurse globale, comune tuturor microprocesoarelor. Fiecare microprocesor poate avea şi resurse locale (private).

În sistemele cu coprocesor există un al doilea microprocesor în sistem. În acest caz, cele două microprocesoare nu pot face acces la bus în acelaşi timp. Unul dintre ele trebuie să transmită celuilalt controlul bus-ului sistem şi să-şi suspende pentru un timp operaţiile. În sisteme cu microprocesor 8086 cuplat în modul maxim se întîlnesc facilităţi suplimentare, în ceea ce priveşte implementarea alocării resurselor globale şi transmiterea controlului bus-ului altor microprocesoare sau coprocesoare.

6.3 Instrucţiunile microprocesorului

Instrucţiunile microprocesorului 8086 pot fi împărţite în următoarele categorii:

- Instrucţiuni pentru transfer de date;
- Instrucţiuni aritmetice;
- Instrucţiuni pentru manipulare de biţi;
- Instrucţiuni pentru manipulare de şiruri;
- Instrucţiuni pentru transferul controlului programului;
- Instrucţiuni pentru controlul microprocesorului.

În continuare se face o prezentare sumară a categoriilor de instrucţiuni. La fiecare categorie se va prezenta un tabel cu mnemonicile asociate instrucţiunilor şi cu definiţia instrucţiunii, aşa cum este dată de producător (în limba engleză). Anexa **??** prezintă mai detaliat fiecare instrucţiune în parte.

6.3.1 Instrucţiuni pentru transfer de date

Instrucţiunile pentru transfer de date (tabelul 6.9) mută (copiază) date reprezentate pe 8 sau 16 biţi între memorie şi registre sau între registre şi porturi I/O. Acest grup include şi instrucţiunile de manipulare a stivei şi instrucţiunile de încărcare a registrelor de segment.

6.3.2 Instrucţiuni aritmetice

Instrucţiunile aritmetice (tabelul 6.10) se pot executa asupra patru tipuri de date binare:

Transferuri generale	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
Intrare/Ieşire	
IN	Input byte or word
OUT	Output byte or word
Adresare de obiecte	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
Transferuri de indicatori	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags from stack

Tabelul 6.9: Instrucţiuni pentru transfer de date.

- numere binare întregi, fără semn (pozitive);
- numere binare întregi, cu semn;
- numere în baza 10, fără semn, neîmpachetate;
- numere în baza 10, fără semn, împachetate.

Numerele binare pot fi reprezentate pe 8 sau 16 biţi. Totdeauna microprocesorul presupune că operanzii specificaţi conţin date valide. Pentru numere binare fără semn există instrucţiuni de adunare, scădere, înmulţire şi împărţire.

6.3.3 Instrucţiuni pentru manipulare de biţi

Instrucţiunile pentru manipulare de biţi (tabelul 6.11) cuprind instrucţiunile de prelucrare logică, instrucţiunile de deplasare şi instrucţiunile de rotire.

6.3.4 Instrucţiuni pentru manipulare de şiruri

Instrucţiunile elementare pentru manipulare de şiruri (tabelul 6.12) operează asupra unui singur element de şir (8 sau 16 biţi). Prin utilizarea prefixelor de repetare, se pot realiza operaţii asupra unui număr de maxim 128 K elemente.

Instrucţiunile de manipulare de şiruri folosesc implicit următoarele registre şi indicatori:

Adunare	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
Scădere	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
Înmulţire	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
Împărţire	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte or word
CWD	Convert word to doubleword

Tabelul 6.10: Instrucţiuni aritmetice.

SI - offset şir sursă;

DI - offset şir destinaţie;

DX - adresă de port;

CX - numărător de repetiţii;

AL/AX - destinaţie/sursă pentru LODS/STOS;

DF - '0' - autoincrement SI, DI sau '1' - autodecrement SI, DI;

ZF - indicator folosit pentru determinarea sfîrşitului de şir.

6.3.5 Instrucţiuni pentru transferul controlului programului

Locul de unde se execută programul este determinat de conţinutul registrelor CS şi IP. Abaterea de la prelucrarea secvenţială a instrucţiunilor se realizează prin execuţia unor instrucţiuni care modifică valorile stocate în aceste registre (tabelul 6.13).

Logice	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
Deplasare	
SHL/SAR	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
Rotire	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Tabelul 6.11: Instrucţiuni pentru manipulare de biţi.

6.3.6 Instrucţiuni pentru controlul microprocesorului

Instrucţiunile pentru controlul microprocesorului (tabelul 6.14) permit controlarea prin program a diferitelor funcţii ale CPU. Această categorie include instrucţiunile de manipulare a indicatorilor şi cele pentru sincronizare externă.

6.4 Întrebări

I. Justificaţi prezenţa diferitelor grupe de registre în EU sau BIU.

II. Prezentaţi suportul oferit de *microprocesorul 8086* (hardware) pentru realizarea unor programe relocatabile dinamic (programe care pot fi încărcate în memorie la adrese

REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
MOVS	Move byte or word string
MOVSB/MOVSW	Move byte or word string
CMPS	Compare byte or word string
INS	Move byte or word string from I/O
OUTS	Move byte or word string to I/O
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string

Tabelul 6.12: Instrucţiuni pentru manipulare de şiruri.

Transferuri necondiţionate	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
Transferuri condiţionate	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
Controlul iteraţiilor	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX=0
Întreruperi	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Tabelul 6.13: Instrucţiuni pentru transferul controlului programului.

Operaţii cu indicatori	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt flag
CLI	Clear interrupt flag
Sincronizări externe	
HLT	Halt until interrupt or reset
WAIT	Wait until TEST pin active
ESC	Escape to external processor
LOCK	Lock bus during next instruction
Operaţie nulă	
NOP	No operation

Tabelul 6.14: Instrucţiuni pentru controlul microprocesorului.

aleatoare). Care sînt condiţiile pe care trebuie să le îndeplinească un *program* (software) pentru a fi dinamic relocatabil?

III. Se poate ca o procedură să implementeze o stivă diferită de cea a programului care o apelează? Daţi explicaţii referitoare la suportul *microprocesorului 8086* (hardware) şi la particularităţile *programelor* (software).

IV. Care este adresa fizică a primei instrucţiuni executate de microprocesorul 8086, după resetare? Decodificaţi datele existente în memorie la acea adresă şi deduceţi ce instrucţiuni execută un calculator PC imediat după resetare. Folosiţi utilitarul *debug*. Dacă se tastează '?' la prompterul '-' se obţine lista comenzilor, aşa cum este prezentată în continuare:

```
assemble     A [address]
compare      C range address
dump         D [range]
enter        E address [list]
fill         F range list
go           G [=address] [addresses]
hex          H value1 value2
input        I port
load         L [address] [drive] [firstsector] [number]
move         M range address
name         N [pathname] [arglist]
output       O port byte
proceed      P [=address] [number]
quit         Q
register     R [register]
search       S range list
trace        T [=address] [value]
unassemble   U [range]
```

```
write        W [address] [drive] [firstsector] [number]
allocate expanded memory        XA [#pages]
deallocate expanded memory      XD [handle]
map expanded memory pages       XM [Lpage] [Ppage] [handle]
display expanded memory status  XS
```

V. Realizaţi o schemă de calcul a *adresei efective* pentru fiecare mod de adresare al microprocesorului 8086. De exemplu, calculul adresei efective la adresarea bazată se face conform schemei din figura 6.4.

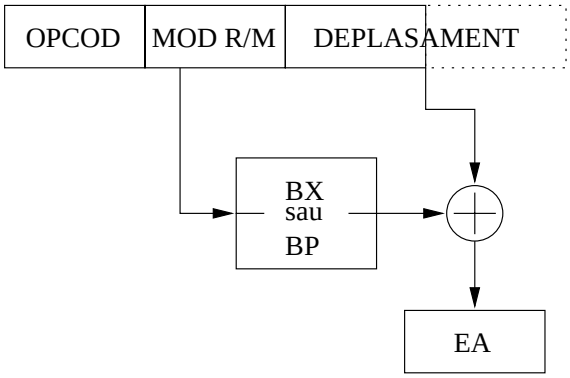

Figura 6.4: Calculul adresei efective la adresarea bazată.

VI. Faceţi o paralelă între modurile de adresare ale microprocesorului 8086 şi cele ale microprocesoarelor RISC.

VII. Referitor la setul de instrucţiuni al microprocesorului 8086 justificaţi atributele: *simplitate*, *ortogonalitate*, *completitudine*.

Lucrarea 7

Programarea în limbaj de asamblare 8086

Această lucrare prezintă etapele care trebuie urmate pentru realizarea unui program executabil pornind de la un cod sursă scris în limbaj de asamblare.

Există un sfat care zice că pentru a te face înţeles trebuie să vorbeşti fiecăruia pe limba lui. Dar care este "limba microprocesorului"? Microprocesorul codifică atît datele cît şi instrucţiunile în cifre binare $\{0, 1\}$. Deşi teoretic este posibil, astăzi nimeni nu mai concepe să "vorbească" (programeze) la un nivel atît de scăzut. Programarea într-un limbaj de nivel înalt (C, Pascal, Fortran) a devenit ceva natural. Limbajele de acest nivel au specificaţii foarte apropiate de modul de gîndire uman: decizii, iteraţii, prelucrări aritmetice şi logice cu scalari sau matrici, etc. Din păcate, cu cît limbajul este mai apropiat de limbajul uman, cu atît este mai depărtat de limbajul microprocesorului. Limbajele de nivel înalt permit gestionarea unor programe mari dar nu totdeauna permit gestionarea eficientă a resurselor hardware interne microprocesorului.

Limbajul de asamblare, spre deosebire de limbajul cifrelor binare, este un limbaj accesibil programatorului uman. Programul scris în limbaj de asamblare (*cod sursă*) nu este direct executabil de către microprocesor. Însă, fiecărei instrucţiuni în limbaj de asamblare îi corespunde o instrucţiune binară ce este "înţeleasă" (decodificată) şi executată de către microprocesor. Codul binar al operaţiei (*opcode*) este asociat cu un grup de litere sugestiv pentru efectul instrucţiunii respective (*mnemonică*).

Se poate face o analogie între limbajele vorbitorilor umani şi limbajele de asamblare ale microprocesoarelor. Limbajul de asamblare este propriu fiecărui tip de microprocesor în parte. Familii diferite de microprocesoare au seturi diferite de instrucţiuni. În cadrul unei familii, o generaţie mai nouă de microprocesoare preia setul de instrucţiuni al generaţiei vechi şi îl sporeşte, incluzînd instrucţiuni specifice gestionării noilor resurse hardware apărute. Dar, în general, un microprocesor de generaţie nouă poate rula şi aplicaţii scrise pentru microprocesoarele din generaţia anterioară. Există elemente comune ale seturilor de instrucţiuni ale diferitelor familii de microprocesoare.

7.1 Ciclul de dezvoltare al programelor scrise în limbaj de asamblare

Pentru a fi executat de către microprocesor, un program scris de un programator uman trebuie să parcurgă cîteva transformări. Reprezentarea grafică a acestor transformări este prezentată în figura 7.1.

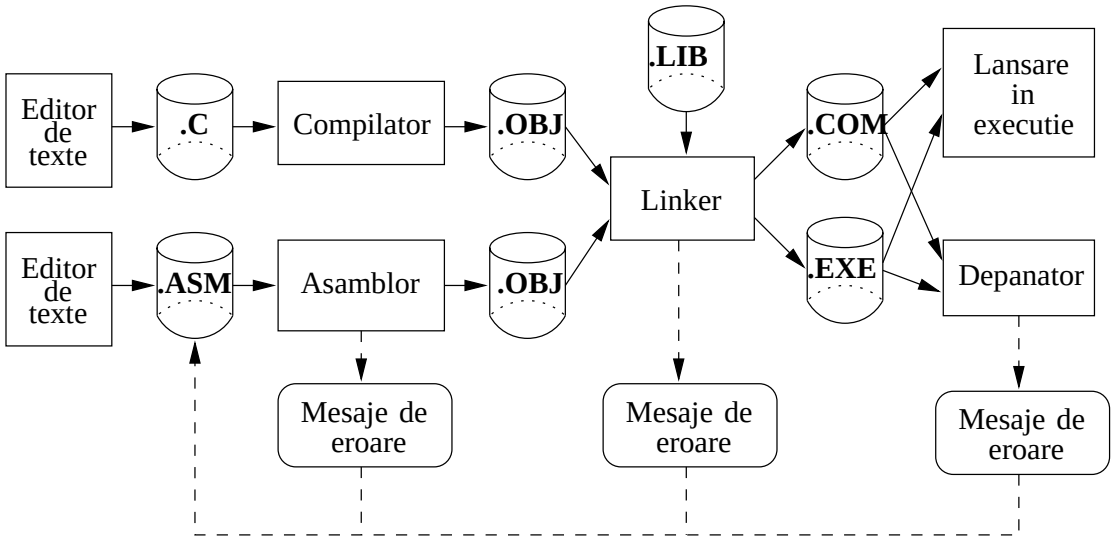

Figura 7.1: Reprezentarea grafică a ciclului de dezvoltare a programelor scrise în limbaj de asamblare

Pentru înţelegerea termenilor care apar în figura 7.1, se prezintă cîteva definiţii.

Limbaj de asamblare - limbaj de programare folosit de programatorul uman pentru a descrie un algoritm folosind setul de instrucţiuni propriu unui anumit microprocesor. Fiecare microprocesor are un limbaj de asamblare propriu, deşi se pot găsi multe elemente comune mai multor microprocesoare. Formatul instrucţiunilor în limbaj de asamblare 8086 este:

[<etichetă>:] <mnemonică> [<destinaţie>[, <sursă>]] [; comentariu]

Program (cod) sursă - program scris în limbaj de asamblare. Pe lîngă instrucţiunile în limbaj de asamblare care au corespondent în setul de instrucţiuni al microprocesorului, programul sursă mai poate conţine şi *directive de asamblare*. Directivele de asamblare nu generează cod executabil ci furnizează asamblorului indicaţii despre modul de transformare a codului sursă în cod executabil.

Asamblor - program (software) care translatează programul sursă scris în limbaj de asamblare într-un limbaj maşină posibil de executat de către microprocesor.

Asamblare - procesul de conversie de cod efectuat de asamblor.

Cod obiect - program aflat între limbajul de asamblare şi limbajul maşină. Codul obiect nu este direct executabil de către microprocesor ci trebuie să treacă prin procesul de editare a legăturilor.

Linker (link editor) - program (software) care realizează procesul de editare de legături. Linker-ul primeşte la intrare unul sau mai multe fişiere obiect (eventual şi fişiere ce conţin biblioteci de module) şi generează un fişier direct executabil.

Link-editare (editare de legături) - procesul de rezolvare a referinţelor externe modulului şi a referinţelor la adrese în cadrul unui modul obiect. Rezultatul link-editării este un program direct executabil de către microprocesor. Editarea de legături realizează trei acţiuni principale:

- Combină modulele separate de cod obiect într-un singur fişier executabil;
- Rezolvă referinţele la variabile externe;
- Opţional, produce un fişier ce conţine informaţia despre modul în care au fost legate fişierele obiect.

Limbaj maşină - limbaj binar interpretat de către microprocesor ca instrucţiuni executabile.

Program executabil - transpunerea unui algoritm în limbaj maşină.

7.2 Asamblarea

Asamblorul firmei *Borland* se numeşte *Turbo Assembler* ($TASM$). Asamblorul acceptă la intrare un *fişier sursă* care conţine instrucţiuni în limbaj de asamblare (specifice microprocesorului) şi un set de directive (specifice asamblorului). Prin directive de asamblare, programatorul poate transmite asamblorului indicaţii despre modul de asamblare a codului sursă. Acest fişier este un fişier ASCII, care are extensia implicită $.ASM$. Ca punct de plecare, se poate spune că un asamblor este un program care translatează un program sursă scris în limbaj de asamblare în cod maşină. În principiu, sarcina asamblorului este de a converti mnemonicile limbajului de asamblare în echivalentele numerice care reprezintă codul maşină.

Pentru a translata codul, asamblorul trebuie să parcurgă programul sursă cel puţin o dată. Fiecare citire a codului sursă se numeşte *trecere* (*pass*, în limba engleză). Cele mai multe asambloare necesită două treceri, deşi există asambloare cu o singură trecere sau cu mai mult de două treceri.

În prima trecere, asamblorul execută următoarele acţiuni:

- face analiza sintactică a codului şi determină offset-ul pentru fiecare linie (adresa la care se va asambla);
- face ipoteze asupra valorilor nedefinite;
- face verificările elementare de corectitudine a codului şi afişează mesaje de eroare;
- opţional, generează un fişier listing intermediar.

În a doua trecere, asamblorul execută următoarele acţiuni:

- încearcă reconcilierea valorilor presupuse în trecerile anterioare;
- generează *fişierul obiect*, care are extensia implicită $.OBJ$.

- opţional, generează *fişierul listing*, care are extensia implicită *.LST*. Fişierul listing poate fi considerat ca un raport al asamblorului. Acest fişier conţine atît specificaţile în limbaj de asamblare cît şi instrucţiunile în limbaj maşină rezultate după asamblare;

- opţional, generează *fişierul de referinţe încrucişate*, care are extensia implicită *.XRF*. Acest fişier conţine informaţii pe care alte programe (specifice asamblorului) le folosesc pentru a crea o listă de referinţe încrucişate ale simbolurilor utilizate în fişierul sursă. Pentru crearea concretă a acestui fişier se utilizează un alt program, care în cazul *TASM* se numeşte *TCREF*.

Turbo Assembler este un program care acceptă parametrii transmişi prin linia de comandă. Sintaxa liniei de comandă a asamblorului, aşa cum este ea specificată de firma producătoare este descrisă în continuare..

```
Turbo Assembler  Version 2.51  Copyright (c) 1988, 1991 Borland International

Syntax:TASM [options] source [,object] [,listing] [,xref]

/a,/s          Alphabetic or Source-code segment ordering
/c             Generate cross-reference in listing
/dSYM[=VAL]    Define symbol SYM = 0, or = value VAL
/e,/r          Emulated or Real floating-point instructions
/h,/?          Display this help screen
/iPATH         Search PATH for include files
/jCMD          Jam in an assembler directive CMD (eg. /jIDEAL)
/kh#           Hash table capacity # symbols
/l,/la         Generate listing: l=normal listing, la=expanded listing
/ml,/mx,/mu    Case sensitivity on symbols: ml=all, mx=globals, mu=none
/mv#           Set maximum valid length for symbols
/m#            Allow # multiple passes to resolve forward references
/n             Suppress symbol tables in listing
/o,/op         Generate overlay object code, Phar Lap-style 32-bit fixups
/p             Check for code segment overrides in protected mode
/q             Suppress OBJ records not needed for linking
/t             Suppress messages if successful assembly
/w0,/w1,/w2    Set warning level: w0=none, w1=w2=warnings on
/w-xxx,/w+xxx Disable (-) or enable (+) warning xxx
/x             Include false conditionals in listing
/z             Display source line with error message
/zi,/zd        Debug info: zi=full, zd=line numbers only
```

Sintaxa lansării în execuţie a utilitarului TCREF este următoarea:

```
Turbo CREF  Version 2.0  Copyright (c) 1988, 1991 Borland International

Syntax:  TCREF xrffiles, reffile [options]

@xxxx indicates use response file xxxx
Options:
```

```
  /c  = lower case significant in symbols
  /r  = full module level report
  /p# = page length (# in lines)
  /w# = page width  (# in characters)
```

7.3 Editarea de legături

Denumirile de *Link – Editor* şi de *Linker* sînt absolut echivalente. Linker-ul este un program care translatează un cod obiect relocatabil (produs de un asamblor sau un compilator) în cod maşină executabil. Linker-ul realizează trei funcţii principale:

- combină module obiect separate într-un singur fişier executabil;
- încearcă rezolvarea referinţelor la variabile externe;
- opţional, produce un fişier listing în care se prezintă modul de legare a fişierelor obiect.

Link editorul firmei *Borland* se numeşte *Turbo Link* ($TLINK$).

Link editorul acceptă la intrare un *fişier obiect*, cu extensia implicită $.OBJ$. Ca rezultat al link-editării se va genera un *fişier de cod direct executabil*, cu extensia $.EXE$ sau $.COM$. Opţional, poate fi creat un *fişier listing*, cu extensia implicită $.MAP$, care conţine informaţii despre modul de legare a modulelor în fişierul executabil (adrese de început şi sfîrşit ale modulelor etc.).

Turbo Link este un program care acceptă parametrii transmişi prin linia de comandă. Sintaxa liniei de comandă a linkerului, aşa cum este ea specificată de firma producătoare este descrisă în continuare.:

```
Turbo Link  Version 4.0 Copyright (c) 1991 Borland International

Syntax: TLINK objfiles, exefile, mapfile, libfiles, deffile

@xxxx indicates use response file xxxx
Options:
/m = map file with publics
/x = no map file at all
/i = initialize all segments
/l = include source line numbers
/L = specify library search paths
/s = detailed map of segments
/n = no default libraries
/d = warn if duplicate symbols in libraries
/c = lower case significant in symbols
/3 = enable 32-bit processing
/v = include full symbolic debug information
/e = ignore Extended Dictionary
/t = create COM file (same as /Tc)
/o = overlay switch
/P[=NNNNN] = pack code segments
```

```
/A=NNNN = set NewExe segment alignment factor
/ye = expanded memory swapping
/yx = extended memory swapping
/C  = case sensitive exports and imports
/Txx = specify output file type
  /Tdx = DOS image (default)
  /Twx = Windows image (third letter can be c=COM, e=EXE, d=DLL)
```

7.4 Depanarea

Depanarea (*debugging*, în limba engleză) constă în depistarea şi eliminarea greşelilor din programe. Firma *Borland* produce un program numit *Turbo Debugger* (*TD*) care facilitează depanarea programelor, la nivel de limbaj de asamblare, în mod interactiv, prin intermediul unei interfeţe cu meniuri. Sintaxa lansării în execuţie a programului TD este descrisă în continuare.

```
Turbo Debugger  Version 3.1  Copyright (c) 1988, 92 Borland International

Syntax: TD [options] [program [arguments]]-x

-           turn option x off
-c<file>    Use configuration file <file>
-do,-dp,-ds Screen updating: do=Other display, dp=Page flip, ds=Screen swap
-h,-?       Display this help screen
-i          Allow process id switching
-k          Allow keystroke recording
-l          Assembler startup
-m<#>       Set heap size to # kbytes
-p          Use mouse
-r          Use remote debugging
-rp<#>      Set COM # port for remote link
-rs<#>      Remote link speed: 1=slowest, 2=slow, 3=medium, 4=fast
-sc         No case checking on symbols
-sd<dir>    Source file directory <dir>
-sm<#>      Set spare symbol memory to # Kbytes (max 256Kb)
-vg         Complete graphics screen save
-vn         43/50 line display not allowed
-vp         Enable EGA/VGA palette save
-w          Debug remote Windows program (must use -r as well)
-y<#>       Set overlay area size in Kb
-ye<#>      Set EMS overlay area size to # 16Kb pages
```

Modul de utilizare a programului *TD* pentru depanarea programelor scrise în limbaj de asamblare este prezentat în anexa **??**.

7.5 Apelul procedurilor în limbaj de asamblare

Un fişier ce conţine codul sursă a unei proceduri descrisă în limbaj de asamblare are formatul următor:

```
code  SEGMENT PUBLIC          ; declaraţie de segment de cod
ASSUME CS:code, DS:code

<nume_proc> PROC  NEAR        ; declaraţia procedurii (de tip NEAR)

  <salvarea pe stivă a registrelor ce vor fi folosite în cadrul procedurii>
  <implementarea algoritmului propriu-zis al procedurii>
  <refacerea din stivă a registrelor folosite în cadrul procedurii>
  RET                         ; instrucţiune de întoarcere din procedură

<nume_proc> ENDP              ; terminarea procedurii

code   ENDS                   ; încheierea segmentului de cod

PUBLIC <nume_proc>            ; declararea publică a numelui de procedură
                              ; pentru a fi posibilă apelarea acesteia
                              ; dintr-un alt fişier de cod
END
```

Un fişier ce conţine în codul sursă apelul unei proceduri externe (corpul procedurii se află într-un alt fişier sursă) are formatul următor:

```
EXTRN  <nume_proc> : NEAR     ; declaraţie de procedură externă

code  SEGMENT PUBLIC          ; declaraţie de segment de cod
ASSUME CS:code, DS:code

<declaraţii de date şi directive EQU>

start:
  MOV  AX, CS
  MOV  DS, AX                 ; suprapune segmentul de date peste cel de cod

  <pregăteşte parametrii de intrare ai procedurii>
  CALL <nume_proc>            ; apel de procedură
  <preia parametrii de ieşire ai procedurii>

  MOV  AX, 4C00H              ; terminare program prin apelul funcţiei DOS 4CH
  INT  21H

code ENDS

; lansează în execuţie programul la adresa etichetei start:
END     start
```

7.6 Experimente

I. Faceţi o comparaţie între limbajul de asamblare şi limbajul de nivel înalt C. Ca exemplificare, se consideră problema afişării mesajului *"Hello, world!"* pe ecran. Acest lucru se poate descrie în C cu instrucţiunea:

```
printf("Hello, world!\n");
```

Obţinerea aceluiaşi efect se descrie în limbaj de asamblare cu mai multe instrucţiuni:

```
mov  dx, offset HelloMessage
mov  ah, 9
int  21H
HelloMessage  DB   'Hello, world!',13,10,'$'
```

Editaţi în fişierul $HELLOA.ASM$ următorul program în limbaj de asamblare:

```
code SEGMENT
assume cs:code, ds:code
org 100H

start: mov  ax, cs
       mov  ds, ax
       mov  dx, offset HelloMessage     ; pointer la şirul "Hello, world!"
       mov  ah, 9                       ; funcţie DOS de afişare şir
       int  21H                         ; afişează mesajul "Hello, world!"
       mov  ah, 4CH
       int  21H                         ; terminarea programului

HelloMessage DB 'Hello, world!',13,10,'$'

code ends

end start
```

Asamblaţi fişierul $HELLOA.ASM$ lansînd comanda:

```
TASM HELLOA.ASM
```

Ca efect, se va genera fişierul obiect $HELLOA.OBJ$. Consultaţi cu un editor de texte conţinutul acestui fişier.

Link-editaţi fişierul $HELLOA.OBJ$ lansînd comanda:

```
TLINK  HELLOA.OBJ
```

Ca efect, se va genera fişierul executabil $HELLOA.EXE$ şi fişierul de asocieri $HELLOA.MAP$. Lansaţi în execuţie fişierul $HELLOA.EXE$. Consultaţi cu un editor de texte conţinutul fişierului $HELLOA.MAP$.

Link-editaţi fişierul $HELLOA.OBJ$ lansînd comanda:

```
TLINK  /t HELLOA.OBJ
```

Opţiunea `/t` determină link-editorul să genereze fişierul executabil $HELLOA.COM$, în format $.COM$.

Lansaţi în execuţie fişierul $HELLOA.COM$.

Editaţi în fişierul $HELLOC.C$ următorul program în limbajul C:

```
#include <stdio.h>
void main()
{
  printf("Hello, world!\n");
}
```

Compilaţi fişierul sursă $HELLOC.C$ în mediul $BorlandC$ ($Alt - C, C$ sau $Alt - F9$). Ca efect, se va genera fişierul obiect $HELLOC.OBJ$. Consultaţi cu un editor de texte conţinutul acestui fişier şi comparaţi-l cu cel al fişierului $HELLOA.OBJ$.

Link-editaţi fişierul $HELLOC.C$ ($Alt-C, L$) şi obţineţi fişierul executabil $HELLOC.EXE$ şi fişierul de asocieri $HELLOC.MAP$. Lansaţi în execuţie fişierul $HELLOC.EXE$. Consultaţi cu un editor de texte conţinutul fişierului $HELLOC.MAP$ şi comparaţi-l cu cel al fişierului $HELLOA.MAP$.

Revenind la prompterul sistem, convertiţi fişierul $HELLOC.EXE$ din format $.EXE$ în fişierul $HELLOC.COM$ în format $.COM$, folosind utilitarul $EXE2BIN.EXE$. Comanda este:

```
EXE2BIN HELLOC.EXE HELLOC.COM
```

Studiaţi diferenţele între descrierea unui program în limbaj de asamblare şi descrierea aceluiaşi program în limbajul C urmărind chestiunile enumerate în continuare.

- Comparaţi fişierele $HELLOA.ASM$ şi $HELLOC.C$ din punct de vedere al numărului de linii şi din punct de vedere al dimensiunii acestora;
- Comparaţi conţinutul fişierele $HELLOA.MAP$ şi $HELLOC.MAP$;
- Comparaţi dimensiunile fişierelor $HELLOA.COM$ şi $HELLOC.COM$;
- Comparaţi dimensiunile fişierelor $HELLOA.EXE$ şi $HELLOC.EXE$;
- Comparaţi dimensiunile fişierelor $HELLOA.EXE$ şi $HELLOA.COM$, respectiv $HELLOC.EXE$ şi $HELLOC.COM$;
- Rulaţi "pas cu pas" fişierul $HELLOC.C$ în mediul BorlandC. Rulaţi "pas cu pas" fişierul $HELLOC.COM$ în *Turbo Debugger*. Linia de comandă este:

  ```
  TD HELLOC.COM
  ```

- Rulaţi "pas cu pas" fişierul $HELLOC.EXE$ în *Turbo Debugger*. Linia de comandă este:

  ```
  TD HELLOC.EXE
  ```

- Rulaţi "pas cu pas" fişierul $HELLOA.COM$ în *Turbo Debugger*.

- Comparaţi rularea "pas cu pas" în *Turbo Debugger* a fişierului $HELLOA.COM$ cu cea a fişierului $HELLOC.COM$. Faceţi corespondenţa între modul în care s-a descris programul în limbaj de asamblare şi în limbaj C, cu instrucţiunile efectiv executate de către microprocesor.
- Asamblaţi fişierul $HELLOA.ASM$ cu opţiunea de includere a informaţiei pentru depanare, lansînd comanda:

  ```
  TASM /zi HELLOA
  ```

  Link-editaţi fişierul $HELLOA.OBJ$ cu opţiunea de includere a informaţiei pentru depanare, lansînd comanda:

  ```
  TLINK /v HELLOA, HELLOA_D
  ```

  Se va genera fişierul $HELLOA\_D.EXE$. Comparaţi imaginea în *Turbo Debugger* a fişierului $HELLOA.EXE$ cu cea a fişierului $HELLOA\_D.EXE$.
- Generaţi un listing la asamblarea fişierului $HELLOA.ASM$ cu comanda:

  ```
  TASM /la HELLOA
  ```

  Consultaţi cu un editor de texte fişierul $HELLOA.LST$.

II. Scrieţi un program în limbaj de asamblare care să apeleze o procedură scrisă în limbaj de asamblare, dar care se află într-un alt fişier. Procedura trebuie să înscrie într-un bloc de memorie un octet şi să genereze suma de control asociată blocului. Parametrii de intrare ai procedurii sînt transmişi în următoarele registre:
AX - adresa de început a blocului de memorie;
BX - numărul de locaţii de memorie din bloc (dimensiunea blocului);
DL - octetul care trebuie înscris în fiecare locaţie a blocului de memorie.

La revenirea din procedură, registrul DL conţine suma de control. Suma tuturor locaţiilor blocului, plus cea a registrului DL este zero.

Editaţi în fişierul $PROC.ASM$ următorul cod sursă în limbaj de asamblare:

```
code SEGMENT PUBLIC
ASSUME CS:code, DS:code

WriteDL PROC   NEAR
        PUSH   SI
        PUSH   BX

        MOV    SI, BX   ; Offset-ul byte-ului curent
        MOV    BX, AX
        DEC    BX       ; Adresa începutului de bloc în BX
        XOR    DH,DH    ; Suma de control
muta:   MOV    [BX+SI], DL
        SUB    DH,DL    ; Calculează suma de control
        DEC    SI       ; Pointează la următorul byte
        JNZ    muta

        POP    BX
        POP    SI
```

```
        RET
WriteDL ENDP

code    ENDS
PUBLIC WriteDL

END
```

Editaţi în fişierul $APEL.ASM$ următorul cod sursă în limbaj de asamblare:

```
EXTRN WriteDL : NEAR

code    SEGMENT
ASSUME CS:code, DS:code

Dimens  EQU     3H
DataB   EQU     1H

start:  MOV     AX, CS
        MOV     DS, AX

        MOV     AX, offset Bloc
        MOV     BX, Dimens
        MOV     DL, DataB
        CALL    WriteDL

        MOV     AX, 4C00H
        INT     21H

Bloc DB  Dimens DUP(?)

code ENDS

END start
```

Din fişierele sursă, realizaţi fişierul executabil $TEST.EXE$, lansînd comenzile:

```
TASM PROC
TASM APEL
TLINK APEL PROC, TEST
```

Urmăriţi execuţia programului prin rulare "pas cu pas" cu *Turbo Debugger*:

```
TD TEST
```

Lucrarea 8

Declararea datelor şi a segmentelor

Această lucrare prezintă modul în care se declară în limbaj de asamblare structurile de date necesare oricărui program. La microprocesoarele familiei 8086, memoria este "segmentată". Gestionarea memoriei în limbaj de asamblare revine exclusiv programatorului. Exerciţiile propuse în finalul lucrării demonstrează că nu este suficientă exprimarea algoritmului în limbajul de asamblare (atît de sărac faţă de C...), ci este necesară înţelegerea modului de administrare a memoriei segmentate.

8.1 Declararea datelor

Sintaxa declaraţiei unei date (variabile) este:

[<nume>] <tip> <listă expresii de iniţializare>

sau

[<nume>] <tip> <factor> DUP (<listă expresii de iniţializare>)

<nume> este numele asociat variabilei, prin care se va putea face o referire la aceasta. Numele este opţional. Dacă într-o declaraţie de date nu apare nici un nume, sînt alocate date, dar adresa de început nu poate fi referită printr-un nume simbolic. Numele poate fi considerat ca fiind un pointer (adresa începutului zonei de memorie) la data asociată.

<tip> reprezintă unul din cuvintele rezervate ce desemnează un tip de date. Tipurile de date existente şi cuvintele rezervate pentru desemnarea acestora sînt prezentate în tabelul 8.1.

<listă expresii de iniţializare> specifică valorile cu care este iniţializată zona de memorie asociată datelor definite. În lipsa unor valori concrete, prin caracterul '?', se indică faptul că zona de date este rezervată dar nu şi iniţializată. În lista de iniţializare pot apare constante sau expresii evaluate de către asamblor (static).

Operanzii expresiilor pot fi constante numerice, alfanumerice şi nume pentru care asamblorul asociază valori: nume de date, etichete de instrucţiuni, nume de segmente, etc. Constantele întregi pot reprezenta numere scrise în bazele 2, 8, 10 sau 16. Baza în care

<tip>	*Semnificaţie (în limba engleză)*	*Tip*
DB	Define Byte (1 byte)	byte
DW	Define Word (2 bytes)	word
DD	Define Doubleword (4 bytes)	dword
DQ	Define Quadword (8 bytes)	qword
DT	Define Ten bytes (10 bytes)	tbyte

Tabelul 8.1: Tipuri de date şi cuvinte rezervate pentru desemnarea acestora.

este reprezentat numărul este desemnată de o literă aflată la sfîrşitul şirului de cifre ce formează constanta $\{B, Q, D, H\}$. Implicit, dacă nu apare nici una din literele ce desemnează baza de numeraţie, se consideră că numărul este exprimat în baza 10. Constantele în virgulă flotantă se pot exprima în formatul cu mantisă şi exponent. Constantele alfanumerice, individuale sau în şiruri, sînt incluse între caractere apostrof simple (') sau duble ("). O constantă alfanumerică este asociată de către asamblor cu codul ASCII al caracterului corespunzător.

O *dată* iniţializată cu un *nume de dată* poate fi considerată similară cu o variabilă de tip *pointer*. În acest caz, data conţine adresa unei date ce are o valoare şi nu o valoare propriu-zisă. Pointerii de dimensiune 2 baiţi conţin adrese cu *referinţe apropiate* (în acelaşi segment de memorie). Pointerii de dimensiune 4 baiţi conţin adrese cu *referinţe îndepărtate* (în segmente de memorie disjuncte).

<factor> specifică de cîte ori se repetă *<lista de expresii de iniţializare>* ce este inclusă între paranteze rotunde. Valoarea iniţială poate fi caracterul '?' (neiniţializat), orice expresie constantă, orice valoare evaluată la o constantă de către asamblor sau un alt operator DUP. Prin utilizarea operatorului DUP se pot defini structuri de date similare cu matricile uni sau multidimensionale.

Datele reprezentate pe mai mult de un byte sînt stocate în memorie conform regulii 'little-endian' (cel mai puţin semnificativ byte este stocat la cea mai mică adresă de memorie).

Adresa locaţiei de memorie curente (unde se alocă data) poate să fie referită prin simbolul $ sau prin expresia *this <tip>*.

Adresa logică (offset-ul) a unei date poate fi obţinută prin operatorul *OFFSET*. Adresa de bază a segmentului în care a fost definită data poate fi obţinută prin operatorul *SEG*.

Adresarea unei date ca fiind de alt tip decît a fost declarată este posibilă prin utilizarea operatorului de conversie *PTR*. Sintaxa unei expresii construite cu acest operator este:

<tip> PTR <expresie>

Ca efect, *<expresia>* va fi interpretată prin intermediul atributului *<tip>*. În continuare se prezintă, ca exemplu, o porţiune de cod.

```
. . .
DataW   dw    10 dup (?)
DataB   db    10 dup (?)
. . .
```

```
mov al, byte ptr DataW  ; mută în AL primul byte din cei 10 x 2 rezervaţi
                        ; pentru DataW
mov ax, word ptr DataB  ; mută în AX primii doi baiţi din cei 10 rezervaţi
                        ; pentru DataB
```

În continuare se prezintă listingul unor declaraţii de date în limbaj de asamblare. Listingul a fost obţinut cu TASM (cu opţiunea `/la`). Listingul prezintă, în partea din stînga, adresa de memorie şi datele rezultate în urma asamblării, iar în partea din dreapta codul, aşa cum a apărut în fişierul sursă.

```
 1   0000                          data segment
 2   0000  10                      D1    db   16
 3   0001  000C                    D2    dw   4*3
 4   0003  FFFFFFFF                D3    dd   4294967295
 5   0007  ????????????????        D4    dq   ?
 6   000F  000000000002DFDC1C35    D5    dt   12345678901D
 7   0019  12345678901234567890    D6    dt   12345678901234567890
 8   0023  00000000012345678901    D7    dt   12345678901
 9   002D  05 17 1D AF             D8    db   0101B,27O, 29, 0AFH
10   0031  33                      D9    db   3+"0"
11   0032  01 02 03                      db   1, 2, 3
12   0035  0032r                   D10   dw   D9+1
13   0037  61 62 63                D11   db   97, 98, 99
14   003A  61 62 63                D12   db   'a', 'b', 'c'
15   003D  61 62 63                D13   db   'abc'
16   0040  61 62 63                D14   db   "abc"
17   0043  4D 65 73 61 6A 3A 0D+   D15   db   'Mesaj:', 13, 10, '$'
18         0A 24
19   004C  53 61 6C 75 74 21       D16   db   "Salut!"
20   0052  06                      D17   db   $-D16
21   0053  07                      D18   db   this byte-D16
22   0054  004Cr                   D19   dw   OFFSET D16
23   0056  0000s                   D20   dw   SEG D16
24   0058  61 62                   D21   db   "ab"
25   005A  00006162                D22   dd   "ab"
26   005E  00000061                D23   dd   "a"
27   0062  54 65 78 74 00          D24   db   "Text",0
28   0067  0062r                   D25   dw   D24
29   0069  00000062sr              D26   dd   D24
30   006D  0A*(00000001)           D27   dd   10 dup (1)
31   0095  10*(??)                 D28   db   16 dup (?)
32   00A5  02*(02*(02*     +       D29   dd   2 dup (2 dup (2 dup (0)))
33         (00000000)))
34   00C5  05*(03*(01) 02)         D30   db   5 dup (3 dup (1), 2)
35   00D9                          data ends
36                 end
```

- Pe linia 2 se declară o dată de tip *byte* cu numele `D1`, iniţializată cu constanta `16`.

- Pe linia 3 se declară o dată de tip *word* cu numele `D2`, iniţializată cu valoarea `12`. Valoarea este determinată în momentul asamblării prin evaluarea expresiei `4*3`.

- Pe linia 4 se declară o dată de tip *dword* cu numele `D3`, iniţializată cu cea mai mare valoare posibilă, exprimată în baza 10. Exprimarea în baza 16 este `FFFFFFFF`.

- Pe linia 5 se declară o dată de tip *qword* cu numele `D4`, lăsată neiniţializată.

- Pe liniile 6, 7 şi 8 se declară date de tip *tbyte* cu numele `D5`, `D6` şi `D7`, iniţializate cu diverse valori. De remarcat corespondenţa între valorile din codul sursă şi cele din memorie.

- Pe linia 9 se declară o dată de tip *byte* cu numele `D8`, iniţializată cu trei valori constante exprimate în bazele de numeraţie 2, 8, 10 şi 16.

- Pe linia 10 se declară o dată de tip *byte* cu numele `D9`, iniţializată cu codul ASCII al caracterului "3", obţinut prin însumarea valorii întregi 3 la codul ASCII al caracterului "0".

- Pe linia 11 se declară o dată de tip *byte* căreia nu i se asociază nici un nume. Aceste date pot fi referite prin numele altor date.

- Pe linia 12 se declară o dată de tip *word* cu numele `D10`, iniţializată cu o valoare constantă obţinută prin evaluarea unei expresii în care intră şi numele unei date. Valoarea obţinută este adresa de început a datelor declarate pe linia 11. `D10` poate fi privit ca un pointer la datele declarate pe linia 11.

- Pe liniile 13, 14, 15 şi 16 se declară date de tip *byte* cu numele `D11`, `D12`, `D13` şi `D14`, iniţializate cu aceleaşi valori exprimate în moduri diferite: întregi, caractere, şir delimitat de caractere apostrof simplu, şir delimitat de caractere apostrof dublu.

- Pe linia 17 se declară o dată de tip *byte* cu numele `D15`, iniţializată cu o listă de expresii de diferite tipuri (şir de caractere, întregi, caracter). Caracterele ASCII asociate codurilor `13` şi `10` sînt caracterele de control `CR` şi `LF`. Şirul astfel definit poate fi transmis ca argument al funcţiei DOS 9H de tipărire şir de caractere.

- Pe linia 19 se declară o dată de tip *byte* cu numele `D16`, iniţializată cu un şir de caractere ce conţine şi un semn de punctuaţie.

- Pe linia 20 se declară o dată de tip *byte* cu numele `D17`, iniţializată cu dimensiunea datelor `D16`. Adăugarea unor caractere şirului `D16` determină actualizarea automată a valorii din `D17`, datorită modului în care a fost scrisă expresia de iniţializare.

- Pe linia 21 se declară o dată de tip *byte* cu numele `D18`, iniţializată cu o valoare provenind dintr-o expresie similară cu cea de pe linia 20.

- Pe liniile 22 şi 23 se declară date de tip *word* cu numele `D19` şi `D20`, iniţializate cu valorile offset-ului şi adresei de bază a segmentului în care a fost definită data `D16`. Adresa datei `D16` poate fi considerată `D20:D19`.

- Pe liniile 24, 25 şi 26 se observă modul în care se ordonează datele de mai mulţi baiţi, conform regulii "little-endian".

- Pe linia 27 se declară o dată de tip *byte* cu numele `D24`, iniţializată cu un şir de caractere urmat de un întreg.

- Pe linia 28 se declară o dată de tip *word* cu numele `D25`, ce poate fi interpretată ca pointer apropiat la data `D24`. `D25` are valoarea offset-ului datei `D24`.

- Pe linia 29 se declară o dată de tip *dword* cu numele `D26`, ce poate fi interpretată ca pointer îndepărtat la data `D24`. `D25` are ca valoare doi baiţi segmentul şi doi baiţi offset-ul datei `D24`.

- Pe linia 30 se declară o dată de tip *dword* cu numele `D27`, inițializată cu un vector de 10 valori întregi.

- Pe linia 31 se declară o dată de tip *byte* cu numele `D28`, de dimensiune 16 baiți. Zona de memorie asociată se lasă neinițializată.

- Pe linia 32 se declară o dată de tip *dword* cu numele `D29`, de dimensiune $2 \times 2 \times 2 = 8$. Zona de memorie asociată este în întregime inițializată cu 0.

- Pe linia 34 se declară o dată de tip *byte* cu numele `D30`. Zona de memorie asociată este inițializată cu 5 de pattern-uri: 1, 1, 1, 2. În total, dimensiunea datei `D30` este de 20 de baiți.

Zona de memorie rezervată pentru declarațiile anterioare este prezentată în figura 8.1.

```
ds:0000 10 0C 00 FF FF FF FF 00 00 00 00 00 00 00 00 35
ds:0010 1C DC DF 02 00 00 00 00 00 90 78 56 34 12 90 78
ds:0020 56 34 12 01 89 67 45 23 01 00 00 00 00 05 17 1D
ds:0030 AF 33 01 02 03 32 00 61 62 63 61 62 63 61 62 63
ds:0040 61 62 63 4D 65 73 61 6A 3A 0D 0A 24 53 61 6C 75
ds:0050 74 21 06 07 4C 00 56 5A 61 62 62 61 00 00 61 00
ds:0060 00 00 54 65 78 74 00 62 00 62 00 56 5A 01 00 00
ds:0070 00 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00
ds:0080 00 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00
ds:0090 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:00C0 00 00 00 00 00 01 01 01 02 01 01 01 02 01 01 01
ds:00D0 02 01 01 01 02 01 01 01 02 00 00 00 00 00 00 00
```

Figura 8.1: Zonă de memorie rezervată prin declarații de date.

8.2 Declararea segmentelor

Microprocesoarele familiei 8086 implementează un mecanism de segmentare a memoriei. Se pot defini oricîte segmente logice, de dimensiune maximă 64 KB. Segmentele pot fi disjuncte sau suprapuse parțial sau total. La un moment dat, microprocesorul poate avea acces numai la patru segmente. Adresele de bază ale acestor segmente se găsesc în cele patru registre segment: *CS*, *DS*, *SS* şi *ES*.

8.2.1 Sintaxa declarației de segment

Sintaxa declarației unui segment este următoarea:

<nume> SEGMENT [<aliniere>] [<combinare>] [<clasa>]
<corpul segmentului>
<nume> ENDS

<nume> definește numele segmentului. Acest nume poate fi unic sau poate fi același nume

atribuit şi altor segmente din program. Segmentele cu nume identic sînt tratate ca şi cum ar fi acelaşi segment. De exemplu, dacă este necesar să se plaseze diferite porţiuni ale unui singur segment în module sursă diferite, atunci segmentului îi este atribuit acelaşi nume în ambele module.

<*aliniere*> este un cîmp opţional ce defineşte tipul adresei de început a segmentului. Prin valorile cîmpului aliniere: PARA/BYTE/WORD/PAGE se specifică faptul că adresa de început a zonei de memorie rezervată segmentului este divizibilă cu 16/1/2/256. Valoarea implicită a acestui cîmp este *PARA*.

<*combinare*> este un cîmp opţional care defineşte modul de combinare a segmentelor ce au acelaşi nume. Informaţia transmisă de programator prin intermediul acestui cîmp este folosită de către link-editor. Cîmpul combinare poate avea cinci valori, cuvinte rezervate.

- PUBLIC - concatenează toate segmentele cu acelaşi nume pentru a forma un segment unic, contiguu. Toate instrucţiunile şi adresele de date din noul segment se referă la un registru de segment unic, iar toate deplasamentele sînt ajustate pentru a reprezenta distanţa de la începutul segmentului.
- STACK - concatenează toate segmentele cu acelaşi nume pentru a forma un segment unic, contiguu. Acest tip de combinare este identic cu tipul de combinare PUBLIC, cu excepţia faptului că toate adresele din noul segment se referă la registrul SS al segmentului. Registrul indicator de stivă (SP) este iniţializat la lungimea segmentului. Segmentul stivă din programul utilizatorului va folosi în mod normal STACK întrucît acesta iniţializează automat registrul SS. Dacă se creează un segment de stivă şi nu se utilizează tipul STACK, se vor da implicit instrucţiuni pentru iniţializarea registrelor SS şi SP.
- COMMON - creează segmente suprapuse prin plasarea începutului tuturor segmentelor cu acelaşi nume la aceeaşi adresă. Lungimea zonei rezultate este lungimea celui mai lung segment. Toate adresele din segmente sînt raportate la aceeaşi adresă de bază. Dacă variabilele sînt iniţializate în mai multe segmente ce au acelaşi nume şi tipul COMMON, datele cel mai recent iniţializate înlocuiesc toate datele iniţializate anterior.
- MEMORY - concatenează toate segmentele cu aceleaşi nume pentru a forma un segment unic, contiguu. Link-editorul tratează segmentele MEMORY în aceleaşi mod ca şi segmentele PUBLIC. Segmentul curent va fi plasat în memorie în spaţiul rămas disponibil după plasarea în memorie a celorlalte segmente.
- AT <*adresă*> - determină ca toate adresele de etichete şi variabile definite în segment să fie relative la o adresă. Adresa poate fi orice expresie corectă, dar nu poate conţine o referinţă anticipată (o referinţă la un simbol definit mai tîrziu în fişierul sursă). Un segment AT nu conţine, de obicei, cod sau date iniţializate. În schimb, el reprezintă un model de adresă ce poate fi plasat peste codul sau datele aflate deja în memorie, cum ar fi o zonă tampon sau o altă locaţie de memorie absolută definită de partea hardware. Editorul de legături nu va genera cod sau date pentru segmentele AT, dar datele sau codul existent pot fi accesate prin nume dacă se specifică o etichetă într-un segment AT.

Dacă nu apare nici un tip combinare, segmentul va avea un tip special. Segmentele cu acelaşi nume nu vor fi combinate. În schimb, fiecare segment va primi propriul segment

fizic atunci cînd este încărcat în memorie. Deşi un nume de segment dat poate fi utilizat de mai multe ori într-un fişier sursă, fiecare definiţie de segment ce utilizează acel nume trebuie să aibe exact aceleaşi atribute. Dacă sînt precizate tipuri de combinare într-o definiţie iniţială de segment, definiţile ulterioare pentru acel segment nu necesită specificarea nici unui alt tip de combinare.

<*clasa*> este un cîmp opţional care specifică modul de asociere a segmentelor ce au nume diferite, dar scopuri similare. Tipul se foloseşte pentru a controla ordinea segmentelor şi pentru a identifica segmentul de cod. Numele de clasă va fi inclus între caractere apostrofuri simple (’). Segmentele pentru care nu este stabilit explicit un nume de clasă vor avea numele de clasă nul. Link-editorul nu impune restricţii asupra numărului sau dimensiunii segmentelor dintr-o clasă.

<*corpul segmentului*> conţine instrucţiunile care se vor asambla în spaţiul de memorie al segmentului şi declaraţiile de date prin care se vor rezerva zone de memorie.

8.2.2 Asocierea segmentelor cu registre

Deşi la un moment dat doar patru segmente pot fi accesate, într-un program se pot defini mult mai multe segmente. De cele mai multe ori, instrucţiunile în limbaj de asamblare apelează datele din memorie prin adresa lor logică, segmentul fiind determinat implicit de către procesor. În tabelul 6.4 sînt prezentate segmentele considerate implicite în cazul unor apeluri la memorie. De exemplu, instrucţiunea JMP consideră că segmentul implicit este cel asociat cu registrul CS. Instrucţiunile MOV consideră că segmentul este asociat cu registrul DS.

Determinarea segmentului în care se face referirea la memorie se face de către microprocesor, în momentul execuţiei efective a instrucţiunii. În limbaj de asamblare se poate apela o dată care nu se află în segmentul implicit prin plasarea înainte de numele datei a numelui registrului de segment.

```
mov  ax, Data1     ; apel la Data1 aflatã în segmentul DS (implicit)
mov  ax, ES:Data2  ; apel la Data2 aflatã în segmentul ES (explicit)
mov  ax, CS:Data3  ; apel la Data3 aflatã în segmentul CS (explicit)
```

Prin utilizarea directivei $ASSUME$, programatorul poate transfera asamblorului sarcina stabilirii registrelor explicite în cazul apelării unor date declarate în segmente diferite.

Directiva $ASSUME$ se foloseşte pentru a indica asamblorului asocierea dintre un segment şi un registru segment. Această directivă nu controlează activitatea procesorului. Programatorul trebuie să scrie în program instrucţiuni explicite de încărcare a registrelor de segment cu valorile declarate în directiva $ASSUME$. Directiva ASSUME afectează numai consideraţiile referitoare la momentul asamblării. Există şi situaţii cînd se pot folosi instrucţiuni pentru modificarea consideraţiilor referitoare la momentul execuţiei.

Sintaxa directivei $ASSUME$ este:

ASSUME <registru segment> : <nume>
ASSUME <registru segment> : NOTHING
ASSUME NOTHING

<*registru segment*> poate fi oricare din cele patru nume de registre segment: CS, DS, ES sau SS.

<*nume*> trebuie să fie numele segmentului ce se va asocia cu <*registru segment*>. Instrucţiunile ulterioare directivei, care consideră un registru implicit pentru referirea în mod automat a etichetelor sau a variabilelor, presupun că dacă segmentul implicit este <*registru segment*>, atunci eticheta sau variabila accesată se află în segmentul <*nume*>.

Cuvîntul cheie *NOTHING* anulează selecţia de segmente curentă, pentru unul sau toate registrele de segment.

În mod normal, o singură instrucţiune *ASSUME* defineşte toate cele patru registre segment la începutul registrului sursă. Totuşi, directiva *ASSUME* poate fi folosită în orice punct pentru a modifica supoziţiile referitoare la segment.

În continuare este prezentat un program pentru exemplificarea efectului directivei *ASSUME*. Pe coloana din stînga apare codul sursă. Pe coloana din dreapta apare codul dezasamblat cu *TurboDebbuger*.

```
A  segment
A1   dw  1
A2   dw  2
A  ends

B  segment
B1   dw  1
B2   dw  2
B  ends

code segment
assume cs:code, ds:A, es:B
C1  dw 1
C2  dw 2

start:
; inițializează registrele de segment
; conform directivelor "assume"
  mov  ax, A                              mov    ax,5A56
  mov  ds, ax                             mov    ds,ax
  mov  ax, B                              mov    ax,5A57
  mov  es, ax                             mov    es,ax

; referiri la date
  mov  ax, A1                             mov    ax,[0000]
  mov  B1, ax                             mov    es:[0000],ax
  mov  es:B2, ax                          mov    es:[0002],ax
  mov  dx, A2                             mov    dx,[0002]
  mov  B2, dx                             mov    es:[0002],dx

; suprapune segmentul DS peste CS
; segmentul A nu mai poate fi referit
  mov  ax, cs                             mov    ax,cs
  mov  ds, ax                             mov    ds,ax
```

```
  assume ds:code

; referiri la date în segmentul DS
; care, în acest caz,
; coincide cu segmentul CS

  mov  ax, B1                                   mov    ax,es:[0000]
  mov  C1, ax                                   mov    [0000],ax
  mov  ds:C2, ax                                mov    [0002],ax
  mov  cs:C1, ax                                mov    cs:[0000],ax

  mov  B2, ax                                   mov    es:[0002],ax

; asamblorul semnalează o eroare
; la linia următoare
; mov  A2, ax
code ends

end start
```

Au fost definite două segmente: A şi B. În fiecare segment s-au definit două date de tip *word*. La începutul segmentului de cod, prin directiva $ASSUME$, s-au asociat cele două segmente cu registrele DS şi ES. La începutul programului efectiv, imediat după eticheta `start:`, s-au inserat instrucţiuni de transfer care să iniţializeze registrele de segment cu valorile presupuse. Ca efect al directivei $ASSUME$, asamblorul a asamblat instrucţiunea `mov B1, ax` sub forma: `mov es:[0000],ax`. Offset-ul datei `B1` (`0000`) a fost precedat de numele de segment `es`.

Prin schimbarea ipotezei asupra conţinutului registrului DS (prin directiva: `assume ds:code`) segmentul A nu mai poate fi accesat. Din acest motiv, în cazul existenţei instrucţiunii de pe ultimul rînd (`mov A2, ax`), asamblorul va genera o eroare. Mesajul de eroare este:

```
**Error** assume.ASM(50) Can't address with currently ASSUMEd segment registers
```

8.2.3 Iniţializarea registrelor

Registrele CS şi IP sînt iniţializate prin specificarea unei adrese de început cu directiva END. Sintaxa directivei este:

END [<adresa de început>]

Cîmpul *<adresa de început>* este o etichetă sau o expresie ce identifică adresa la care utilizatorul doreşte începerea execuţiei atunci cînd programul este încărcat în memorie. Dacă un program este format dintr-un singur modul sursă, este necesară ca adresa de început să se precizeze în acel modul. Dacă un program are mai multe module, toate modulele se vor termina cu directiva END. Numai o directivă END defineşte adresa de început. Link-editorul nu generează eroare la omiterea adresei de început, dar execuţia va începe probabil la o adresă greşită.

Registrul DS trebuie iniţializat la adresa segmentului ce va fi folosit pentru date. Adresa segmentului va fi încărcată în registrul DS. Deoarece o valoare din memorie nu poate

fi încărcată direct într-un registru segment, iniţializarea registrului DS necesită două instrucţiuni de transfer. Instrucţiunile de iniţializare a registrului DS apar, de obicei, la începutul sau foarte aproape de începutul segmentului de cod. Un exemplu de iniţializare a registrului de segment DS este prezentat în continuare:

```
date1 SEGMENT
       ...
date1 ENDS

code1 SEGMENT
ASSUME cs:code1, ds:date1
start:
  mov  ax, date1  ; AX <= adresa de bază a segmentului date1
  mov  ds, ax     ; DS <= AX
  ...
code1 ENDS
END start
```

Registrele SS şi SP sînt iniţializate automat la lansarea în execuţie a unui program. Registrul de segment SS este iniţializat automat la valoarea ultimului segment din codul sursă cu tipul de combinare STACK. Registrul SP este iniţializat automat la dimensiunea segmentului de stivă. Astfel, SS:SP indică iniţial adresa de sfirşit a segmentului de stivă. Segmentul de stivă poate fi iniţializat sau reiniţializat direct prin schimbarea valorilor SS şi SP, prin program. Întrucît întreruperile hardware folosesc aceeaşi stivă ca şi programul, acestea trebuie dezactivate în timpul schimbării stivei.

Registrul ES nu este iniţializat automat. Dacă programul foloseşte segmentul de date auxiliar (folosit implicit în operaţiile cu şiruri), programatorul trebuie să iniţializeze explicit registrul ES. Iniţializarea se face prin transferarea în registrul ES a valorii corespunzătoare adresei de început a segmentului auxiliar, folosind două instrucţiuni de transfer.

8.2.4 Definirea simplificată a segmentelor

Versiunile actuale de asambloare au introdus un nou mecanism de definire simplificată a segmentelor. Acest mecanism este mai uşor de utilizat mai ales cînd se doreşte legarea mai multor module scrise în diferite limbaje de programare. Pentru utilizarea acestui mod de definire trebuie specificat mai întîi modelul de memorie pentru care este scris programul. Modelul de memorie are o semnificaţie software, precizînd modul în care se utilizează segmentele în cadrul programului respectiv. Caracteristicile celor 6 modele de memorie sînt sintetizate în tabelul 8.2.

Declararea modelului trebuie să preceadă alte pseudo-instrucţiuni sau instrucţiuni care implică referiri la segmente. Declararea modelului se face cu ajutorul unei pseudoinstrucţiuni de forma:

.MODEL <tip model>

8.3 Experimente

I. Editaţi într-un fişier programul care urmează. Denumiţi fişierul *DEBUG.ASM*.

Model de memorie	*Volum de date*	*Volum de cod*	*Observaţii*
TINY	< 64 KB - adrese relative	< 64 KB - apeluri "near"	- date+cod+stivă < 64 KB - toate segmentele fac parte din acelaşi grup - programele sînt de tip .COM
SMALL	< 64 KB - adrese relative	< 64 KB - apeluri "near"	- segmente de date şi cod distincte
MEDIUM	< 64 KB - adrese relative	> 64 KB - apeluri "far"	
COMPACT	> 64 KB - adrese fizice	< 64 KB - apeluri "near"	- deşi datele pot avea un volum mai mare de 64 KB, există o restricţie: o structură de date nu poate depăşi 64 KB
LARGE	> 64 KB - adrese fizice	> 64 KB - apeluri "far"	- deşi datele pot avea un volum mai mare de 64 KB, există o restricţie: o structură de date nu poate depăşi 64 KB
HUGE	> 64 KB - adrese fizice	> 64 KB - apeluri "far"	- o structură de date poate avea un volum mai mare de 64 KB

Tabelul 8.2: Caracteristici ale modelelor de memorie.

```
code SEGMENT
      assume cs:code, ds:code, es:code
      org 100H

start:
      mov  ax, cs
      mov  ds, cx
      mov  si, OFFSET Sir_sursa
      mov  di, OFFSET Sir_dest
      mov  cx, 17
rep   movs Sir_dest, Sir_sursa
      mov  ah, 4ch
      int  21h

Sir_sursa DB 'Acesta este sirul'
Sir_dest  DB 17 dup (?)

code ends
end start
```

Scopul programului este de a copia un şir de caractere definit cu numele `Sir_sursa` într-o altă zonă de memorie unde i s-a rezervat un spaţiu de aceeaşi dimensiune sub numele `Sir_dest`. Programul conţine cîteva greşeli deşi nici asamblorul şi nici link-editorul nu furnizează vreun mesaj de eroare. Realizaţi următoarele:

- Studiaţi efectul fiecărei instrucţiuni sau directive de asamblare care apare în codul sursă. Studiaţi instrucţiunea de transfer de şiruri `movs` şi efectul prefixului `rep`.

- Studiaţi modul de definire a datelor şi a segmentelor. Ţinînd cont de directivele `assume` şi `org`, estimaţi offset-ul la care se vor găsi datele definite.
- Asamblaţi şi link-editaţi fişierul *DEBUG.ASM*.
- Utilizînd *Turbo Debugger* corectaţi greşelile depistate în program.
- Modificaţi definirile de date şi programul pentru a minimiza modificările ulterioare determinate de redefinirea şirului sursă, ca lungime şi conţinut.
- Studiaţi diferenţele dintre instrucţiunile:

```
mov  si, OFFSET Sir_sursa
mov  si, word PTR Sir_sursa
mov  al, Sir_sursa
```

II. Studiaţi conţinutul fişierului *AFISARE.ASM*. Codul sursă descrie un program de iniţializare a registrului AX şi ulterior afişează conţinutul acestuia pe ecran, exprimat în baza 10.

- Studiaţi modul de definire a datelor şi segmentelor.
- Cum explicaţi că programul utilizează stiva (prin instrucţiuni `PUSH` şi `POP`) dar nu are un segment de stivă definit explicit? Verificaţi conţinutul stivei de-a lungul execuţiei programului.
- Studiaţi modul în care este descrisă şi apelată o procedură din acelaşi segment de cod. Apelaţi la *Turbo Debugger* pentru a intra în detalii referitoare la modul de asamblare a instrucţiunilor implicate în apelul şi revenirea din procedură.

III. Studiaţi conţinutul fişierului *MEDIE.ASM*. Codul sursă descrie un program care însumează valorile dintr-un vector de date de tip *word* şi ulterior determină valoarea medie a elementelor.

- Studiaţi modul în care s-a parametrizat definirea datelor. Modificaţi programul pentru a însuma un număr dublu de date. Cîte linii de cod trebuie modificate?
- Studiaţi instrucţiunile de împărţire (`div`).
- Plecînd de la codul considerat, scrieţi o procedură care primeşte ca parametrii un pointer la un vector de date şi dimensiunea acestuia şi întoarce într-un registru suma elementelor vectorului de date. Scrieţi un program care apelează procedura de două ori, cu parametrii diferiţi.

Lucrarea 9

Programarea cu întreruperi software

Un sistem de calcul cu microprocesor, aşa cum este prezentat în figura 1, nu poate fi imaginat fără dispozitive periferice de intrare/ieşire. De obicei, aceste periferice sînt foarte lente faţă de microprocesor şi sînt (mai mult sau mai puţin) controlate de acesta. Microprocesorul poate trata perifericele în două moduri, fiecare cu avantaje şi dezavantaje.

Tratarea perifericelor prin polling (prin program) presupune suspendarea rulării programului curent de către microprocesor şi interogarea perifericelor. Avantajul constă în faptul că programatorul controlează precis momentele cînd programul principal poate fi suspendat. Dezavantajul constă în faptul că un periferic "grăbit" (de exemplu o placă de achiziţie de date) nu îşi poate permite să aştepte un timp nedefinit pentru a fi servit. În plus, interogarea perifericelor se face, şi consumă timp, chiar dacă nici un periferic nu are nimic de comunicat.

Tratarea perifericelor prin întreruperi presupune existenţa unui semnal hardware exterior microprocesorului prin activarea căruia perifericele cer să fie deservite de către acesta. La activarea semnalului de întrerupere, microprocesorul îşi suspendă execuţia programului curent, depistează perifericul care a lansat cererea de întrerupere şi lansează procedura specifică de tratare a acestuia.

Această lucrare prezintă sistemul de întreruperi al microprocesorului 8086 şi apelarea din limbaj de asamblare a funcţiilor BIOS şi DOS, prin întreruperi software.

9.1 Sistemul de întreruperi

De obicei, setul de instrucţiuni al microprocesorului conţine instrucţiuni pentru dezactivarea (mascarea) întreruperilor. Acestea sînt folosite în cazul în care microprocesorul urmează să execute o porţiune critică de program, ce nu trebuie perturbată de evenimente externe. Pentru cazuri deosebite, CPU are un pin dedicat pentru primirea întreruperilor nemascabile.

Microprocesoarele familiei 8086 recunosc 256 de întreruperi. Există posibilitatea apelării acestor întreruperi şi prin program, utilizînd instrucţiuni specifice. Aceste întreruperi sînt denumite *întreruperi software* pentru a fi deosebite de *întreruperile hardware* ce sînt determinate de activarea unui semnal provenit din exteriorul microprocesorului. Întreruperile software sînt tratate la fel ca şi cele hardware cu deosebirea că acestea nu pot fi mascate. Întreruperile software

sînt sincrone cu ceasul sistem şi au loc la momente de timp predictibile, specificate prin program. Întreruperile hardware sînt în general asincrone şi inpredictibile în timp. Instrucţiunea de întrerupere software la microprocesorul 8086 are sintaxa:

INT <număr întrerupere>

Pe baza *<numărului întreruperii>* (între 0 şi 255), microprocesorul determină adresa procedurii de tratare a întreruperii.

9.1.1 Întreruperi externe

Microprocesorul 8086 are doi pini dedicaţi pentru primirea întreruperilor din exterior: *INTR* şi *NMI*. Pinul *INTR* (INTerrupt Request) este comandat de un *controler programabil de întreruperi* (8259A) care, la rîndul său, este conectat la perifericele care pot lansa întreruperi. Circuitul 8259A este comandat de microprocesor prin software, fiind văzut de acesta ca un set de porturi I/O. Sarcina controlerului de întreruperi este de a primi şi ierarhiza întreruperile de la periferice şi de a activa pinul *INTR*.

CPU verifică starea pinului *INTR* la terminarea fiecărei instrucţiuni. Întreruperea este ignorată dacă indicatorul *IF* (Interrupt-enable Flag) este resetat. Starea indicatorului *IF* poate fi controlată prin program cu instrucţiunile *CLI* (Clear IF) şi *STI* (SeT IF). CPU semnalizează acceptarea întreruperi prin executarea a doi cicli de confirmare a întreruperii (INTA - INTerrupt Acknowledge). Primul ciclu INTA semnalizează controlerului de întreruperi că întreruperea a fost acceptată. În al doilea ciclu INTA, controlerul de întreruperi răspunde prin plasarea pe bus-ul de date a numărului întreruperii asociat cu dispozitivul care a lansat întreruperea. Asocierea între periferic şi numărul de întrerupere este făcută prin software, la iniţializarea controlerului de întreruperi. CPU citeşte numărul întreruperii şi îl utilizează pentru a determina adresa procedurii de tratare a întreruperii.

O întrerupere externă poate sosi şi pe pinul denumit *NMI* (Non-Maskable Interrupt). Întreruperile venite pe această linie nu pot fi mascate şi sînt prioritare faţă de întreruperile venite pe pinul *INTR*. Întreruperile nemascabile sînt predefinit asociate cu numărul 2. Din acest motiv, nu mai este necesar ca CPU să execute ciclii INTA şi poate apela imediat procedura de tratare a întreruperii nemascabile.

9.1.2 Întreruperi software

Execuţia instrucţiunilor *INT* (INTerrupt) generează imediat o întrerupere. Numărul întreruperii este inclus în codul instrucţiunii şi permite CPU determinarea imediată a adresei procedurii de tratare a întreruperii. Prin utilizarea întreruperilor software se poate testa procedura de tratare a întreruperii provenite de la un dispozitiv extern.

Întreruperea cu numărul 0 (*Divide error*) este generată de CPU cînd, după o instrucţiune de împărţire, cîtul are dimensiune mai mare decît locaţia specificată ca destinaţie.

Întreruperea cu numărul 1 (*Single Step*) este generată de CPU cînd indicatorul TF (Trap Flag) este setat. În acest caz, microprocesorul intră în modul de lucru de depanare ("pas cu pas").

Dacă indicatorul OF (Overflow Flag) este setat, la terminarea execuţiei unei instrucţiuni INTO (INTerrupt on Overflow) se generează o întrerupere cu numărul 4.

Toate întreruperile interne (INT n, INTO, eroare la împărţire, pas cu pas) au următoarele caracterisitici:

- numărul întreruperii este fie inclus în instrucţiune, fie predefinit;
- nu se execută ciclii INTA;
- nu pot fi mascate, cu excepţia întreruperii de lucru pas cu pas;
- cu excepţia întreruperii de lucru pas cu pas, sînt prioritare faţă de orice întrerupere externă.

9.1.3 Tabela vectorilor de întrerupere

Tabela vectorilor de întrerupere reprezintă legătura dintre numărul întreruperii şi procedura de tratare a întreruperii. Tabela vectorilor de întrerupere ocupă primul 1 KB din memoria sistem. Fiecare din cele 256 de întreruperi are cîte o intrare în această tabelă. Fiecare intrare în tabelă conţine un dublu cuvînt (4 baiţi). Cei mai semnificativi doi baiţi conţin adresa de bază a segmentului în care se găseşte procedura de tratare a întreruperii. Cei mai puţin semnificativi doi baiţi conţin offset-ul procedurii de tratare a întreruperii. CPU calculează adresa intrării în tabel prin înmulţirea numărului întreruperii cu 4.

Figura 9.1 prezintă structura tabelei vectorilor de întrerupere.

9.1.4 Acţiuni executate după acceptarea unei întreruperi

După acceptarea unei întreruperi şi determinarea numărului întreruperii, CPU execută cîteva acţiuni specifice, enumerate în continuare:

- Plasează în stivă registrul de indicatori;
- Execută acţiuni similare unei instrucţiuni *CALL* intersegment indirect. Adresa procedurii este conţinută de elementul aflat la adresa (numărul întreruperii $\times$ 4), în tabela vectorilor de întrerupere;
- Plasează în stivă registrele CS şi IP pentru a se putea continua programul abandonat;
- Resetează indicatorii TF şi IF;
- Înlocuieşte registrele CS şi IP cu al doilea şi primul cuvînt din elementul selectat din tabela vectorilor de întrerupere.

După executarea acestor acţiuni, se dă controlul procedurii de tratare a întreruperii. În cadrul acesteia, întreruperile externe pot fi din nou permise dacă se setează IF cu instrucţiunea STI. Sarcina salvării şi restaurării registrelor folosite în procedură revine programatorului.

Procedura trebuie să se încheie cu instrucţiunea *IRET* (Interrupt RETurn). Ca efect, se restaurează din stivă registrele IP, CS şi de indicatori, controlul revenind în programul întrerupt.

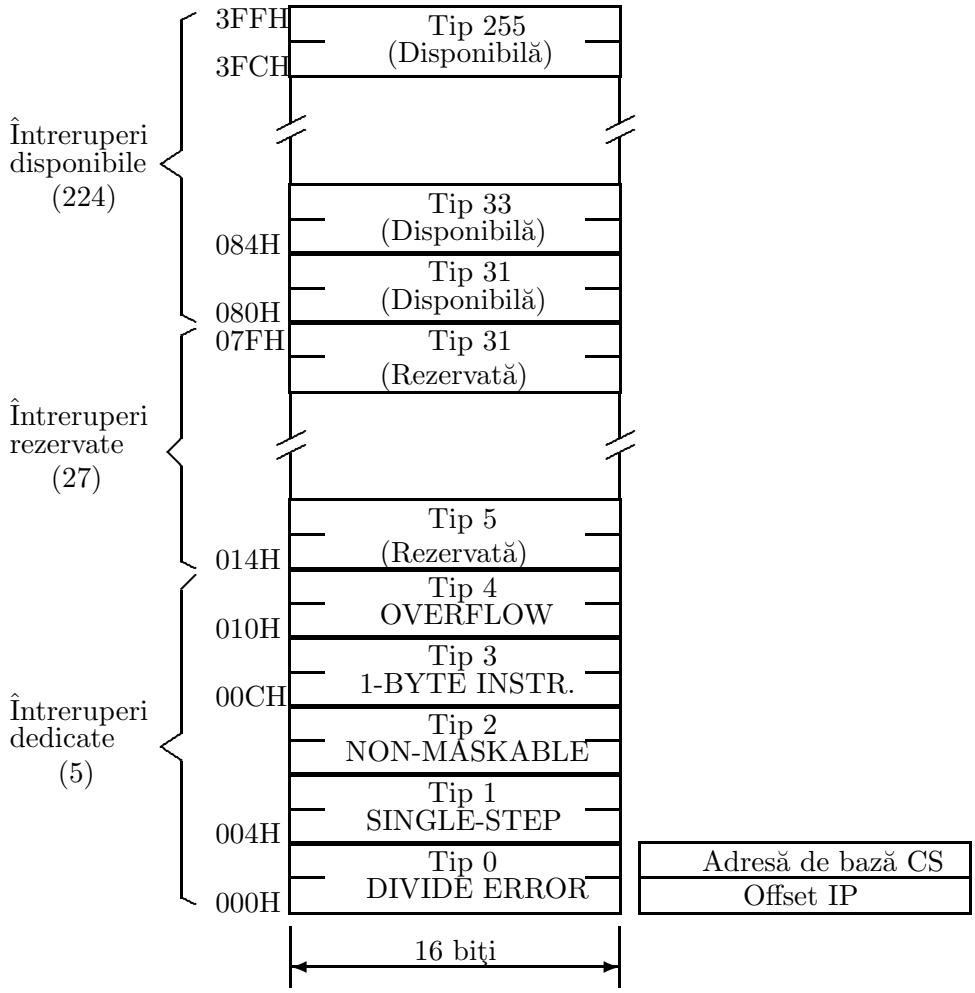

Figura 9.1: Structura tabelei vectorilor de întrerupere.

9.2 Funcţii DOS

Sistemul de operare *DOS* (Disk Operating System) pune la dispoziţia aplicaţiilor un set de rutine (funcţii) pentru gestionarea resurselor sistemului. Utilizarea acestor rutine uşurează munca programatorului permiţindu-i o viziune de nivel înalt asupra aplicaţiei dezvoltate. Mecanismul prin care o aplicaţie poate utiliza un servicu pus la dispoziţie de sistemul de operare este cel al întreruperilor software.

Funcţiile DOS pot fi apelate prin INT 21H atît din limbaj de asamblare cît şi din limbaje de nivel înalt, aşa cum este C. Pentru apelul funcţiilor DOS este rezervată întreruperea 21H. Numărul funcţiei DOS apelate este transmis ca parametru în registrul AH. Parametrii de intrare şi ieşire ai funcţiilor DOS sînt transmişi prin registre.

Pentru a apela o funcţie DOS, programul trebuie să realizeze următoarele acţiuni:

- încarcă în registrele corespunzătoare funcţiei parametrii de intrare;
- încarcă în registrul AL codul subfuncţiei (dacă este necesar);
- încarcă în registrul AH codul funcţiei;

- lansează instrucţiunea *INT 21H*.

La terminarea procedurii, programul trebuie să preia eventualii parametrii de ieşire din registrele corespunzătoare funcţiei.

Lista principalelor funcţii DOS este prezentată în tabelul 9.1

Funcţie DOS	*Denumire*
01H	Character input with echo
02H	Output character
08H	Character input without echo
09H	Output character string
25H	Set interrupt vector
2AH	Get system date
2BH	Set system date
2CH	Get system time
2DH	Set system time
31H	Terminate and stay resident
35H	Get interrupt vector
39H	Create subdirectory (MKDIR)
3AH	Remove directory entry (RMDIR)
3BH	Set directory (CHDIR)
3CH	Create file (CREAT)
3DH	Open file (OPEN)
3EH	Close file (CLOSE)
3FH	Read file (READ)
40H	Write file (WRITE)
41H	Delete file (UNLINK)
43H	Get/set file attributes
4BH	Load or execute program (EXEC)
4CH	Process terminate (EXIT)
4DH	Get return code of a subprocess
56H	Rename file
57H	Get/set file date and time
62H	Get program segment prefix (PSP) address

Tabelul 9.1: Funcţii DOS.

9.2.1 Exemplul 1: Preluare caracter de la tastatură, cu ecou/scriere caracter pe ecran

Acest exemplu prezintă un program în limbaj de asamblare care citeşte de la tastatură un număr de două cifre, cu ecou. Cele două cifre sînt interpretate ca fiind un număr reprezentat în baza 10. Pe o linie nouă, se va afişa restul împărţirii cu 9 a numărului introdus.

Pseudocodul algoritmului implementat de program este următorul:
- preia primul caracter, cu ecou, prin utilizarea funcţiei DOS 01H (character input with echo);

- converteşte primul caracter la valoarea numerică prin scăderea codului ASCII al caracterului '0';
- preia al doilea caracter, cu ecou;
- converteşte al doilea caracter la valoarea numerică;
- compune numărul introdus (prima cifră × 10 + a doua cifră);
- împarte numărul la 9 şi păstrează restul;
- converteşte numărul la codul ASCII al caracterului corespunzător prin adunarea codului ASCII al caracterului '0';
- poziţionează cursorul la începutul liniei următoare;
- afişează caracterul prin utilizarea funcţiei DOS 02H (character output);
- termină programul prin utilizarea funcţiei DOS 4CH (process terminate).

Programul face apel la trei funcţii DOS:

Funcţie DOS 01H (character input with echo)

Registre la intrare:
AH: 1

Registre la ieşire:
AL: caracter

Descriere: Aşteaptă introducerea unui caracter de la tastatură. În momentul introducerii unui caracter, codul ASCII asociat acestuia este returnat în registrul AL iar simbolul caracterului este afişat pe ecran la poziţia curentă a cursorului. Dacă tastatura generează un cod ASCII extins (cazul apăsării une taste funcţionale sau taste de control), atunci funcţia returnează 0 în registrul AL. Invocarea ulterioară a funcţiei returnează codul tastei (*scan code*) fără a mai aştepta apăsarea unei taste.

Funcţie DOS 02H (character output)

Registre la intrare:
AH: 2
DL: cod ASCII de caracter

Registre la ieşire:
nemodificate

Descriere: Conţinutul registrului DL este trimis spre dispozitivul standard de ieşire (ecran). Simbolul asociat codului ASCII al caracterului este afişat la poziţia curentă a cursorului.

Funcţie DOS 4CH (process terminate)

Registre la intrare:
AH: 4CH
AL: cod returnat

Registre la ieşire:
nemodificate

Descriere: Termină programul care a apelat funcţia şi întoarce controlul programului părinte (care a apelat programul) sau sistemului de operare. Codul returnat poate fi determinat de programul părinte prin apelul funcţiei DOS 4DH. Dacă programul a fost lansat din DOS, codul de retur este disponibil prin intermediul variabilei ERRORLEVEL, în fişiere de tip *.BAT*.

Codul sursă al programului ce rezolvă problema enunţată este prezentat în continuare.

```
code segment
  assume cs:code, ds:code
  CR equ 0AH
  LF equ 0DH

start:
  mov ax, code
  mov ds, ax      ; suprapune segmentul de date peste cel de cod
  mov bh, 0

  mov ah, 1       ; citire tastatură prin serviciu DOS 01H
  int 21h

  sub al, '0'     ; prima cifră în AL
  mov cl, 10
  mul cl          ; AL <- AL * 10
  mov bl, al      ; prima cifră în BL înmulţită cu 10

  mov ah, 1       ; citire a doua cifră prin apel DOS
  int 21h

  sub al, '0'
  add al, bl
  mov cl, 9
  mov ah, 0
  div cl
  mov bl, ah      ; restul este în BL
  add bl, '0'
  call afisare    ; afişare şir de caractere

  mov ax, 4C00h   ; terminare program (apel funcţie DOS 4CH)
  int 21h

afisare PROC      ; afişare şir de caractere prin întrerupere
  mov dl, CR
  mov ah, 2
  int 21h
  mov dl, LF
  int 21h
  mov dl, bl
  int 21h
  ret
afisare ENDP

code ENDS
END start
```

9.2.2 Exemplul 2: Citire caracter fără ecou

Acest exemplu prezintă un program care primeşte de la tastatură un şir de cuvinte pe care le afişează pe ecran. Toate cuvintele vor avea iniţiala majusculă, indiferent de modul în care a fost tastată (majusculă sau minusculă). Terminarea programului se va face la apăsarea tastei ESC.

Programul necesită prelucrarea fiecărui caracter primit de la tastatură înainte de a fi afişat pe ecran. Soluţia constă în preluarea caracterului cu funcţia DOS 8 (character input without echo), analizarea şi eventuala prelucrare a acestuia şi, ulterior, afişarea caracterului cu funcţia DOS 2 (character output).

Codul sursă al programului ce rezolvă problema enunţată este prezentat în continuare.

```
code SEGMENT
assume cs:code, ds:code
tasta_ESC equ 27

start:
  mov   ax, cs
  mov   ds, ax
  mov   bl, 1           ; indicator: BL=1 - converteşte caracterul în majusculă
                        ;            BL=0 - lasă caracterul neschimbat

caracter_in:
  mov   ah, 8
  int   21h             ; apel funcţie DOS 8 (character input without echo)
                        ; returnează în AL codul tastei apăsate
  cmp   al, tasta_ESC
  jz    final           ; s-a tastat ESC

  cmp   al, ' '
  jz    schimb_maj      ; s-a tastat ' ' (spaţiu=delimitator de cuvinte)
                        ; următorul caracter este transformat în majusculă
  cmp   bl, 0           ;
  jz    scrie_al        ; caracterul se lasă neschimbat

  mov   bl, 0           ; este o iniţială...
  cmp   al, 'a'
  jl    scrie_al        ; cu cod ASCII mai mic decît 'a'...
  cmp   al, 'z'
  jg    scrie_al        ; sau mai mare decît 'z' (în afara domeniului
                        ; minusculelor)
  sub   al, 20H         ; transformă minuscula în majusculă
  jmp   scrie_al

schimb_maj:
   mov  bl, 1

scrie_al:               ; afişează caracterul
```

```
   mov  ah, 2
   mov  dl, al
   int  21h
   jmp  caracter_in

final:
   mov  ah, 4ch
   int  21h
code ends
end start
```

9.2.3 Exemplul 3: Preluare dată sistem

Acest exemplu prezintă un program care afişează pe ecran data sistemului sub forma:

<*AA*> <*Luna*>

unde:
<*AA*> este anul (ultimele două cifre);
<*Luna*> este denumirea lunii în limba română.

Codul sursă al programului ce rezolvă problema enunţată este prezentat în continuare.

```
code SEGMENT
assume cs:code, ds:code
Denumire_luna db 'Ianuarie$  Februarie$ Martie$    Aprilie$   '
              db 'Mai$       Iunie$     Iulie$     August$    '
              db 'Septembrie$Octombrie$ Noiembrie$ Decembrie$ '

start:
  mov  ax, cs
  mov  ds, ax

  mov  ah, 2Ah          ; preia data sistem, funcţie DOS 2AH
  int  21h
                        ; preluare parametrii, prelucrare şi afişare
  sub  cx, 1900         ; CX=anul (parametru de ieşire din funcţia DOS 2AH)
  mov  ah, 0
  mov  al, cl
  mov  bl, 10
  div  bl               ; (AH:AX) / BL => AL=cît, AH = rest
                        ; AL=cifra zecilor, AH=cifra unităţilor (AN)
  mov  bx, ax
  mov  dl, bl
  add  dl, '0'
  mov  ah, 2
  int  21h              ; scrie cifra zecilor din reprezentarea anului

  mov  dl, bh
  add  dl, '0'
```

```
  mov  ah, 2
  int  21h              ; scrie cifra unităţilor din reprezentarea anului

  mov  dl, ' '
  mov  ah, 2
  int  21h              ; scrie ' '
                        ; DH=luna (parametru de ieşire din funcţia DOS 2AH)

  dec  dh               ; DH=index în tabelul Denumire_luna
  mov  al, 11           ; numărul de baiţi alocaţi în tabelul Denumire_luna
                        ; pentru o intrare (o denumire de lună)
  mul  dh               ; AL x DH = (AH:AL), index sir
  mov  dx, offset Denumire_luna
  add  dx, ax           ; DX=pointer la denumirea lunii curente, şir terminat
                        ; cu '$'

  mov  ah, 9            ; afişează şir cu funcţia DOS 9
  int  21h
  mov  ah, 4ch
  int  21h
code ends
end start
```

Programul face apel la două funcţii DOS:

Funcţie DOS 2AH (get system date)

Registre la intrare:
AH: 2AH

Registre la ieşire:
AL: ziua din săptămînă
CX: anul
DH: luna
DL: ziua

Descriere: Se returnează ziua (DL), luna (DH) şi anul (CX) datei sistem. În registrul AL se returnează codul zilei din săptămînă: 0 = duminică, 1 = luni, 2 = marţi, etc.

Funcţie DOS 9 (output character string)

Registre la intrare:
AH: 9
DX: offset adresă şir
DS: segment adresă şir

Registre la ieşire:
nemodificate

Descriere: Se afişează pe ecran şirul de caractere care începe la adresa DS:DX. Terminatorul de şir, care nu se afişează, este caracterul '$' (cod ASCII 36, 24H). Prin această funcţie nu se poate afişa caracterul '$'.

Categorie	*Întrerupere*
Servicii video	10H
Servicii tastatură	16H
Servicii disk	13H
Servicii imprimantă	17H
Servicii port comunicaţii	14H
Servicii dată/timp	1AH
Servicii unitate de bandă	15H
Servicii sistem	11H, 12H, 19H

Tabelul 9.2: Categorii de servicii BIOS.

Denumirea în limba română a lunilor anului a fost specificată sub forma unei definiţii de date de tip byte. Şirul a fost conceput astfel încît să se aloce fiecărei luni acelaşi număr de caractere (11). Adresa care trebuie transmisă ca parametru funcţiei DOS 9 este calculată după formula: *Denumire_luna + 11 × (luna curentă - 1).*

Funcţia DOS 2AH (get system date) se apelează chiar la începutul programului. Restul programului se ocupă cu prelucrarea şi afişarea parametrilor returnaţi de această funcţie. Anul este preluat în registrul CX şi este prelucrat pentru a se obţine cele două caractere asociate cifrelor zecilor şi unităţilor. Apoi este afişat un caracter spaţiu. Pe baza numărului lunii, returnat în registrul DH, se calculează adresa care trebuie transmisă funcţiei DOS 9 (output character string).

9.3 Funcţii BIOS

BIOS (Basic Input/Output System) este cel mai de jos nivel de software care interacţionează cu structura hardware a calculatorului. BIOS reprezintă un set de proceduri conţinute în memoria ROM a sistemului. Funcţiile BIOS sînt disponibile pentru a fi apelate din programe, indiferent de sistemul de operare.

Categoriile de servicii BIOS şi numerele întreruperilor asociate sînt prezentate în tabelul 9.2.

Lista principalelor funcţii BIOS este prezentată în tabelul 9.3.

9.3.1 Exemplul 4: Servicii video BIOS

Acest exemplu prezintă un program în limbaj de asamblare care citeşte de la tastatură cinci cifre cu ecou. Ulterior, programul afişează cu intermitenţă pe cea mai mică. Dacă cifra minimă apare de mai multe ori, se va afişa cu intermitenţă prima apariţie (cea mai din stînga).

Pseudocodul algoritmului implementat de program este următorul:
- citeşte (BIOS 10h, serviciul 3) şi memorează (pe stivă) poziţia cursorului la începutul programului;
- pregăteşte registre pentru codul ASCII (DL) şi coloana caracterului (DH) minim;
- citeşte cîte un caracter (DOS 1) şi, dacă este necesar, actualizează minimul;
- determină şi plasează cursorul (BIOS 10h, serviciul 2) pe poziţia caracterului minim;

Serviciu BIOS - Întrerupere	*Funcţie*	*Denumire*
Servicii video - INT 10H	00H	Set video mode
	01H	Set cursor size
	02H	Set cursor position
	03H	Read cursor position
	06H	Scroll window up
	07H	Scroll window down
	08H	Read character and attribute
	09H	Write character and attribute
	0CH	Write pixel dot
	0DH	Read pixel dot
	0EH	TTY character output
	0FH	Get current video state
	13H	Write string
Servicii tastatură - INT 16H	00H	Read keyboard character
	01H	Read keyboard status
	02H	Read keyboard shift status
Servicii disk - INT 13H	01H	Get floppy disk status
	02H	Read disk sectors
	03H	Write disk sectors
	04H	Verify disk sectors
	08H	Return drive parameters
Servicii imprimantă - INT 17H	00H	Print character
	01H	Initialize printer
	02H	Get printer status
Servicii port comunicaţii - INT 14H	00H	Initialize communication port
	01H	Transmit character
	02H	Receive character
	03H	Get communication port status
Servicii dată/timp - INT 1AH	00H	Get clock counter
	01H	Set clock counter
	06H	Set alarm
	07H	Disable alarm
	09H	Read alarm
Servicii sistem - INT 12H		Get memory size
Servicii sistem - INT 19H		Warm boot

Tabelul 9.3: Funcţii BIOS.

- afişează caracterul minim pe poziţia cursorului, cu intermitenţă (BIOS 10h, serviciul 9).

Programul face apel la trei funcţii BIOS din categoria video (INT 10h):

Serviciul 3 (read cursor position and size)

Registre la intrare:
AH: 3
BH: număr pagină video

Registre la ieşire:
BH: număr pagină video
CH: început linie cursor
CL: sfîrşit linie cursor
DH: rînd cursor
DL: coloană cursor

Descriere: Returnează caracteristicile cursorului, în funcţie de modul grafic curent.

Serviciul 2 (set cursor position)

Registre la intrare:
AH: 2
BH: număr pagină video
DH: rînd cursor
DL: coloană cursor

Registre la ieşire:
nemodificate

Descriere: Poziţionează cursorul pe ecran pe rîndul şi coloana transmise prin DH şi DL.

Serviciul 9 (write character and attribute)

Registre la intrare:
AH: 9
AL: cod ASCII caracter
BH: număr pagină video
BL: atribut video al caracterului din AL
CX: număr de caractere afişate

Registre la ieşire:
nemodificate

Descriere: Afişează unul sau mai multe caractere pe ecran. Atributul de culoare este transmis prin BL iar numărul de caractere prin CX. Poziţia cursorului nu se modifică nici dacă se afişează mai mult de un caracter. Semnificaţia biţilor registrului BL este prezentată în figura 9.2.

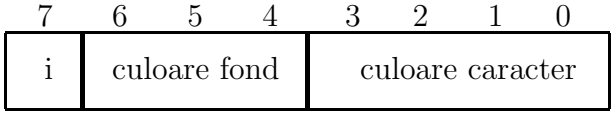

Figura 9.2: Semnificaţia biţilor registrului de atribut video BL.

Tabelul 9.4 prezintă codificarea culorilor caracterelor. Culoarea de fond este reprezentată numai pe 3 biţi. În consecinţă, sînt posibile numai primele 8 culori (culorile închise). Dacă bitul 7 (MSB) al registrului BL este 1, afişarea caracterului se va face cu intermitenţă.

Cod	*Culoare*	*Cod*	*Culoare*
00H	negru	08H	gri
01H	albastru	09H	albastru intens
02H	verde	0aH	verde intens
03H	cyan	0bH	cyan intens
04H	roşu	0cH	roşu intens
05H	magenta	0dH	magenta intens
06H	maro	0eH	galben
07H	alb	0fH	alb intens

Tabelul 9.4: Codificarea culorilor caracterelor.

Codul sursă al programului ce rezolvă problema enunţată este prezentat în continuare.

```
code SEGMENT
assume cs:code, ds:code

start:
  mov  ax, cs
  mov  ds, ax

  mov  ah, 3
  mov  bh, 0
  int  10h          ; citeşte poziţia şi dimensiunea cursorului
                    ; (funcţie BIOS 10h, serviciu 3)

  push dx           ; poziţie cursor la intrare în program
                    ; DH - rînd, DL - coloană
  mov  cx, 5        ; contor număr de caractere citite
  mov  dl, '9'      ; caracterul cel mai mare posibil
  mov  ah, 1        ; DL va conţine codul ASCII al caracterul minim
                    ; DH va conţine coloana caracterului minim

caracter_in:
  int  21h          ; DOS 1 (citeşte caracter în AL)
  cmp  dl, al
  jg   schimb       ; DL>AL implică înlocuire caracter minim din DL

intorc:
  dec  cx           ; actualizează contor caractere
  jnz  caracter_in
  jmp  afisare      ; au intrat 5 caractere, se afişează minimul
```

```
schimb:
  mov  dl, al
  mov  dh, 5
  sub  dh, cl
  jmp  intorc

afisare:
  ; DL conţine codul ASCII al caracterului minim
  ; DH conţine coloana caracterului minim

  ; pregăteşte parametrii de intrare pentru întrerupere BIOS 10h, serviciu 2
  ; (set cursor position)
  pop  cx           ; CX <= poziţie cursor la intrare în program
  push dx           ; pune în stivă DL, DH
  push cx           ; repune în stivă poziţia iniţială a cursorului
  mov  ah, 2        ; număr serviciu BIOS video
  mov  bh, 0        ; număr pagină video
  xor  cx, cx       ; CX=(CH:CL) <= 0
  mov  cl, dh       ; CL <= coloana caracterului minim
  pop  dx           ; DX <= poziţie cursor la intrare în program
  add  dx, cx       ; DH=rînd şi DL=coloană caracter minim
  int  10h          ; apelul întreruperii software

  ; pregăteşte parametrii de intrare pentru întrerupere BIOS 10h, serviciu 9
  ; (write character and attribute)
  mov  ah, 9        ; număr serviciu BIOS video
  pop  dx
  mov  al, dl       ; cod ASCII caracter
  mov  bh, 0        ; număr pagină video
  mov  bl, 128+7    ; atributul caracterului (alb pe fond negru, cu intermitenţă)
  mov  cx, 1        ; numărul de caractere
  int  10h          ; apelul întreruperii software

  mov  ah, 4ch
  int  21h
code ends
end start
```

9.4 Experimente

I. Studiaţi programul *CICLU_A.ASM*. Programul şterge ecranul (BIOS video, int 10h, serviciul 6), şi citeşte ciclic un caracter de la tastatură prin funcţii BIOS, int 16h, serviciul 0. În cazul în care se apasă tasta 'A' (caracter 'A' sau 'a'), programul afişează un mesaj în centrul ecranului.

II. Rezolvarea propusă de exemplul 3 nu este "compatibilă cu anul 2000". Modificaţi programul astfel încît să realizaţi compatibilitatea. Estimaţi costul operaţiei considerînd următoarele metrici:

- număr de linii de cod modificate;
- număr de iteraţii (asamblare, link-editare, depanare) necesare;
- număr de variante testate prin simulare;
- timp de execuţie.

Considerînd costul manoperei şi al amortizării calculatorului folosit, estimaţi costul pe care ar trebui să-l plătească beneficiarul.

III. Folosind bibliografia şi programul de documentare hipertext $TECHhelp$, studiaţi parametrii de intrare, parametrii de ieşire şi acţiunile funcţiilor DOS specificate în tabelul 9.1 şi a funcţiilor BIOS specificate în tabelul 9.3.

Lucrarea 10

Noţiuni avansate de programare în limbaj de asamblare

Această lucrare prezintă modul de utilizare a limbajului de asamblare în sistem de operare DOS pentru:

- Accesarea fişierelor prin funcţii DOS;
- Scrierea programelor rezidente.

10.1 Accesarea fişierelor din limbaj de asamblare

Deşi limbajele de nivel înalt pun la dispoziţia programatorului un set larg de funcţii pentru accesarea fişierelor, uneori este nevoie de a accesa fişiere direct din limbaj de asamblare. Pentru aceasta, sistemul de operare DOS furnizează funcţii sistem dedicate, apelabile prin întreruperi software.

10.1.1 Formate de fişiere executabile

DOS este un sistem de operare capabil să încarce în memorie şi să execute două tipuri de fişiere program, avînd extensiile *.COM* şi *.EXE*. Ambele tipuri de programe sînt relocatabile (pot fi încărcate la orice adresă fizică). Acest lucru este asigurat prin mecanismul de segmentare a memoriei şi prin faptul că toate salturile în program sînt realizate cu instrucţiuni de salt relativ.

Fişierul în formatul COM conţine o imagine binară a codului şi datelor programului. Se poate considera că programele *COM* sînt scrise într-un model de memorie *"tiny"*, conform tabelului 8.2. Codul, datele şi stiva împreună nu pot avea o dimensiune mai mare de un segment (64 KB).

Fişierul în formatul EXE conţine, în plus faţă de fişierul în format *COM*, un antet care informează sistemul de operare despre modul de gestiune a segmentelor.

Înainte de a încărca un program în memorie, DOS selectează o adresă de segment la care va încărca programul. Ca adresă de bază, DOS alege totdeauna cea mai mică adresă de memorie

liberă. Indiferent de formatul fişierului (*COM* sau *EXE*), la începutul zonei de memorie rezervate pentru program se încarcă un prefix de 256 de baiţi numit *PSP* (*Program Segment Prefix*). PSP conţine informaţii pe care sistemul de operare le pune la dispoziţia programului. Structura PSP-ului este prezentată în tabelul 10.1.

După ce controlul este predat programului încărcat în memorie, registrele *DS* şi *ES* conţin adresa PSP. Informaţia din PSP poate fi utilizată în cadrul programului pentru a prelua argumentele din linia de comandă, pentru a determina cîtă memorie este disponibilă programului sau a pentru a accesa variabilele de sistem. Adresa PSP poate fi accesată ulterior prin funcţiile DOS 62*H* (*query current PSP*).

Offset	*Dimensiune*	*Semnificaţie*
0	2	Codul instrucţiunii `INT 20H` (`CD 20`). Printr-o instrucţiune de salt la această adresă se poate termina programul.
2	2	Adresa de sfîrşit a memoriei alocate programului, exprimată în blocuri de 16 baiţi.
4	1	Rezervat.
5	5	Codul instrucţiunii de apel de procedură de tip FAR spre punctul de intrare al funcţiilor DOS.
0AH	4	Adresă de revenire în sistemul de operare la execuţia instrucţiunilor `INT 22H` sau `INT 21H` funcţia `4CH`.
0EH	4	Adresă de tratare a întreruperii `INT 23H` (Ctrl-Break).
12H	4	Adresă de tratare a întreruperii `INT 24H` (erori critice DOS).
16H	22	Rezervat.
2CH	2	Adresă de segment a variabilelor DOS.
2EH	46	Rezervat.
5CH	16	FCB pentru primul parametru.
6CH	20	FCB pentru al doilea parametru.
80H	1	Numărul de baiţi din linia de comandă (fără numele programului).
81H	127	Linia de comandă a programului (fără numele programului şi fără directive de redirectare).

Tabelul 10.1: Structura PSP.

Imediat după încărcarea unui program *EXE* în memorie:

- registrele *DS* şi *ES* conţin adresa PSP;
- registrele *CS*, *IP*, *SS* şi *SP* conţin valorile indicate în antetul fişierului *EXE*;
- cîmpul al doilea din PSP (adresa sfîrşitului memoriei disponibile) conţine valoarea din antetul fişierului *EXE*.

Imediat după încărcarea unui program *COM* în memorie:

- registrele *CS*, *DS*, *ES* şi *SS* conţin adresa PSP;
- registrul *SS* conţine adresa de sfîrşit a segmentului (de obicei `0FFFEH`);

- registrul IP este iniţializat cu valoarea `0100H`.

Utilizarea formatului COM presupune respectarea următoarelor constrîngeri:

- Programul trebuie să fie format dintr-un singur segment;
- Codul sursă trebuie să înceapă cu o pseudoinstrucţiune `ORG 100H`. Programul trebuie să înceapă cu o instrucţiune executabilă a cărei etichetă trebuie să apară în pseudoinstrucţiunea `END`, care încheie fizic textul programului sursă. Această instrucţiune este plasată la adresa 100H şi poate fi o instrucţiune de salt la adresa la care începe efectiv programul;
- Datele pot fi plasate oriunde în codul sursă dar este de preferat să se plaseze la începutul acestuia deoarece asamblorul poate semnala o eroare în cazul unor referinţe la date nedeclarate încă;
- Nu este necesară iniţializarea registrelor segment, toate avînd aceeaşi valoare ca şi registrul CS;
- Stiva este iniţializată automat la sfîrşitul segmentului ($SS = CS$ iar $SP =$ `0FFFE0H`);
- Încheierea execuţiei programului se face prin execuţia secvenţei de instrucţiuni:
  ```
  mov  ax, 4c00H
  int 21H
  ```

Formatul unui fişier sursă în limbaj de asamblare pentru generarea unui fişier executabil în format COM este prezentat în continuare. Pentru a genera acest format de fişier executabil, link-editorul trebuie lansat cu opţiunea */t*.

```
code  SEGMENT
assume cs:code, ds:code
org  100H

start:
  jmp  inceput

;
; Definiţii de date
;

data1 db  'Sir de caractere'   ; exemple
a1    dw  10

;
; Codul programului
;

start:
  ...                          ; instrucţiuni în limbaj de asamblare
  ...
```

```
;
; Sfîrşit execuţie program
;

  mov  ax, 4c00H
  int  21H

code  ENDS

end start
```

10.1.2 Funcţii DOS pentru lucru cu fişiere

Începînd cu versiunea 2.0, sistemul de operare DOS pune la dispoziţia programelor utilizatorului cîteva funcţii apelabile prin întreruperea software $21H$, similare cu funcţiile UNIX. Prin aceste funcţii se pot realiza atît operaţii elementare asupra fişierelor (citire, scriere) cît şi gestionarea acestora (creare, deschidere, închidere, ştergere).

Un fişier pe disc este identificat pe baza numelui său şi a directorului în care se află. Numele fişierului este transmis ca parametru funcţiilor DOS sub forma unui pointer la un şir de caractere aflate în memorie. În cadrul unui program, un fişier este identificat pe baza unui *identificator de fişier* (*file handle*, în limba engleză). Identificatorul fişierului este un număr de 16 biţi returnat de funcţiile de creare sau deschidere de fişiere. Operaţiile ulterioare de accesare a fişierului (căutare, citire, scriere) se fac pe baza identificatorului de fişier. Funcţiile DOS returnează identificatorul unui fişier în registrul de 16 biţi BX.

Există cîţiva identificatori rezervaţi de către sistemul de operare DOS pentru anumite dispozitive din calculator. În acest fel, aceste dispozitive pot fi accesate cu aceleaşi funcţii ca şi fişierele. Tabelul 10.2 prezintă identificatorii rezervaţi pentru dispozitivele standard dintr-un calculator PC. Identificatorii standard sînt automat iniţializaţi la începutul programului.

Identificator	*Descriere*
0000H	Dispozitiv standard de intrare (tastatură)
0001H	Dispozitiv standard de ieşire (monitor)
0002H	Dispozitiv standard pentru afişarea erorilor (monitor)
0003H	Dispozitiv standard auxiliar (COM1)
0004H	Imprimantă standard (LPT1)

Tabelul 10.2: Identificatorii rezervaţi pentru dispozitivele standard dintr-un calculator PC.

Tabelul 10.3 prezintă funcţiile DOS din categoria *serviciilor de disc*. Toate funcţiile returnează în registrul AX un cod de eroare asociat operaţiei realizate de funcţie.

Funcţie DOS	*Denumire funcţie*
3CH	Creare fişier
3DH	Deschidere fişier
3EH	Închidere fişier
3FH	Citire din fişier
40H	Scriere în fişier
41H	Ştergere fişier
5BH	Creare fişier
44H	Controlul dispozitivului I/O

Tabelul 10.3: Funcţii DOS din categoria serviciilor de disc.

10.1.3 Exemplul 1: Program în limbaj de asamblare pentru copierea unui fişier

Acest exemplu prezintă un program pentru copierea unui fişier sub un alt nume. Dacă există deja un fişier cu numele celui de destinaţie, copierea se abandonează şi se returnează un mesaj de atenţionare.

Pseudocodul programului este prezentat în continuare.
- Identifică argumentele liniei de comandă, aflate în PSP la offset 81h;
- În şirul de argumente, caută delimitatorii (caractere spaţiu ' ') şi identifică argumentele;
- Dacă linia de comandă are două argumentele, continuă. Altfel, termină programul cu afişarea mesajului cu sintaxa liniei de comandă:

```
Sintaxă: copiere <sursă> <destinaţie>
```

- Deschide fişierul sursă în citire:

```
mov ax, 3d00h
mov dx, offset cale_sursa
int 21h
```

- Dacă operaţia anterioară nu a generat eroare, continuă. Altfel, termină programul cu afişarea mesajului:

```
Eroare deschidere fişier sursă
```

- Salvează identificatorul fişierului sursă;
- Crează fişierul destinaţie:

```
mov ah, 5bh
mov cx, 0
int 21h
```

- Dacă operaţia anterioară nu a generat eroare, continuă. Altfel, înseamnă că numele fişierului destinaţie nu este valid sau există un fişier cu acelaşi nume. În acest caz, termină programul cu afişarea mesajului:

```
Eroare deschidere fişier destinaţie
```

- Salvează identificatorul fişierului destinaţie;
- Citeşte cîte un buffer din fişierul sursă şi scrie bufferul în fişierul destinaţie. Cînd ultimul buffer nu este plin, înseamnă că s-a terminat fişierul sursă:

```
copiere:
```

```
  mov ah, 3fh                 ; citire din fişier <sursă>
  mov bx, [handle1]
  mov dx, offset buff_rw      ; adresa bufferului unde se vor depune octeţii
  mov cx, 1024                ; CX conţine numărul de octeţi citiţi
  int 21h

  cmp ax, cx                  ; cînd numărul de octeţi citiţi este mai mic
  je buffer_intreg            ; decît capacitatea bufferului, va fi ultima
  mov [cond], 1               ; citire
buffer_intreg:

  mov cx, ax
  mov ah, 40h                 ; scriere în fişier <destinaţie>
  mov bx, [handle2]
  int 21h

  cmp [cond], 1
  je inchidere
  jmp copiere
```

- Închidere fişiere sursă şi destinaţie:

```
inchidere:
  mov ah, 3eh
  mov bx, [handle1]
  int 21h
  mov ah, 3eh
  mov bx, [handle2]
  int 21h
```

10.2 Programe rezidente

Sistemul de operare DOS permite unui program să rămînă în memorie şi să predea controlul programului ce l-a lansat (programului părinte) sau sistemului de operare. Programul rămas rezident poate fi reactivat de către un eveniment extern (apăsarea unei taste - *INT 09H*, sau orice întrerupere software sau hardware). În acest mod, deşi la un moment dat se execută un singur porgram, prin comutarea rapidă între programe, se poate crea impresia de 'multitasking'. Terminarea unui program care să rămînă rezident în memorie se face prin apelul funcţiei DOS $31H$. Apelul funcţiei de rămînere rezident trebuie precedat de stabilirea condiţiilor de reactivare. Condiţiile de reactivare sînt stabilite prin modificarea ('deturnarea') unor întreruperi. Programele de acest fel sînt cunoscute ca programe TSR (*Terminate and Stay Resident*).

Descrierea funcţiei DOS 31H este prezentată în tabelul 10.4.

Structura generală a programelor TSR este prezentată în continuare.

- Se verifică dacă programul lansat este deja rezident: se verifică o locaţie de memorie aflată la o adresă fixă, stabilită prin programul TSR;

Registre la intrare	*Descriere*
AH: 31H AL: cod returnat DX: dimensiune memorie rezervată	Termină programul dar îl păstrează rezident în memorie. Codul de retur din AL este întors programului părinte sau sistemului de operare DOS prin variabila ERRORLEVEL. Codul returnat poate fi determinat cu funcţia DOS 4DH.

Tabelul 10.4: Descrierea funcţiei DOS 31H (terminare program prin rămînere rezident în memorie).

- Dacă programul nu este deja rezident (este la prima rulare) atunci se pregăteşte rămînerea rezidentă:
  - Se deturnează întreruperile folosite la reactivare (tastatură 09H, hard-disk 13H, etc.);
  - Se eliberează memoria alocată programului TSR (funcţie DOS 49H);
  - Se termină programul cu rămînere rezidentă în memorie prin apelul funcţiei DOS 31H (*Terminate and Stay Resident*).
- Dacă programul este deja rezident (nu este la prima rulare) se realizează următoarele acţiuni:
  - Se verifică dacă este o comandă de dezinstalare prin evaluarea argumentelor liniei de comandă care se obţine din PSP (adresa de intrare în program ES:0081);
  - Dacă este o comandă de dezinstalare, se refac vectorii întreruperilor deturnate.
- Se termină programul prin funcţia DOS 4CH.

10.3 Experimente şi întrebări

I. Care sînt diferenţele între cele două formatele de fişiere executabile în sistem de operare DOS (EXE şi COM)?

II. Enumeraţi acţiunile care au loc de la lansarea în execuţie a unui program (după tastarea numelui programului la linia de comandă) şi pînă la începerea execuţiei efective a acestuia.

III. Utilizînd *Turbo Debugger*, verificaţi conţinutul registrelor la începerea programului lansat (format EXE şi COM).

IV. Folosind programul de documentare hipertext $TECHhelp$, studiaţi antetul fişierelor EXE (*DOS kernel/EXE File Header Layout*). Folosind un editor de texte hexa (*nc.exe*, $F3$, $F4$) vizualizaţi antetul unui fişier executabil. Ulterior, folosind *Turbo Debugger*, vizualizaţi PSP-ul aceluiaşi program, după ce a fost încărcat în memorie. Faceţi corespondenţa între datele existente în antetul fişierului (date statice) şi cele existente în PSP (date dinamice).

V. Folosind programul de documentare hipertext $TECHhelp$ realizaţi un tabel cu descrierea funcţiilor DOS pentru lucru cu fişiere. Pentru fiecare din funcţiile prezentate în tabelul 10.3, realizaţi un tabel similar celui din figura 10.5, corespunzător funcţiei 3FH (citire din fişier).

Registre		*Descriere*
intrare	*ieşire*	
AX: 3FH BX: identificator fişier CX: nr. baiţi citiţi DX: offset buffer DS: segment buffer	AX: cod eroare	Citeşte informaţie dintr-un fişier. Identificatorul de fişier este în BX, iar bufferul în care se trimit datele citite este la adresa DS:DX. CX conţine numărul de baiţi care se citesc din fişier. Prima citire se face de la începutul fişierului. Citirile ulterioare se fac din poziţia de unde s-a terminat citirea anterioară. Dacă a apărut o eroare la citire se setează CF iar în registrul AX se returnează codul de eroare. Dacă citirea s-a făcut fără eroare, CF este resetat iar în registrul AX se returnează numărul de baiţi citiţi.

Tabelul 10.5: Descrierea funcţiei DOS 3FH, citire din fişier.

VI. Un fişier poate fi creat fie cu funcţia DOS 3CH, fie cu funcţia DOS 5BH. Care este diferenţa între cele două funcţii DOS?

VII. Studiaţi fişierul *COPIERE.ASM*, ce conţine un program în limbaj de asamblare pentru copierea unui fişier. Testaţi programul cu diferite tipuri de argumente şi aduceţi-i îmbunătăţiri.

VIII. Studiaţi fişierul *ALTR.ASM*, ce conţine un program în limbaj de asamblare capabil să rămînă rezident în memorie. Programul se reactivează la apăsarea combinaţiei de taste 'Alt-R'. La reactivare, programul scrie un mesaj pe ecran. Comanda de dezinstalare a programului rezident este `altr /u`. Modificaţi programul pentru a se reactiva şi la alte combinaţii de taste şi pentru a avea o interfaţă mai 'prietenoasă'. De exemplu, sintaxa linie de comandă să fie:

altr [/u][/h]

Dacă se lansează programul cu opţiunea */h*, apar pe ecran informaţii care descriu sintaxa liniei de comandă.

Lucrarea 11

Teme de programare în limbaj de asamblare

Această lucrare prezintă o parte din temele propuse la colocviul de laborator, în perioada 1995-1998, de Catedra de Electronică şi Calculatoare.

I. Scrieţi o funcţie care să caute un caracter într-un şir de caractere. Funcţia returnează numărul primei poziţii a caracterului în şir. Dacă respectivul caracter nu apare în şir, funcţia returnează 0. Modul de transmitere al parametrilor este la latitudinea programatorului.

II. Scrieţi un program care să calculeze produsul scalar a doi vectori. Vectorii se află în segmentul de date, au dimensiuni egale şi elemente de tip byte. Rezultatul va fi un cuvînt.

III. Scrieţi o funcţie care transformă minusculele în majuscule lăsînd orice alt caracter neschimbat. Se operează pe un şir ASCII ('A' - 'Z', 41h - 51h, 'a' - 'z', 61h - 7Ah).

IV. Scrieţi o funcţie care să aibe ca parametru de intrare un pointer la un şir de caractere terminat cu 0. Funcţia returnează un pointer la alt şir, cu caracterele inversate. Toate caracterele de control (cod ASCII $< 32h$) se vor înlocui cu constanta $20h$.

V. Scrieţi un program care preia de la tastatură două şiruri de maxim 10 caractere terminate cu caracterul CR (Carrige Return, cod ASCII 0DH) şi le compară. În caz că sînt identice, se afişează pe ecran mesajul 'ADEVARAT', altfel se afişează mesajul 'FALS'. Primul şir, al doilea şir şi mesajul vor apare pe linii diferite, aşa ca în exemplele următoare:

```
Exemplul 1        Exemplul 2        Exemplul 3
----------        ----------        ----------
abdf              abcde             abc
cdfrer            abcde             abcde
FALS              ADEVARAT          FALS
```

VI. Scrieţi un program care să primească de la tastatură două numere binare reprezentate pe cîte 8 biţi şi să afişeze rezultatul produsului lor sub forma:
`00001100*00000010=0000000000011000`

VII. Scrieţi un program care să citească de la tastatură trei litere majuscule $\{A, B, ..., Z\}$, să scrie pe ecran un spaţiu (cod ASCII 20H) şi apoi să le scrie în ordine alfabetică. Păstraţi cele trei litere tastate în registrele BH, BL şi CL.

VIII. Scrieţi un program care să utilizeze funcţii DOS şi să funcţioneze după cum urmează. Utilizatorul tastează două litere. Dacă una dintre ele este 'D', programul afişează în continuare cuvîntul DA urmat de o linie nouă. Altfel, este afişat cuvîntul NU, urmat de o linie nouă. Secvenţa de evenimente se repetă de două ori, după care controlul este întors în DOS.

IX. Scrieţi un program care să primească de la tastatură un număr N, între 1 şi 9. Programul va afişa pe ecran un pătrat cu latura N, umplut cu caractere '*', colţul stînga sus fiind pe rîndul 10, coloana 10.

X. Scrieţi un program care să şteargă ecranul şi să afişeze în partea din dreapta sus ora sub forma:

```
HH:MM
```

XI. Scrieţi un program care să afişeze pe ecran unul din mesajele de salut care urmează, în funcţie de ora sistem:

`Buna dimineata!`, între orele 5:00 şi 10:00,
`Buna ziua!`, între orele 10:00 şi 20:00,
`Noapte buna!`, între orele 20:00 şi 5:00.

De exemplu, la ora 7:28p (oră furnizată de comanda DOS `time`), pe ecran va apare salutul:

```
Buna ziua!
```

XII. Scrieţi un program care să şteargă ecranul şi să afişeze în partea de stînga jos ora sistem sub forma:

```
Ora: HH
Minutul: MM
```

XIII. Pentru fiecare din următoarele perechi de adrese date sub forma *segment:offset*, decideţi dacă cele două adrese corespund aceleiaşi locaţii de memorie:

a) 1234:1234 şi 1358:0040

b) 0500:ABCD şi 0EB0:10CD

XIV. Scrieţi un program care să citescă de la tastatură un număr de două cifre şi apoi să afişeze pe display, pe o linie nouă, restul împărţirii cu 9 a numărului introdus.

XV. Scrieţi un program care la apăsarea unei taste să afişeze pe ecran semnificaţia tastei, urmată de un spaţiu şi de codul ASCII al tastei apăsate (exprimat în baza zece). Programul se va termina la apăsarea tastei ESC.

Exemplu:

```
A 65
O 79
g 103
5 53
H 72
```

XVI. Scrieţi un program care să preia de la tastatură un număr de o cifră (1 - 9) şi să afişeze pe ecran toţi divizorii acestui număr (despărţiţi de virgule). Terminarea programului se face dacă se tastează 0. În cazul tastării altui caracter, în afara celor de la 0 la 9, programul nu reacţionează.

Exemplu:

```
4 1,2,4
a
7 1,7
8 1,2,4,8
0
```

XVII. Scrieţi un program care să citescă de la tastatură un număr de două cifre şi apoi să afişeze pe ecran, în continuarea cifrelor, clasa de resturi modulo 5 căreia îi aparţine numărul. Terminarea programului se face tastînd 00.

Exemplu:

```
47 = 2 mod 5
23 = 3 mod 5
16 = 1 mod 5
15 = 0 mod 5
00
```

XVIII. Scrieţi un program care să afişeze pe ecran data sistemului sub forma:

`ZZ Luna`,

unde:

`ZZ` este numărul zilei (pe două cifre) iar
`Luna` este denumirea lunii în limba română.

De exemplu, în data de 05-13-1999 (dată furnizata de comanda DOS `date`), pe ecran va apare:

```
13 Mai
```

XIX. Scrieţi un program care să primească de la tastatură un număr reprezentat pe două cifre în baza zece. Programul va afişa pe ecran, în continuarea numărului tastat, caracterele '`->`' urmate de reprezentarea numărului în baza 16.

De exemplu, dacă se tastează `3` şi `5`, pe ecran va apare:

```
35->23
```

XX. Scrieţi un program care să primească de la tastatură un număr reprezentat pe două cifre în baza 16. Programul va afişa pe ecran, în continuarea numărului tastat, caracterele '`=>`' urmate de reprezentarea numărului în baza 10.

De exemplu, dacă se tastează `1` şi `A`, pe ecran va apare:

```
1A=>26
```

Partea III

Anexe

Anexa A

Utilizarea Turbo Debugger

Această anexă prezintă utilitarul *Turbo Debugger* al firmei *Borland*. Interfaţa grafică este aceeaşi atît pentru DOS cît şi pentru Windows. *Turbo Debugger* este un dezasamblor de programe folosit pentru depanarea şi depistarea erorilor acestora.

A.1 Descrierea interfeţei grafice

Interfaţa grafică a *Turbo Debugger* este compusă din:

- *meniului principal* - aflat pe linia superioră a ecranului;
- *meniul general sau local* - aflat pe linia inferioară ecranului;
- *ferestre* dedicate afişării diferitelor date utilizate pentru depanarea programelor.

Fereastra principală utilizată pentru depanarea codului este fereastra CPU. Această fereastră este împărţită în cinci subferestre în care sînt prezentate următoarele date:

- segmentul de cod cu instrucţiuni dezasamblate;
- setul de registre;
- indicatorii;
- stiva;
- zona de memorie din segmentul de date.

Figura A.1 prezintă interfaţa grafică a programului *Turbo Debugger*, cu fereastra CPU maximizată.

A.2 Meniul principal

Meniul principal se află în partea superioră a ferestrei şi poate fi accesat în două moduri:

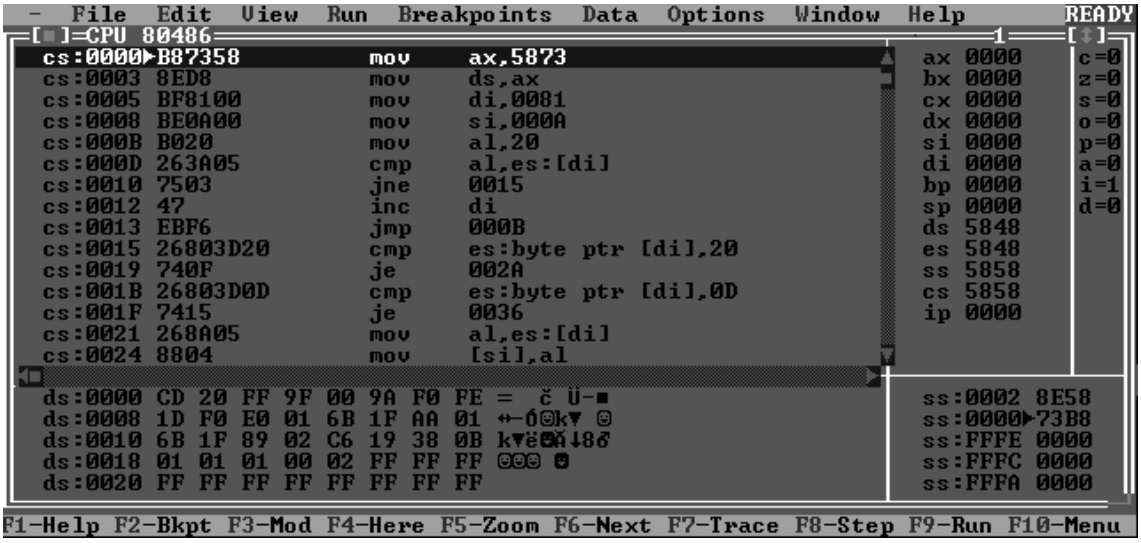

Figura A.1: Interfaţa grafică a *Turbo Debugger*.

- apăsarea tastei funcţionale F10;
- apăsarea tastei *ALT* simultan cu prima literă a unui meniu (F - *File*, E - *Edit*, V - *View*, R - *Run*, B - *Breakpoints*, D - *Data*, O - *Options*, W - *Window*, H - *Help*).

A.2.1 File

Meniul *File*, obţinut prin tastarea $ALT - F$, conţine opţiunile ce permit utilizatorului operaţii externe dezasamblorului. Cînteva dintre acestea sînt:

Open - comanda permite deschiderea un fişier .exe, .com sau .dll pentru a fi dezasamblat. În funcţie de opţiunea de asamblare (tasm /zi şi tlink/v), codul dezasamblat poate fi însoţit de informaţii de depanare complete sau nu. Programul depanat poate fi specificat şi ca argument în linia de lansare a programului *TD*.

DOS Shell - comanda permite suspendarea temporară a *TurboDebugger* pentru lansarea unor comenzi DOS. Revenirea se face prin tastarea la prompterul sistem a comenzii "exit".

Resident - comanda termină *TD* cu rămînerea sa rezidentă. Această comandă se utilizează pentru depanarea programelor *TSR*.

Quit ($ALT - X$) - comanda termină programul *TD* şi întoarce controlul în DOS.

A.2.2 Edit

Meniul *Edit*, obţinut prin tastarea $ALT - E$, permite comenzi de copiere a unor articole prin intermediul *clipboard*.

A.2.3 View

Meniul *View*, obţinut prin tastarea $ALT - V$, conţine comenzi pentru afişarea separată a tuturor tipurilor de ferestre. Cîteva din comenzile meniului sînt prezentate în continuare.

Breakpoints (F2) - comandă de deschidere a ferestrei cu descrierea punctelor de oprire în program. Fereastra prezintă o listă a adreselor de oprire şi a condiţiilor asociate acestora.

Stack - comandă de deschidere a ferestrei care prezintă starea actuală a stivei, adresele de întoarcere ale instrucţiunilor CALL precum şi variabile memorate în stivă. Informaţia din această fereastră este disponibilă şi într-o subfereastră a ferestrei CPU.

Watches - comandă de deschidere ferestrei în care se poate urmări modificarea unei variabile sau expresii în timpul rulării unei secvenţe de cod. Prin comanda *Data/Add Watch*, variabila sau expresia este adăugată în fereastra $Watches$ de la poziţia cursorului sau prin specificare explicită.

Variable - comandă de deschidere a ferestrei în care se prezintă o listă cu simboluri şi valorile asociate lor în modulul de cod curent. Se pot prezenta şi simboluri globale şi valorile lor asociate.

Module... (F3) - comandă de deschidere a unei ferestre cu codul sursă al modulului curent. Fereastra $Module$ este fereastra deschisă implicit la lansarea în execuţie a TD, în cazul în care programul depanat include informaţii suplimentare de depanare.

File... - comandă de deschidere a unei ferestre unde se poate încărca un program. Conţinutul programului poate fi afişat în hexazecimal sau ASCII, folosind comanda $Ctrl - D$ (Display).

CPU - comandă de deschidere a ferestrei CPU, compusă din cele cinci subferestre prezentate în paragraful A.1. În fereastra principală este afişat codul dezasamblat al programului. Datele sînt structurate pe trei coloane:

- coloana cu adresa sub forma *segment:offset*, exprimate în baza 16;
- coloana cu codurile instrucţiunilor, exprimate în baza 16; numărul de octeţi din această coloană este dependent de instrucţiunea respectivă;
- coloana cu instrucţiunile în limbaj de asamblare (mnemonici şi operanzi).

Celelalte subferestre sînt similare cu ferestrele $Registers$, $Dump$ şi $Stack$. Fereastra CPU este fereastra deschisă implicit la lansarea în execuţie a TD, în cazul în care programul depanat nu conţine informaţii suplimentare de depanare.

Dump - comandă de deschidere a ferestrei în care se afişează conţinutul unei zone de memorie. Formatul de reprezentare implicit este în hexazecimal. Modificarea formatului se face prin comanda $Ctrl - D$ (Display), sau din meniul ce apare prin apăsarea butonului din dreapta a mouse-ului. Modificarea conţinutul se face direct prin scrierea noii valori la adresa dorită, sau prin comanda $Ctrl - C$ (Change). Informaţia din această fereastră este disponibilă şi într-o subfereastră a ferestrei CPU.

Registers - comandă de deschidere a ferestrei de regiştrii şi indicatori. Modificarea se face direct prin scrierea noii valori în registrul selectat sau prin comanda $Ctrl - C$ (Change). Modificarea indicatorilor se face prin tasta ENTER sau cu comanda $Ctrl - T$ (Toggle). Informaţia din această fereastră este disponibilă şi într-o subfereastră a ferestrei CPU.

A.2.4 Run

Meniul Run, obţinut prin tastarea $Alt - R$, conţine comenzi pentru execuţia programului.

Run (F9) - comandă ce determină execuţia continuă a programului pînă la întîlnirea unui punct de oprire, pînă la întreruperea de către utilizator prin tastarea $CTRL - Break$ sau pînă la teminarea acestuia.

Go to cursor (F4) - comandă ce determină execuţia programului pînă la atingerea instrucţiunii pe care se află cursorul.

Trace Into (F7) - comandă ce determină execuţia unei singure instrucţiuni. Dacă instrucţiunea este de apel de procedură (CALL), atunci execuţia se opreşte la prima instrucţiune din procedură.

Step over (F8) - comandă ce determină execuţia unei singure instrucţiuni. Dacă instrucţiunea este de apel de procedură (CALL), se execută întreaga procedură şi execuţia se opreşte la instrucţiunea din programul principal, care urmează instrucţiunii CALL.

Execute to... (Alt-F9) - comandă ce determină execuţia programului pînă la o adresă specificată. Specificarea adresei poate face printr-o constantă, la care TD adaugă valoarea de segment, sau o expresie ce reprezintă o locaţie de memorie.

Until Return (Alt-F8) - comandă ce determină execuţia programului pînă cînd funcţia sau procedura curentă se reîntoarce în programul care a apelat-o.

Back trace (Alt-F4) - comandă ce determină anularea rezultatului ultimei instrucţiuni executate. Starea procesorului şi a memorie este refăcută la valoarea anterioară execuţiei instrucţiunii.

Instruction trace (Alt-F7) - comandă ce determină execuţia unei singure instrucţiuni maşină în cazul depanării unui program scris într-un limbaj de nivel înalt. Se foloseşte pentru a urmări o procedură de tratare a întreruperii sau o funcţie într-un modul compilat fară includerea informaţiilor pentru depanare.

Arguments... - comandă ce permite setarea sau modificarea argumentelor liniei de comandă a programului depanat.

Program reset (Ctrl-F2) - comandă ce determină reîncărcarea programului iniţial de pe disc, cu reiniţializarea procesorului.

A.2.5 Breakpoints

Meniul *Breakpoints*, obţinut prin tastarea $ALT - B$, conţine comenzi pentru plasarea, setarea sau anularea punctelor de oprire în programul depanat.

Toggle (F2) - comandă ce setează sau anulează un punct de oprire la o adresă sau linie de cod indicată de cursor. Programul se va opri de fiecare dată cînd va ajunge la acel punct. Aceelaşi efect îl are şi apăsarea butonului din stînga al mouse-ului pe primele două coloane ale liniei la care se doreşte setarea unui punct de oprire. Programul poate fi rulat şi între două puncte de oprire specificîndu-se primul punct ca pornire în execuţie. Acţiunea executată la întîlnirea unui punct de oprire poate fi stabilită prin fereastra *Breakpoints*. La atingerea unui punct de oprire TD poate declanşa următoarele acţiuni:

- *Break* - oprire program, controlul este preluat de către TD putîndu-se examina starea procesorului şi a memoriei;
- *Log* - memorarea valorii unei variabile sau expresii în fereastra *Log*;

- *Execute* - executarea unei expresii şi evaluarea ei. Expresia poate fi în orice limbaj suportat de *TD* (C, Pascal, Assembler) prin alegerea limbajului de interpretare. *TD* evaluează implicit expresia de la adresa cerută în limbajul în care a fost scris codul;
- *Enable* - setarea unui alt punct de oprire;
- *Disable* - anularea unui alt punct de oprire.

At...(Alt-F2) - comandă ce setează un punct de oprire la adresa specificată de utilizator.

Changed memory global... - comandă ce setează un punct de oprire în cazul în care conţinutul unui bloc de memorie se schimbă.

Expression true global... - comandă ce setează un punct de oprire în cazul în care o expresie devine adevărată.

Delete all - comandă ce anulează toate punctele de oprire definite anterior.

A.2.6 Data

Meniul *Data*, obţinut prin tastarea $ALT - D$, conţine comenzi pentru evaluarea, inspectarea şi urmărirea unor variabile, porţiuni de memorie sau expresii în cadrul programului.

Inspect - comandă ce deschide fereastra *Inspector* ce prezintă valoarea unei variabile sau a unei expresii de referinţă la memorie. Dacă cursorul se află în fereastra cu cod sursă, sub numele unei variabile, atunci aceasta este adăugată în fereastra de inspecţii.

Evaluate/Modify... (Ctrl-F4) - comandă pentru evaluarea unei expresii arbitrare introdusă de utilizator.

Add Watch... (Ctrl-F7) - comandă ce plasează o variabilă sau o expresie în fereastra de urmărire (*Watches*). Variabila de sub cursor, dacă acesta este în zona de cod sursă, este adăugată în mod automat în fereastra de urmărire.

A.2.7 Options

Meniul *Options*, obţinut prin tastarea $ALT - O$, conţine comenzi pentru configurarea mediului de depanare şi a opţiunilor care au efect global asupra comportării *Turbo Debugger*. Opţiunile pot fi salvate într-un fişier cu denumirea implicită $tdconfig.td$.

A.2.8 Window

Meniul *Window*, obţinut prin tastarea $ALT - W$, conţine comenzi pentru manipularea ferestrelor. Deşi manipularea ferestrelor se poate face mai uşor cu mouse-ul, în lipsa acestuia, comenziile meniului *Window* rămîn singura soluţie de organizare a afişării mai multor ferestre pe ecran.

- *Zoom (F5)* - comandă ce maximizează fereastra curentă pe tot ecranul. Revenirea la dimensiunea iniţială a ferestrei se face prin aceeaşi comandă. Acelaşi efect se obţine dacă se acţionează butonul din stînga al mouse-ului cînd cursorul este poziţionat pe simbolul [↑] aflat în partea din dreapta sus al ferestrei.

- *Next (F6)* - comandă care determină schimbareea ferestrei curente. Fereastra curentă are un chenar cu linie dublă. Comutarea între ferestre se poate face prin acţionarea butonului din stînga al mouse-ului pe o fereastră diferită de cea curentă.

- *Next pare (Tab)* - comandă care determină mutarea cursorului între subferestrele ferestrei curente.

- *Size/More (Ctrl -F5)* - comandă ce permite redimensionarea şi mutarea ferestrei curente. Cu mouse-ul, redimensionarea se face dacă se acţionează asupra laturilor din dreapta sau de jos ale ferestrei. Mutarea ferestrei se face prin acţionarea mouse-ului asupra laturilor din stînga sau de sus ale ferestrei.

- *Iconize/Restore* - comandă ce minimizează fereastra curentă. Revenirea la dimensiunea iniţială a ferestrei se face prin acceiaşi comandă. Acelaşi efect se obţine dacă se acţionează butonul din stînga al mouse-ului cînd cursorul este poziţionat pe simbolul [↓] aflat în partea din dreapta sus al ferestrei.

- *Close (Alt-F3)* - comandă de închidere a ferestrei curente.

- *Undo Close (Alt-F6)* - comandă de redeschidere a ultimei ferestre închise.

- *User screen (Alt-F5)* - comandă care prezintă ecranul sub forma în care ar apărea dacă s-ar executa programul utilizatorului. După vizualizarea ecranului, apăsarea oricărei taste determină revenirea la ecranul *Turbo Debugger*.

A.3 Meniuri generale

A.3.1 Meniul general

Funcţiile meniului general sînt disponibile tot timpul şi pot fi apelate prin tastele funcţionale. Descrierea funcţiilor meniului este prezentată pe bara de jos a ferestrei principale, aşa ca în figura A.1.

- F1 *Help* - obţinerea informaţiilor despre fereastra în care se află poziţionat cursorul.

- F2 *Bkpt* - plasarea unui punct de oprire la linia cursorului. Comandă similară cu meniul *Breakpoints/Toggle*.

- F3 *Mod* - deschiderea fereastrei corespunzătoare modulelor din cadrul programului. În fereastră se va afla codul sursă a modulului curent. Comandă similară cu meniul *View/ Module*.

- F4 *Here* - executarea programului pînă la linia pe care se află cursorul. Comandă similară cu meniul *Run/Go to cursor*.

- F5 *Zoom* - maximizarea ferestrei în care se află cursorul. Comandă similară cu meniul *Window/Zoom*.

- F6 *Next* - activarea următoarei ferestre deschise. Comandă similară cu meniul *Window/ Next*.

- F7 *Trace* - execuţia unei singure instrucţiuni din program. Comandă similară cu meniul *Run/Trace into*.
- F8 *Step* - execuţia unei singure instrucţiuni. Dacă instrucţiunea este apel de procedură, se execută toată procedura. Comandă similară cu meniul *Run/Step over*.
- F9 *Run* - execuţia întregului program. Comandă similară cu meniul *Run/Run*.
- F10 *Menu* - mutarea cursorului pe bara de sus, a meniului principal.

A.3.2 Meniul general alternativ

Funcţiile meniului general alternativ sînt disponibile tot timpul şi pot fi apelate prin apăsarea tastei ALT şi a unei taste funcţionale.

- Alt-F2 *Bkpt at* - stabilirea unui punct de oprire la adresa specificată. Comandă similară cu meniul *Breakpoints/At*.
- Alt-F3 *Close* - închiderea ferestrei curente. Comandă similară cu meniul *Windows/Close*.
- Alt-F4 *Back* - anularea rezultatului ultimei instrucţiuni executate. Comandă similară cu meniul *Run/Back trace*.
- Alt-F5 *User* - vizualizarea rezultatelor execuţiei programului în fereastra utilizator. Comandă similară cu meniul *Window/User screen*.
- Alt-F6 *Undo* - anularea efectului comenzii anterioare. Comandă similară cu meniul *Window/Undo*.
- Alt-F7 *Instr* - execuţia unei singure instrucţiuni. Comandă similară cu meniul *Run/ Instruction Trace*.
- Alt-F8 *Rtn* - execuţia programului pînă cînd funcţia curentă se reîntoarce în programul care a apelat-o. Comandă similară cu meniul *Run/Until return*.
- Alt-F9 *To* - execuţia programului pînă la adresa specificată. Comandă similară cu meniul *Run/Execute to*.
- Alt-F10 *Local* - deschiderea meniului local, asociat ferestrei active. Aceeaşi acţiune o determină şi apăsarea butonului din dreapta al mouse-ului.

A.4 Meniuri locale

Meniurile locale conţin comenzi specifice ferestrelor selectate. Apelarea meniurilor locale se poate face prin:

- comanda ALT-F10 din meniul general alternativ;
- apăsarea tastei CTRL;
- apăsarea butonului din dreapta al mouse-lui, în suprafaţa ferestrei selectate.

Informaţii suplimentare despre meniurile locale ale ferestrei curente se pot obţine prin apăsarea tastei F1.

A.4.1 Meniul ferestrei CPU

- G - *Goto* - poziţionarea cursorului la adresa indicată astfel încît cursorul se va afla pe prima linie a ferestrei.
- O - *Origin* - poziţionarea liniei pe care se află cursorul pe prima linie a ferestrei.
- F - *Follow* - poziţionarea cursorului pe linia de destinaţie a unei instrucţiuni de salt sau apel de procedură. Instrucţiunea curentă trebuie să fie una de salt sau apel de procedură, altfel instrucţiunea nu are nici un efect.
- C - *Caller* - poziţionarea cursorului pe instrucţiunea care a apelat întreruperea sau procedura curentă.
- P - *Previous* - poziţionarea cursorului pe instrucţiunea pe care a fost anterior poziţionării acestuia cu comanda *Follow* sau *Caller*.
- S - *Search* - căutarea unui byte sau a unei instrucţiuni în cod.
- V - *View source* - deschiderea unei ferestre *Module* care prezintă codul sursă asociat celui dezasamblat.
- A - *Assamble* - asamblarea unei instrucţiuni la adresa curentă a cursorului.
- N - *New* - determină ca următoarea instrucţiune executată să fie cea de la cursor. Adresa instrucţiunii pe care se află cursorul este încărcată în registrele CS şi IP. În acest mod, se poate sări peste o porţiune de cod.

A.4.2 Meniul ferestrei Dump

- G - *Goto* - salt la o adresă specificată de utilizator sub forma *segment:offset*.
- S - *Search* - căutarea unei secvenţe de biţi în fereastra curentă.
- N - *Next* - căutarea următoarei apariţii a secvenţei specificate de comanda *Search*.
- C - *Change* - modificarea conţinutului uneia sau mai multor locaţii de memorie de la poziţia curentă a cursorului.
- F - *Follow* - poziţionarea datelor sau codului la o nouă adresă pe baza datelor din memorie aflate la adresa curentă a cursorului.
- P - *Previous* - poziţionarea cursorului în locul anterior execuţiei comenzilor *Follow* sau *Caller*.
- D - *Display* - schimbarea modului de reprezentare a datelor în fereastra de memorie (Byte, Word, Long, Comp, Float, Real, Double, Extended).
- B - *Block* - manipularea unui bloc de memorie . Submeniul conţine opţiunile:

- *clear* - iniţializare cu zero a unui bloc de memorie;
- *move* - copierea unui bloc de memorie de la o adresă la alta;
- *set* - iniţializarea unui bloc de memorie cu o valoare specificată;
- *read* - citirea dintr-un fişier a datelor şi plasarea acestora într-un bloc de memorie;
- *write* - scrierea datelor dintr-un bloc de memorie într-un fişier.

A.4.3 Meniul ferestrei Register

- I - *Increment* - incremetarea conţinutului registrului selectat;
- D - *Decrement* - decremetarea conţinutului registrului selectat;
- Z - *Zero* - resetarea registrului selectat;
- C - *Change* - modificarea conţinutului registrului selectat;
- R - *Registers* - comutarea între afişarea registrelor pe 16 sau 32 de biţi.

A.4.4 Meniul ferestrei Flag

- T - *Toggle* - comutarea valorii indicatorului selectat. Acelaşi lucru se obţine prin apăsarea tastei ENTER după plasarea cusorului pe indicatorul respectiv.

A.4.5 Meniul ferestrei Stack

- G - *Goto* - mutarea cursorului la adresa specificată;
- O - *Origin* - mutarea cursorului la vîrful stivei, specificat de SS:SP;
- F - *Follow* - mutarea cursorului la adresa indicată în locaţia de date a poziţiei curente din stivă;
- P - *Previous* - mutarea cursorului la adresa anterioară unei poziţionări cu comenziile *Follow* sau *Caller*;
- C - *Change* - modificarea conţinutului locaţiei de memorie selectate.

Anexa B

Schemele machetei MPF1-B microprofessor

Această anexă prezintă schemele machetei MPF1-B Microprofessor echipată cu microprocesor Z80.

Figura B.1 prezintă schema bloc a machetei în care apar denumirea şi plasamentul componentelor şi al tastelor.

Figurile B.2, B.3, B.4, B.5 şi B.6 prezintă schemele electrice complete ale machetei.

Figura B.2 prezintă circuitul generator de semnal de ceas şi de generare a semnalelor de selecţie.

Figura B.3 prezintă conectarea circuitelor Z80-CPU, Z80-PIO şi Z80-CTC.

Figura B.4 prezintă conectarea circuitului 8255. În aceeaşi figură apar schemele circuitelor de intrare/ieşire ale machetei, difuzorului şi stabilizatorului de tensiune.

Figura B.5 prezintă circuitele de memorie şi tastatura machetei.

Figura B.6 prezintă blocul de afişaj al machetei realizat cu şase afişoare 7 segmente.

Figura B.1: Schema bloc a machetei MPF1-B microprofessor.

Figura B.2: Circuitul generator de semnal de ceas şi de generare a semnalelor de selecţie.

Figura B.3: Conectarea circuitelor Z80-CPU, Z80-PIO şi Z80-CTC.

Figura B.4: Conectarea circuitului 8255. Schemele circuitelor de intrare/ieşire ale machetei, difuzorului şi stabilizatorului de tensiune.

Figura B.5: Circuitele de memorie şi tastatura machetei.

Figura B.6: Blocul de afişaj al machetei.