SCRIEREA PROGRAMELOR ÎN ASAMBLARE LA CALCULATOARELE COMPATIBILE IBM – PC

1. Scopul lucrarii

Lucrarea de fata îsi propune prezentarea etapelor de realizare precum si structura unui program în limbaj de asamblare al procesorului Intel 80x86.

2. Memento teoretic

Programele în limbaj de asamblare sunt scrise sub forma de fisiere sursa, care pot fi asamblate cu ajutorul unui program asamblor (MASM, TASM), în fisiere obiect. Legaturile între mai multe module obiect asociate aceluiasi program pot fi facute folosind un program link-editor (LINK, TLINK), pentru a forma fisiere executabile. Etapele de realizare a unui program executabil sunt:

i. editarea fisierului sursa (.ASM), folosind un editor ASCII;

ii. asamblarea fisierului sursa, folosind programul asamblor, obtinându-se fisierul obiect (.OBJ);

iii. editarea legaturilor (link-editarea) pentru fisierul (fisierele) obiect, obtinându-se programul executabil (.EXE).

iv. daca programul este scris în format .COM se va folosi utilitarul EXE2BIN pentru obtinerea programului executabil .COM;

v. rularea programului, eventual depanarea acestuia folosindu-se un program depanator (debugger) ca: DEBUG, TURBO DEBUGGER, SST.

Fisierele sursa sunt formate din instructiuni în limbaj de asamblare, acestea fiind formate la rândul lor, din eticheta, mnemonica, operanzi si comentariu,

despartite prin spatii sau caractere TAB. Mnemonicele pot fi de doua feluri: comenzi date asamblorului (directive) sau instructiuni propriuzise. Directivele specifica modul în care asamblorul va genera codul obiect în momentul asamblarii.

Directive referitoare la definirea structurii de segmente.

Un segment reprezinta o colectie de instructiuni sau de date ale caror adrese sunt raportate la acelasi registru segment. Segmentele din cadrul unui program pot fi definite utilizând definitiile de segment simplificate sau cele complete.

Definirea simplificata a segmentelor utilizeaza în mod implicit conventiile limbajelor de nivel înalt (C). Pentru folosirea definitiilor de segmente simplificate se va declara un model de memorie pentru program. Acesta specifica marimea datelor si codului utilizate în program. Modelele de memorie uzuale sunt:

TINY - datele si codul programului se încadreaza într-un singur segment (64K);

SMALL - datele ocupa un singur segment si codul ocupa un singur segment;

MEDIUM - datele ocupa un singur segment, în timp ce codul poate ocupa mai multe segmente (accesarea codului se face cu referinta îndepartata);

COMPACT - codul ocupa un singur segment, datele ocupând mai multe segmente (accesarea datelor se va face cu referinta îndepartata);

LARGE - atat datele cât si codul pot ocupa mai multe segmente;

HUGE - codul si datele pot ocupa fiecare mai mult de un segment. În plus, pot exista tablouri de date mai mari de 64K.

Definirea modelului de memorie folosit se face cu directiva:

.*MODEL* <nume model>.

Începutul unui segment este indicat prin una din urmatoarele directive:

.*STACK* <marime>	- segment de stiva;
.*CODE* <nume>	- segment de cod;
.*DATA*	- segment initializat de date cu referinta apropiata;
.*DATA?*	- segment neinitializat de date cu referinta apropiata;
.*FARDATA*	- segment initializat de date cu referinta îndepartata;
.*FARDATA?*	- segment neinitializat de date cu referinta îndepartata;
.*CONST*	- segment de date constante.

Datele din segmentele definite cu directivele .*STACK*, .*CONST*, .*DATA*, .*DATA*? sunt plasate într-un grup de segmente denumit *DGROUP*.

Pentru definiri complete de segmente, începutul unui segment este definit cu directiva:

<nume> *SEGMENT* <aliniere>, <combinare>, <utilizare>, <‘clasa’>

iar sfârsitul segmentului este indicat prin folosirea directivei:

<nume> *ENDS*

Alinierea unui segment caracterizeaza adresa unde poate începe un segment. Alinierea poate fi:

BYTE - aliniere la octet;

WORD - aliniere la cuvânt;

DWORD - aliniere la dublu cuvânt;

PARA - aliniere la paragraf (16 octeti) - utilizat implicit;

PAGE - aliniere la pagina (256 octeti).

Câmpul combinare defineste modul de combinare al segmentelor ce au acelasi nume. Ea poate fi:

PUBLIC - concateneaza segmentele cu acelasi nume;

STACK - concateneaza segmentele formând un segment raportat la registrul segment SS;

COMMON - creaza segmente suprapuse prin plasarea adreselor de început ale segmentelor cu acelasi nume la aceeasi adresa;

AT adresa - etichetele si variabilele segmentului vor fi relative la adresa.

Câmpul utilizare defineste marimea cuvântului segmentului atunci când se foloseste un procesor 386/486:

USE16 - cuvântul este de 16 biti;

USE32 - cuvântul este de 32 de biti.

Tipul ‘clasa’ este un mijloc de asociere al segmentelor cu nume diferite si scopuri similare. Acest câmp se foloseste pentru a controla ordinea segmentelor si a identifica segmentele de cod (clasa ‘CODE’).

Pentru referirea cu un singur registru segment a mai multor segmente de date, definite separat în programul sursa, acestea se pot grupa folosind directiva:

nume *GROUP* segment <,segment>...

Când asamblorul necesita referirea unei adrese el trebuie sa cunoasca segmentul în care se afla adresa. Acest lucru se face folosind directiva:

ASSUME registru segment : nume...

unde nume este numele segmentului sau al grupului ce se va asocia cu registrul segment specificat. Atentie: directiva ASSUME indica segmentul asociat doar asamblorului, nu si procesorului.

Initializarea registrelor segment.

Registrele CS si IP sunt intializate prin specificarea unei adrese de început (etichete) cu directiva:

END <adresa început>

Registrul DS trebuie initializat la adresa segmentului ce va fi folosit pentru date. Daca se foloseste definirea simplificata a segmentelor, initializarea registrului DS poate fi facuta în modul urmator:

```
mov ax,@data
mov ds, ax
```

Daca se folosesc definitii complete:

```
mov ax, <nume segment>
mov ds, ax
```

Registrul SS este initializat automat la valoarea segmentului *.STACK* în cazul definirii simplificate, respectiv la valoarea ultimului segment cu tipul de combinare *STACK*.

Registrul ES nu este initializat automat, putând fi folosit pentru alte referiri.

Folosirea specificatorilor de tip.

Pentru specificarea dimensiunii unei variabile sau a unui operand se pot folosi urmatoarele cuvinte cheie:

BYTE	- variabila octet;
WORD	- variabila cuvânt;
DWORD	- variabila dublu cuvânt;
FWORD	- variabila de 6 octeti;
QWORD	- variabila de 8 octeti;
TBYTE	- variabila de 10 octeti.

Modul în care este referita o variabila poate fi specificat prin:

FAR	- referinta îndepartata (segment + offset);
NEAR	- referinta apropiata (offset);

PROC - referinta cu tipul implicit al modelului curent de memorie.

Folosirea procedurilor.

Sintaxa pentru definirea unei proceduri este:

eticheta *PROC* <*NEAR*/*FAR*>
    instructiuni
eticheta *ENDP*

Etichete de cod.

Înafara de etichetele create prin definirea procedurilor, mai pot exista doua tipuri de etichete:

- etichete cu referinta apropiata, cu sintaxa:

    nume:

- etichete definite cu directiva LABEL, cu sintaxa:

    nume LABEL distanta

unde distanta poate fi: NEAR, FAR sau PROC.

Definirea de variabile.

Variabilele sunt formate din unul sau mai multe obiecte de date de marime specificata. Sintaxa:

<nume> directiva <initializator>

Marimea datelor este data de directiva folosita:

DB - defineste variabila octet;
DW - defineste variabila cuvânt;
DD - defineste variabila dublu cuvânt;
DF - defineste variabila de 6 octeti;
DQ - defineste variabila de 8 octeti;
DT - defineste variabila de 10 octeti.

Pentru definirea tablourilor se foloseste operatorul *DUP*:

numar *DUP* (valoare initiala)

Exemplu: *DB* 10 *DUP* (0) - aloca 10 octeti cu valoarea initiala 0.

Utilizarea structurilor si înregistrarilor.

O variabila structura reprezinta o colectie de date de dimensiuni diferite ce pot accesate prin intermediul aceleiasi variabile. Sintaxa de declarare a unei structuri:

nume structura *STRUC*
    declaratii câmp
nume structura *END*

Pentru declararea unei variabile de tip structura se foloseste sintaxa:

nume nume structura <valoare initiala>...

Pentru accesarea unui anumit câmp al unei structuri se foloseste operatorul '.':

nume.câmp

O variabila de tip înregistrare este o variabila de lungime octet sau cuvânt unde anumite câmpuri de biti pot fi accesate simbolic. Declararea unei înregistrari:

nume înregistrare *RECORD* câmp,...

unde câmp reprezinta:

nume_câmp: marime <=expresie>

Exemplu: color *RECORD* blink:1, back:3, intense:1, fore:3

Declararea unei variabile de tip înregistrare se face dupa sintaxa:

nume nume înregistrare <valoare initiala>

Utilizarea macrodefinitiilor.

Macrodefinitiile permit atribuirea unui nume simbolic pentru un bloc de instructiuni în vederea folosirii numelui respectiv pentru referirea la blocul de

instructiuni. Pentru o macrodefinitie se pot defini parametri ce îi vor fi pasati. Definirea unei macrodefinitii se fece dupa sintaxa:

nume macrodefinitie *MACRO* <parametru>...
instructiuni
ENDM

Apelarea unui macro se va face:

nume macrodefinitie <argument>...

Definirea etichetelor în cadrul unei macrodefinitii trebuie sa se faca cu directiva *LOCAL*. Sintaxa:

LOCAL nume eticheta <, nume eticheta>...

Exemplu:
```
aduna MACRO ad1,ad2
    mov ax,ad1
    add ax,ad2
ENDM
```

Apelarea: aduna bx,10

Crearea programelor din module multiple.

Programele în limbaj de asamblare de dimensiuni medii si mari sunt create pornind de la mai multe fisiere sursa. Pentru folosirea globala a unor simboluri,

definite într-unul din module, în modulul de definitie simbolurile se vor declara folosind directiva:

PUBLIC nume simbol<,nume simbol>...

În modulele în care se folosesc simboluri declarate alte module, ele se vor declara cu sintaxa:

EXTERN nume simbol:tip <nume simbol:tip>...

Folosirea operatorilor.

Cei mai utilizati operatori pusi la dispozitie de macroasambloarele pentru I8086 sunt:

PTR - specifica tipul pentru o variabila sau o expresie:

tip *PTR* expresie

Exemplu: *WORD PTR* es;[si]

THIS - creaza un operand de tip specificat, ale carui valori de deplasament si segment sunt egale cu valoarea curenta a contorului de locatii. Sintaxa:

THIS tip

SEG - întoarce adresa segmentului unei expresii. Sintaxa:

SEG expresie

OFFSET - întoarce deplasamentul unei expresii. Sintaxa:

OFFSET expresie

SIZE - întoarce numarul total de octeti pentru o variabila definita cu operatorul *DUP*:

SIZE variabila

$ - reprezinta adresa instructiunii curente ce se executa

EQU - folosit pentru a atribui o constanta numerica unui simbol. Sintaxa:

nume *EQU* expresie

ORG - pozitioneaza contorul de locatii la un nou deplasament. Sintaxa:

ORG deplasament

3. Exemple de program rezolvate si probleme propuse

Problema rezolvata:

Testarea mecanismului de prefetch la microprocesorul 8086.

```
cod     segment PARA ‘CODE’
        assume cs:cod, ds:cod
        org 100h

merr    db 13,10,’8086 greseste prin mecanismul de prefetch!!!’,13,10,’$’
ok      db 13,10,’Acest 8086 nu greseste!!!’,13,10,’$’
mes     db 13,10,’Test mecanism de prefetch la microprocesor.’,13,10,’$’

begin:  mov dx,cs
        mov ds,dx
        mov dx,offset mes       ;afisarea scopului programului
        mov ah,09h
        int 21h
        jmp start               ;golire QFIFO
start:  mov si,offset et
        mov al,0aah
        mov [si+1], al          ;modificarea instructiunii urmatoare in memorie:
        mov cx,0ffaah !!
et:     mov cx,0ffffh
        sub cx,0ffaah
        jz et1
        mov dx,offset merr      ;8086 nu simte modificarea in QFIFO
        jmp et2
et1:    mov dx,offset ok                ;8086 simte modificarea
et2:    mov ah,09h
        int 21h
        mov ax,4c00h
```

```
    int 21h
cod ends
end begin
```

Sa se ruleze programul de mai sus atât prin lansare de la prompt-ul DOS, cât si pas cu pas folosind un program de depanare (DEBUG, SST, TURBO DEBUGGER, etc).

Problema propusa:

Sa se studieze comportarea microprocesorului 8086 la accesarea unui operand pe cuvânt de la o adresa de offset 0ffffh.

Bibliografie

[1] **Caprariu V., Enyedi A., Muntean M.** – *Sistemul de operare DOS*: *Ghidul Programatorului*, Editura MicroInformatica, Editia III, Cluj Napoca, 1993.

[2] **Athanasiu I., Panoiu A.** – *Calculatoare Personale*: *Microprocesoarele 8086, 286, 386*, Editura Teora, Bucuresti, 1993.