Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Programare în limbaj de asamblare

20. Definirea şi utilizarea segmentelor (directive simplificate şi complete).

Definirea şi utilizarea segmentelor

Un segment este definit ca o colecție de instrucțiuni sau date, ale căror adrese sunt relative față de începutul segmentului (față de registrul segment corespunzător). Specificarea segmentelor se poate face în două moduri:

- definire simplificată a segmentelor;
- definire completă a segmentelor.

Cele patru registre segment la care procesorul are acces în orice moment, CS, DS, ES şi SS, deci care pot face referire la patru segmente logice, pot corespunde la patru segmente fizice distincte sau pot exista suprapuneri parțiale sau totale ale acestor segmente. Adresa fizică aferentă unui segment este multiplu de paragraf (16 octeți).

Forma simplificată are avantajul, din punct de vedere al utilizatorului, că oferă o gestiune mai simplă a segmentelor. Un program are, în general, un segment de cod, unul de date şi unul de stivă. Excepție de la această regulă fac programele cu extensia *.com*, care pot fi definite utilizând definirea simplificată, modelul *tiny* şi opțiunea respectivă de editare de legături (/t), fără a fi necesară definirea unui segment de stivă şi inițializarea registrelor segment, deoarece toate vor fi inițializate cu aceeaşi valoare, reprezentând adresa de început a segmentului ce conține întregul program, date şi stivă (toate intră într-un segment de 64 Ko).

La încărcarea în memorie a unui program, pentru execuție, sistemul de operare inițializează registrul segment CS cu prima adresă de segment disponibilă, iar registrul IP cu adresa relativă, din cadrul segmentului, a primei instrucțiuni ce trebuie executată. Dacă programul conține cele trei segmente (cod, date şi stivă), atunci aceste segmente logice se încarcă în memorie în ordinea logică: cod, date, stivă.

Forma completă de definire a segmentelor

Se utilizează o declarație de forma:

```
nume_seg SEGMENT [tip_aliniere][tip_combinare][tip_utilizare][clasa_seg]
        ....
        ...     < corpul segmentului >                  ; instructiuni sau date
        ....
nume_seg ENDS
```

unde *nume_seg* este numele asociat segmentului, care trebuie să fie unic şi căruia i se asociază o adresă de segment (de 16 biți) corespunzătoare poziției în memorie a segmentului în faza de execuție a programului. Inițializarea unui registru segment (DS, ES sau SS) cu un segment declarat revine utilizatorului, care o va executa în mod explicit în cadrul programului, utilizând pentru aceasta numele segmentului respectiv:

```
mov     ax, <nume_seg>
mov     ds, ax          ; in mod asemanator pentru ES sau SS
```

Următoarele referiri la memorie care implică registrul DS trebuie să conțină trimiteri la date din acest segment (asamblorul nu verifică dacă aceste referințe sunt corecte).

tip_aliniere este o informație referitoare la adresa fizică de început la care este depus segmentul în memorie, şi anume dacă este divizibilă cu 1, 2, 4, 16 sau 256. Acest tip poate fi:

- byte – adresă divizibilă cu 1;
- word – adresă divizibilă cu 2;
- dword – adresă divizibilă cu 4;
- para – adresă divizibilă cu 16 (paragraf);
- page – adresă divizibilă cu 256.

Omiterea operatorului tipului de aliniere pentru primul segment va cauza alinierea implicită la tipul *para*. Omiterea tipului de aliniere pentru un segment care nu este unic, implicit va determina alinierea la tipul specificat la definirea anterioară a segmentului cu acelaşi nume. Pentru alinierea datelor, de exemplu la dublu cuvânt, se poate folosi directiva *ALIGN 4*.

tip_combinare sau tip segment constituie o informație pentru editorul de legături, care specifică raportul dintre acest segment şi alte segmente definite în alte module obiect. Acest parametru poate fi:

- *PUBLIC* – ceea ce arată că acest segment va fi concatenat (înlănțuit) cu alte segmente cu aceleaşi nume din alte module obiect, obținându-se un singur modul cu acest nume;
- *COMMON* – precizează că acest segment şi alte segmente cu aceleaşi nume din alte module vor fi suprapuse, deci vor începe de la aceeaşi adresă. Lungimea unui astfel de segment va fi lungimea celui mai mare segment dintre cele suprapuse;
- *AT <expresie>* – un astfel de segment va fi plasat la adresa reprezentată de valoarea <expresie>, care este o valoare de 16 biți, reprezentând adresa în memorie a segmentului. Un astfel de segment nu va conține, însă, cod sau alte date inițializate. El reprezintă un şablon care se poate suprapune peste codul sau datele aflate deja în memorie. Asamblorul nu generează cod sau date pentru un segment de acest tip, dar permite adresarea în cadrul acestui segment.

  Exemplu de utilizare:

```
RAM_graph segment at 0b800H
        start_buffer        label       byte
RAM_graph ends
        .
        .
        .
mov     ax, RAM_graph
mov     es, ax
assume es:RAM_graph     ; in continuare se poate face referirea in acest
                        ; segment utilizand adresa de start 'start_buffer'
```

- *MEMORY* – segmentul curent de acest tip va fi aşezat în memorie în spațiul rămas disponibil după aşezarea celorlalte segmente (către sfârşitul memoriei). Dacă sunt editate împreună mai multe segmente de acest tip, următoarele vor fi tratate ca segmente obişnuite.
- *STACK* – va concatena toate segmentele cu aceleaşi nume, pentru a forma un segment unic; referirea acestui segment se face prin SS:SP. Registrul SP va fi inițializat, în mod automat cu lungimea stivei.

- *NONE* – segmentul va fi separat logic de late segmente, deşi ele se pot termina fizic adiacent. Acest tip este implicit pentru directivele complete de segmentare. Se presupune că segmentul va avea propria sa adresă de bază.

tip_utilizare – reprezintă atributul modului de utilizare, pentru procesoarele de la 386 în sus, ce poate lua valorile: USE16, USE32 sau FLAT; aceste atribute specifică modul de lucru, de bază, pentru aceste segmente: pe 16 biți sau pe 32 de biți. Pentru codul ce se va executa în modul real sub DOS se vor utiliza segmente de 16 biți, în timp ce segmentele ce lucrează pe 32 de biți vor fi cele ce se execută în modul protejat. Amănunte suplimentare pentru aceste două atribute vor fi prezentate în capitolul referitor la combinarea între codul de 16 biți şi codul de 32 de biți. Pentru a stabili sau forța, ca implicit, un anumit tip de utilizare pentru anumite segmente dintr-un modul se poate plasa înainte de începutul segmentelor directiva:

option segment: use16

Ca *tip_utilizare* se mai pot utiliza şi atribute de tipul *read-only*, pentru ca asamblorul să genereze mesaje de eroare la încercarea (doar la asamblare) de a scrie în acel segment. Cu toate acestea, la execuția programului, acesta poate să scrie într-un astfel de segment.

clasa_seg – este un nume cuprins între apostrofuri simple; rolul acestui nume este de a permite controlul ordinii în care vor fi depuse în memorie segmentele care compun un program. Dacă nu se precizează clasa, atunci aceasta este considerată şirul vid de caractere. Două segmente cu aceeaşi clasă vor fi dispuse în memorie la adrese succesive. Specificarea clasei furnizează un mijloc de a colecta segmente specificate similar la editarea legăturilor (primul mijloc este numele segmentului).

Să considerăm, de exemplu, două module care conțin următoarele declarații:

Modul 1:

```
aseg        segment     byte        public      'code'
start:      .
        .......
        .
aseg        ends

bseg        segment     word    common          'data'
        .
        .......
.
bseg        ends

cseg        segment     para        stack       'stack'
        .
        .......
        .
cseg        ends

dseg        segment     at      0b800H
        .......
        .
```

```
dseg        ends
end         start
```

Modul 2:

```
aseg        segment     byte    public      'code'
            .
            .......
            .
aseg        ends

bseg        segment     word        common          'data'
            .
            ......
            .
bseg        ends
end
```

Pentru aceste declarații, editorul de legături va dispune segmentele în memorie astfel:

	aseg (M1)	aseg(M2)	bseg (M2)	cseg		dseg	
			bseg (M1)	(M1+M2)		(M1)	
00000H						B8000H	***FFFFFH***

Segmentele sunt plasate în memorie în ordinea în care au fost declarate. Segmentul de stivă, dacă este declarat complet, va fi amplasat, indiferent de poziția sa în textul sursă, după celelalte segmente. Dacă are clasa stack, SP va fi inițializat automat cu dimensiunea stivei, altfel nu va fi inițializat.

Un segment poate fi redeschis, deci cu un nume declarat anterior, în care caz contorul de locații $ este încărcat cu valoarea salvată în momentul întâlnirii directivei ENDS pentru acel segment.

Pe lângă tipurile de combinare menționate anterior mai există şi tipul PRIVATE, care este un tip special (none) şi care nu se combină cu nici un alt tip.

Segmente imbricate

Segmentele nu sunt niciodată fizic imbricate (incluse unul într-altul); totuşi, este permis să se codifice o porțiune de segment, apoi să se înceapă şi să se încheie un alt segment, după care să se termine codul început primul. Când se editează un program în acest mod, asamblorul va concatena cea de-a doua porțiune a segmentului la prima. Se spune că segmentele sunt logic imbricate, dar nu fizic imbricate. De exemplu:

```
code1       segment
assume      cs:code1, ds:data1
            .......
data1               segment
            .......
data1               ends
```

```
                .....
code1           ends
```

În schimb, aceste segmente nu se pot intersecta:

```
data3           segment
                .....
data4           segment
                .....
data3           ends
                .....
data4           ends
```

În consecință, o astfel de declarare a segmentelor nu este permisă.

Directive simplificate de segmentare

.MODEL tip_model

Prin această directivă se specifică dimensiunea şi modul de dispunere a segmentelor în memoria RAM. Modelele de implementare, tip_model, sunt următoarele:

tiny	pentru care LP + LD + LS < 1 segment (64K)
small	pentru LP < 1 seg (64K), LD + LS < 1 seg.
medium	pentru LP > 1 seg (64K), LD + LS < 1 seg.
compact	pentru LP < 1 seg (64K), LD + LS > 1 seg.
large	pentru LP > 1 seg (64K), LD + LS > 1 seg.
huge	la fel cu modelul anterior, cu diferența că referințele sunt normalizate.

Abrevierile utilizate sunt LP, LD, LS şi reprezintă lungimea programului (codului), dimensiunea memoriei pentru date şi respectiv pentru stivă. Aceste abrevieri pot fi interpretate şi ca număr de segmente de cod (LP), respectiv număr de segmente de date (LD), în sensul că, de exemplu pentru modelul *small* există numai un segment pentru cod şi unul pentru date, în timp ce pentru modelul *large* sunt definite mai mult de câte un segment, atât pentru cod, cât şi pentru date.

Referințele la variabile şi etichete care se găsesc în alte segmente se pot exprima în două forme:

a) prin adresa logică: adresa segmentului + offsetul, care nu este unică; de exemplu, aceeaşi adresă fizică poate fi exprimată astfel:
1234H:0005H, sau 1230H:0045H, sau 1200H:0345H etc.

b) prin adresa normalizată (sau fizică), adică un întreg de 32 biți, sau 48 biți la 386, care însă este unică; pentru exemplul anterior, aceasta este:
12345H.

Directiva *model* este prima din programul sursă. Când se leagă programe scrise în limbaje de nivel înalt cu programe scrise în limbaj de asamblare, toate trebuie să posede acelaşi model. Mai există şi modelul TPASCAL, care este echivalent cu modelul LARGE, PASCAL, la care se adaugă convențiile de transfer pentru parametri între subprograme, conform convențiilor din TPascal.

În cazul în care programul în limbaj de asamblare se va lega cu un program scris în limbajul C sau PASCAL, acest lucru se va specifica în directiva .model:

- .model small, C
- .model large, Pascal (echivalentă cu modelul TPascal)

Aceasta este necesar pentru a se respecta convențiile specifice fiecărui limbaj în ceea ce privește simbolurile externe și transferul parametrilor între subprograme.
Directiva .model presupune o directivă implicită ASSUME, care se referă doar la registrele segment de cod, date și stivă, dar nu și la extrasegmentul de date (ES).

.STACK [dimensiune]

Această directivă alocă o zonă de memorie, de dimensiune specificată, pentru segmentul de stivă. Dacă nu se specifică parametrul dimensiune, acesta va fi implicit de 1Ko; de fapt, cu valoarea acestui parametru va inițializa asamblorul registrul SP.

.CODE [nume]

Această directivă precede segmentul de cod (program). Încărcarea acestui registru segment (CS) se va face automat de către DOS la încărcarea segmentului pentru execuție. Sistemul de operare DOS are sarcina amplasării segmentelor în memorie. Se pot asocia nume (maxim 6 caractere) pentru segmentele de cod, pentru modelele medium, large și huge; pentru celelalte modele, el este ignorat.

.DATA

Aceasta descrie segmentul de date pentru care utilizatorul trebuie să inițializeze, în mod explicit, registrul segment DS, cu adresa segmentului de date, @DATA; simbolul acesta va primi adresa segmentului de date, după editarea legăturilor, în momentul încărcării sale de către sistemul de operare. Pentru ca programele să fie relocabile, registrele segment nu se încarcă niciodată cu valori absolute.

Pe lângă acestea mai există directivele:

.DATA? pentru date neinițializate, referințe de tip NEAR;
.FARDATA [nume] date inițializate sau nu, referințe de tip FAR;
.FARDATA? [nume] date neinițializate, referințe de tip FAR;
.CONST segment cu date de tip „read-only“ (constante), care pot fi rezidente și în ROM, spre deosebire de celelalte care sunt rezidente numai în RAM.

Segmentele definite cu directivele:
data, .const, .data?, .stack
vor fi grupate automat în grupul predefinit DGROUP pentru acele modele la care dimensiunea acestora nu depășește un segment.

Directivele simplificate nu specifică un nume pentru segmentele logice declarate, ele având nume implicite:

directiva	numele	aliniere	combinare	clasa	grupul
.code	nume_TEXT	word	public	'CODE'	-
.data	_DATA	word	public	'DATA'	DGROUP
.const	CONST	word	public	'CONST'	DGROUP
.data?	_BSS	word	public	'BSS'	DGROUP
.stack	STACK	para	stack	'STACK'	DGROUP

.fardata	FAR_DATA	para	private	'FAR_DATA'	-
.fardata?	FAR_BSS	para	private	'FAR_BSS'	-

Numele concatenat cu _TEXT este numele asociat în directiva *.code* sau, dacă acesta lipseşte, va fi numele fişierului. Tabela de segmente şi grupuri de la sfârşitul listingului conține denumirile reale ale segmentelor; la asamblare se poate solicita (prin opțiunea /l) furnizarea listingului (adică lista de instrucțiuni asamblate, alături de codurile şi adresele lor) într-un fişier care are extensia *.lst*.

Directiva model (pentru tiny, small şi medium) va genera liniile sursă adecvate în care apare directiva GROUP:

```
DGROUP      GROUP       _DATA, CONST, _BSS, STACK
```

iar pentru modelele small şi compact, linia sursă pentru directiva ASSUME:

```
ASSUME      CS:nume_TEXT, DS:DGROUP, SS:DGROUP
```

Utilizarea formei simplificate de declarare a segmentelor nu scuteşte programatorul de sarcina specificării modului de utilizare a registrelor de segment şi nici de asigurarea conținutului registrelor segment, conform pseudoinstrucțunii ASSUME.

Numele implicite ale segmentelor pot fi obținute utilizând simbolurile predefinite:

@CODE numele segmentului de cod;
@DATA?, **@DATA** numele segmentului de date (ne)inițializate;
@FARDATA numele segmentului de date inițializate, de tip FAR;
@FARDATA? numele segmentului de date neinițializate, FAR;
@CURSEG numele segmentului curent executat; forma:
@curseg ends
conduce la închiderea segmentului curent;

Aceste simboluri pot fi utilizate în directivele ASSUME şi la inițializarea registrelor segment:

```
assume      cs:@code, ds:@data
mov     ax, @data
mov     ds, ax
```

Alte simboluri predefinite:

@FILENAME numele de bază al fişierului sursă curent; această facilitate permite modificarea simplă a tuturor denumirilor ce depind de numele fişierului sursă.
@CODESIZE are valoarea: 0 pentru tiny, small şi compact
1 pentru medium, large, huge.
@DATASIZE are valoarea: 0 small, medium
1 compact, large
2 huge.

Simbolurile **@CURSEG** şi **@FILENAME** pot fi folosite şi în directivele complete de segmentare. În forma simplificată, dacă dorim să utilizăm instrucțiuni specifice procesoarelor

286/386/, se specifică acest lucru prin directive de forma .286, .386 etc., care se plasează imediat după directiva model.

Pentru a plasa segmentele de cod, date şi stivă în ordinea logică (cod, date, stivă) se poate utiliza directiva DOSSEG, altfel ele vor fi plasate în ordinea în care apar fişierele în comanda TLINK (mai puțin segmentul de stivă, care este oricum plasat la sfârşit).

@CPU returnează o valoare de 16 biți, care determină directivele procesor active, după bitul respectiv setat: 0-8086, 1-186, 2-286, 3-386, 4-486, 5-Pentium, 6-rezervat 8086, 7-instrucțiuni specifice modului protejat, biții 8-11 sunt puşi pe 1 pentru coprocesoarele 8087, 287, respectiv 387 (bitul 11 este setat pentru toate procesoarele începând de la 486/ Pentium).

@WORDSIZE returnează o valoare de 16 biți, care determină tipul de utilizare al segmentului: 2 pentru segment de 16 biți, respectiv 4 pentru segment de 32 de biți.

@MODEL returnează 1 pentru tiny, 2 pentru small, 3-compact, 4-medium, 5-large, 6-huge, 7-flat.

Alte directive:

.SEQ

.ALPHA

se pot utiliza pentru dispunerea segmentelor în ordine secvențială, respectiv alfabetică. Segmentele ce aparțin aceleiaşi clase vor fi dispuse împreună, indiferent de ordinea secvențială sau alfabetică din fişierul sursă.

Se mai pot utiliza următoarele directive cu semnificația specificată pentru fiecare.

.STARTUP

generează instrucțiunile pentru inițializarea registrelor segment, şi

.EXIT

care poate fi utilizată pentru generarea funcției de ieşire dintr-un program şi revenire în DOS, adică funcția 4Ch, int 21h.

ALIGN

determină asamblorul să alinieze data sau instrucțiunea care urmează începând de la o adresă în concordanță cu o anumită valoare. Alinierea aceasta poate facilita procesorului accesul la cuvinte sau cuvinte duble. Formatul ei este următorul:

align valoare

unde *valoare* este o putere a lui 2, cum ar fi 2, 4, 8 sau 16. Să presupunem că valoarea contorului de locații este 0005, când se dă directiva *ALIGN*:

```
0005H           ALIGN 4
0008H           dublu_cuv dd 0
```

care va alinia variabila *dublu_cuv* la o adresă de tip cuvânt dublu, adică divizibilă cu 4.

Asamblorul va umple octeții nefolosiți cu 0 pentru date şi cu *NOP* pentru instrucțiuni.

EVEN

cere asamblorului să avanseze contorul de locații astfel încât data sau instrucțiunea următoare să fie aliniată la o adresă pară. Dacă în exemplul anterior se plasează directiva *EVEN* în locul directivei *ALIGN*, data *dublu_cuv* va fi plasată la adresa *0006H*.

.CREF

specifică asamblorului să genereze tabela cu referințe încrucişate. Ea este implicită, dar dacă se doreşte suprimarea acestei tabele, atunci această directivă trebuie urmată de **.XCREF.**

.tip_procesor

Pentru a specifica utilizarea unui anumit tip de procesor, şi deci a unor instrucțiuni specifice acestuia, se pot utiliza directivele procesor, care sunt plasate, de obicei, la începutul programului sursă. Ele pot fi însă plasate şi în interiorul unui program, în punctul în care se doreşte activarea sau dezactivarea caracteristicilor unui procesor. De exemplu:

.286 .386 .486 permit utilizarea instrucțiunilor specifice acestor procesoare.

.286P .386P .486P .586P permit utilizarea instrucțiunilor specifice acestor procesoare, plus instrucțiunile modului protejat.

Pe lângă acestea, se mai pot utiliza directivele ce specifică utilizarea instrucțiunilor în virgulă mobilă specifice unui anumit tip de coprocesor matematic: .8087, .287 sau .387. Directivele care specifică tipul de procesor (.8086, .286,. 386) permit (activează) în mod implicit şi utilizarea instrucțiunilor coprocesor corespunzătoare (8087, 287, 387). Singurul scop al directivelor coprocesor este acela de a permite execuția de instrucțiuni specifice unui coprocesor alături de cele ale unui alt tip de procesor: 287 cu 8086 sau 186, sau instrucțiuni 387 cu instrucțiuni 8086, 186 sau 286.

COMMENT

Această directivă permite comentarii pe mai multe linii, în formatul următor:

```
COMMENT         delimitator         [comentarii]
[comentarii]
[comnetarii .....]
delimitator         [comentarii]
```

Delimitatorul este primul caracter diferit de spațiu, cum ar fi % sau + sau orice alt caracter ce urmează după directiva *COMMENT*. Comentariile se termină pe linia pe care apare cel de-al doilea delimitator:

```
COMMENT         $           acesta
este un comentariu
pe mai multe
$           linii
```

Exemplu de utilizare a celor două forme de declarare a segmentelor:

Să scriem un program care afişează un mesaj, utilizând ambele forme de declarare a segmentelor.

a) Forma completă de declarare a segmentelor:

```
date    segment    word    public    'data'
mesaj   db         'exemplu program 1a$'
date    ends
cod     segment    word    public    'code'
assume  cs:cod, ds:date, ss:stiva
start:
mov     ax, date                ; initializare registru segment
mov     ds, ax                  ; pentru date DS
mov     dx, offset mesaj        ; initializare adresa mesaj in DX
mov     ah, 9                   ; apel functia 9 – DOS, de afisare
int     21h                     ; a unui text, cu adresa data in DX
mov     ax, 4c00h               ; revenire in DOS
int     21h
cod     ends
stiva   segment    word    stack    'stack'
dw      10  dup    (?)             ; rezervare memorie pentru stiva
stiva   ends
end     start
```

b) Forma simplificată de definire a segmentelor:

```
.model      small
.stack      10
.data
        mesaj   db      'exemplu program 1b$'
.code
        start:  mov     ax, @data       ; initializare registru segment
                mov     ds, ax          ; pentru date DS
                mov     dx, offset mesaj    ; initializare adresa mesaj in DX
                mov     ah, 9           ; apel functia 9 – DOS, de afisare
                int     21h             ; a unui text, cu adresa data in DX
                mov     ax, 4c00h       ; revenire in DOS
                int     21h
end     start
```