UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI
Instrumente Structurale
2007-2013

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

Programare în limbaj de asamblare

18. Operatori şi directive.

Operatori

Operatorii aritmetici

Operatorii aritmetici pot fi unari (cu un singur operand): +, – (deci stabilesc semnul unei expresii) sau binari: +, -, *, /, *MOD* (cu doi operanzi);

MOD – reprezintă restul împărțirii întregi.

/ – este operator pentru împărțire întreagă.

Operatorul index

Acest operator este specificat prin paranteze drepte [ ] reprezintă practic o adunare; este utilizat pentru precizarea operanzilor prin adresare indirectă, indexată, bazată, bazată şi indexată:

```
depl [bx] [di]
depl [bx + di]  echivalente cu: depl + (bx) + (di)
[depl + bx + di]
```

sau o referință de forma:

```
alfa [4]
```

specifică de fapt referirea la locația:

```
alfa + 4.
```

Operatori logici

Aceşti operatori sunt: NOT, AND, OR, XOR şi realizează operațiile logice respective bit cu bit, pentru expresii care sunt evaluate, la asamblare, la valori constante:

	NOT	< expresie >
< expresie 1 >	AND	< expresie 2 >
< expresie 1 >	OR	< expresie 2 >
< expresie 1 >	XOR	< expresie 2 >

Aceşti operatori nu trebuie confundați cu instrucțiunile de procesor anonime, care operează asupra registrelor sau memoriei în timpul execuției programului. Operatorii logici utilizează expresii, evaluate valori constante, în faza de asamblare a programului. De exemplu:

```
alfa        db      01110111B and 0f0H
```

Operatori de deplasare

Aceşti operatori sunt utilizați pentru deplasarea logică a unor expresii, evaluate la valori constante în momentul asamblării programului, cu un număr de biți la stânga sau dreapta:

< expresie >	SHR	< număr >
	SHL	

unde < *număr* > reprezintă o expresie întreagă care indică numărul de biți pe care se execută deplasarea.

La deplasare, biții deplasați în afara operandului se pierd, iar cei eliberați prin deplasare devin zero (indiferent de sensul deplasării). Aceşti operatori nu trebuie confundați cu

instrucțiunile de procesor anonime, dar care acționează asupra unor operanzi din registre sau din memorie.

Operatori relaționali

Aceşti operatori compară două expresii, rezultatul fiind o valoare logică (true = ffH sau false = 00H) în funcție de valoarea de adevăr a relației dintre cei doi operanzi.

Operatorii sunt:

- EQ – egalitate;
- NE – inegalitate;
- GT – mai mare;
- GE – mai mare sau egal;
- LT – mai mic;
- LE – mai mic sau egal.

Sintaxa pentru utilizarea acestor operatori este:

< expresie 1 > [operator] < expresie 2 >

Operatori de tip şi de conversie

Aceşti operatori specifică/analizează tipul operanzilor din memorie sau alte expresii, sau pot realiza conversie de tip.

a) *HIGH, LOW* < *expresie* >

Cei doi operatori furnizează octetul mai semnificativ, respectiv octetul mai puțin semnificativ al unei expresii de tip cuvânt (doi octeți). De exemplu:

```
data_16     equ     0abcdh
mov         ah, high data_16
mov         al, low data_16
```

pentru a încărca primii opt biți, respectiv următorii opt biți dintr-o constantă de tip cuvânt.

b) *SEG* < *expresie* >

Acest operator furnizează adresa de segment a unei adrese exprimată printr-o expresie, care poate fi variabilă, operand memorie, etichetă, numele unui segment sau numele unui grup de segmente. Exemplu:

```
beta        dw.....
                .
                .
mov     bx, seg beta
mov     ds, bx
```

c) *OFFSET* < *expresie* >

Operatorul furnizează deplasamentul în cadrul segmentului pentru o expresie de tip variabilă (simplă sau structurată), operand memorie, etichetă. Aplicat unei variabile sau etichete, operatorul furnizează offsetul (deplasamentul) variabilei sau etichetei. Valoarea este cunoscută în momentul link-editării, când se face alinierea finală a segmentului, deoarece offset-urile de la

asamblare se pot modifica, la localizare, dacă segmentul este combinat cu părți ale aceluiași segment, definite în alte module sau dacă nu este aliniat la un paragraf. De exemplu, poate fi utilizat în adresarea indirectă a variabilelor:

```
sir     dw      .......
....
mov     bx,             offset sir      ; adresa de inceput a sirului
mov     si,             0               ; se initializeaza indexul primului element din sir
                ...
                ...
add     ax,             [bx] [si]               ; se insumeaza elementele
```

Dacă se utilizează directiva group, operatorul OFFSET nu va returna offset-ul unei variabile din grup, ci pe acela al variabilei din segmentul său. Pentru a returna offset-ul variabilei din grupul respectiv trebuie utilizat prefixul grupului respectiv în fața variabilei:

```
dgrup                   group           data1, data2
data1                   segment
                ......
data1                   ends
data2                   segment
                .....
        val             db      ....
                ......
        dw              val             ; offfsetul in cadrul segmentului furnizat de asamblor
        dw              dgrup:val       ; offsetul in grup
        dd              val             ; offset in segment + adresa segment
        dd              drgup:val       ; offset in grup + adresa segment
data2           ends
        mov     bx,offset val           ; offset in segment
        mov     bx,offset dgrup:val     ; offset in grup
```

Observație: - offset [bx] este echivalent cu [bx]
- offset depl[bx] este echivalent cu [depl+bx]

d) *TYPE < expresie >*

Furnizează un număr întreg ce reprezintă tipul expresiei asupra căreia se aplică; are un singur argument, care poate fi o variabilă sau etichetă.

Dacă expresia este o variabilă (simplă sau structurată), operatorul returnează numărul de octeți pe care se reprezintă componentele variabilei simple sau numărul de octeți ai variabilei structurate. Valorile returnate de operator, în funcție de tipul variabilei, sunt:

```
byte    - 1;
word    - 2;
dword   - 4;
qword   - 8;
pword   - 6;
fword   - 6;
```

tbyte - 10;
structura – numarul de octeti ai structurii;

Se utilizează, de obicei, în calcule asupra vectorilor sau structurilor, pentru a determina adresa următorului element.

Dacă argumentul este de tip etichetă, operatorul, în funcție de tipul etichetei, returnează valorile:

near - ffffH;
far - fffeH;

Exemple:

```
var     dw          .....
mov     bx,     type var ; (BX) = 2
vector  dd      10 dup (?)
mov     si,     type vector ; (SI) = 4
num_BCD     dt          13245768,.....
mov     bx,     type num_BCD; (BX) = 10
```

e) *LENGTH < expresie >*

Acest operator furnizează o valoare întreagă ce reprezintă numărul de elemente ale unei variabile (declarată utilizând operatorul DUP). Dacă variabila a fost declarată prin utilizarea operatorului DUP imbricat (unul inclus într-altul), atunci operatorul LENGTH va returna o valoare asociată operatorului exterior. Dacă variabila nu a fost declarată utilizând operatorul DUP, atunci operatorul LENGTH va returna valoarea 1.

Exemple:

```
n1          db          50 dup (?);
n2          dw          150 dup (0, 1, ?);
n3          dd          200 dup (10, 20, 15 dup (?));
mes     db          'Exemplu de mesaj';

length n1 = 50;
length n2 = 150;
length n3 = 200;
length mes = 1;
```

f) *SIZE < expresie >*

Furnizează o valoare ce reprezintă numărul de octeți ocupați de o variabilă şi este corelat cu operatorii LENGTH şi TYPE prin identitatea:

SIZE = LENGTH * TYPE

Dacă o variabilă a fost declarată utilizând operatorul DUP imbricat, SIZE furnizează, conform relației anterioare, numai valoarea operatorului DUP exterior. Pentru declarațiile anterioare, acest operator va furniza următoarele valori:

size n1 = 50;
size n2 = 300;
size n3 = 800;
size mes = 1;

g) *PTR < expresie >*

Acest operator realizează conversia de tip pentru o variabilă, un operand memorie sau o etichetă fie pentru a specifica tipul adresării, fie pentru a permite adresarea unor date cu un alt tip decât utilizat la declararea acestora. Sintaxa acestui operator este:

tip PTR < expresie >

unde tip poate fi BYTE, WORD, DWORD, QWORD, PWORD, FWORD, TBYTE sau nume de structură pentru operanzi din memorie şi variabile, respectiv NEAR, FAR, PROC sau unknown pentru etichete de instrucțiuni, nume de proceduri.

De fapt, acest operator asociază următoarele atribute expresiei:

Tip expresie	Atribute asociate		
	Segment	Offset	Type
Variabilă sau etichetă	SEG <expresie>	OFFSET <espresie >	tip
Expresie numerică	nedefinit	<expresie>	tip

Exemple:

i) Explicitarea tipului referinței la memorie, pentru referințe anonime:

```
inc byte ptr [bx]
inc word ptr [si]
mov byte ptr [bx], 99
mov word ptr [di], 100
and word ptr [bp], 100h
```

ii) Specificarea tipului de salt (intersegment sau intrasegment):

```
jmp dword ptr [bx]
jmp near ptr etich      ; pentru ‘etich’ declarata in alt modul, daca
                        ; aceasta eticheta este in acelasi segment sau
                        ; grup de segmente cu instructiunea de salt
```

iii) Referirea unor variabile cu un alt tip decât cel utilizat la declararea acestora:

```
aw dw 0abcH, 12abH
ab db 0aH, 12H
mov al, byte ptr aw
mov ax, word ptr ab
```

iv) Crearea unei variabile anonime la un offset dat dintr-un segment:

```
mov al, ds: byte ptr 5              ; face referire la octetul de la ds:5
mov bx, data1: word ptr 3000H       ; face referire la cuvantul de la
                                    ; offsetul 3000H din segmentul data1
```

Aceste două declarații sunt echivalente cu următoarele două:

```
mov al, ds:[5]
mov bx, data1:[3000]
```

h) *SHORT*

Acest operator acceptă un argument de tip etichetă (un offset adresabil prin registrul segment CS). Se utilizează în instrucțiuni de salt condiționat, necondiționat şi în instrucțiuni de apel de procedură, când codul țintă are un deplasament autorelativ de un octet cu semn (adică ținta saltului se află în intervalul -128 ÷ 127 față de instrucțiunea de salt). În mod normal, o instrucțiune de salt va codifica deplasamentul la instrucțiunea țintă ca un număr întreg, pe 16 biți. Utilizând acest operator, cu condiția ca ținta să se găsească în intervalul [-128, 127], asamblorul va codifica deplasamentul țintei pe un singur octet, reducând cu un octet codul generat:

```
jmp     short etich
```

De fapt, asamblorul generează în mod automat un deplasament pe 8 biți dacă deplasamentul autorelativ al țintei este în intervalul [-128,+127], dar pentru referințe externe (adică pentru etichete definite în alte module, de tipul near) va genera un deplasament pe 16 biți. Dacă referința se află în intervalul menționat, atunci utilizând operatorul SHORT ca în exemplul anterior, se va genera un deplasament de un octet, reducând cu un octet codul generat pentru instrucțiunea de salt.

i) *WIDTH, MASK*

Aceşti operatori sunt folosiți pentru a returna numărul de biți sau o mască de biți pentru o înregistrare care reprezintă un tip de date structurat.

Operatorul WIDTH returnează numărul de biți ai unei înregistrări sau al unui câmp al unei înregistrări, după cum argumentul său este numele înregistrării sau numele unui câmp al înregistrării.

Operatorul MASK are ca argument numele unui câmp al unei înregistrări, şi returnează o mască de biți, definiți ca fiind 1 pentru pozițiile câmpului respectiv, respectiv zero pentru celelalte poziții. Exemplu:

```
model  record  A:3, B:1, C:4, D:5, E:3; reprezinta numele
                                    ; asociate campurilor respective, impreuna
                                    ; cu numarul de biti pentru fiecare camp
mov cx, mask C                      ; (CX) = 000 0 1111 00000 000 = 0f00H
mov cl, width D                     ; (CL) = 5, dimensiunea campului D
mov cl, C                           ; (CL) = 8, pozitia campului C
```

Prioritatea operatorilor

Ordinea de evaluare a unei expresii este determinată de prioritățile asociate operatorilor. Operatorii cu prioritate mai mare sunt evaluați primii. Expresiile cu aceeaşi prioritate sunt

evaluate de la stânga la dreapta. Ordinea de evaluare poate fi modificată utilizând paranteze, în care caz expresiile dintre paranteze se vor evalua primele. Dacă expresiile conțin paranteze pe mai multe niveluri, atunci evaluarea începe cu prima paranteză interioară. Prioritatea operatorilor este următoarea:

1. length, size, width, mask, (), [ ], < >, operatorul câmp de structură;
2. seg, offset, type, ptr, prefix segment : ;
3. high, low;
4. +, – (operatorii unari);
5. *, /, mod, shl, shr;
6. +, – (operatori binari);
7. eq, ne, gt, lt, ge, le (operatori relaționali);
8. not;
9. and;
10. or, xor;
11. short

Definirea şi inițializarea etichetelor

Etichetele identifică locații de memorie ale instrucțiunilor, operanzilor sau apeluri de proceduri. Ele au trei atribute: segment, deplasament (offset) şi distanță (near sau far) pentru etichete de instrucțiune şi nume de proceduri, sau tip (byte, word etc.) pentru etichete de operanzi.

Etichetele de tip near pot fi definite astfel:

```
etich_near: mov ax, bx
proc_alfa   label near
calcul      proc near
```

În acest ultim caz, dacă nu se specifică tipul, în mod implicit acesta va fi tot de tip near.

Exemple pentru tipul far:

```
beta        proc        far
etich_far   label       far
```

Această directivă nu incrementează contorul de program ($).

În cazul unei etichete de operand se va specifica tipul acesteia, ca de exemplu în cazul declarării vârfului stivei din segmentul de stivă:

```
stiva       segment
dw          100 dup (?)
varf_stiva  label       word
stiva       ends
```

pentru a permite, ulterior, inițializarea registrului SP cu vârful stivei.

Directiva EQU

Această directivă permite atribuirea unei valori, în momentul asamblării, la un simbol. Sintaxa este:

```
<nume>      EQU      <expresie>
```

unde expresia poate fi:

■ constantă:

```
val1        equ     1
valc        equ     5*5
```

■ orice expresie corectă în limbaj de asamblare:

```
inreg       record  A:5, B:5, C:5
v321        equ     inreg < 3, 2, 1 >
e1          equ     (mask a) or (0f0h and mask b)
e2          equ     valc        mod         10
```

■ simbol definit anterior:

```
unu         equ     val1
```

■ şir de caractere, cu diferite semnificații:

- referință indexată:

```
rbp         equ     [ bp + 8 ]
```

care poate fi apoi folosit astfel:

```
mov         ax, rbp.camp
```

- operator prefix segment şi operanzii săi:

```
rdsbp8 equ  ds:[bp + 8]
```

- nume de instrucțiuni:

```
IDAI  equ   AAD     ; se poate apoi inlocui in program mnemonica
                    ; AAD cu cea nou definita.
```

- şir de caractere:

```
octet equ <db> ; se poate folosi in pr. in loc de db noul nume: octet;
sir_car     equ     <'mesaj', 0DH, 0AH, '$'>
```

sau

```
sir_car     equ     'mesaj', 0DH, 0AH, '$'
```

Observație: parantezele unghiulare trebuie utilizate la redefinirea pseudoinstrucțiunilor, pentru a le delimita precis.

În mod asemănător, pentru valori numerice se mai poate folosi şi operatorul =, cu sintaxa:

```
<nume> = <expresie>
```

Diferența dintre cei doi operatori este că numele simbolice utilizând operatorul EQU nu îşi pot modifica valoarea pe durata asamblării programului, pe când identificatorii definiți cu operatorul = îşi pot modifica valoarea, adică se pot utiliza declarații de forma:

```
n = 0
.....
```

n = n + 1

Operatorul *THIS*

Acest operator creează un operand cu adresa de segment şi offset egală cu aceea a locației curente a contorului program. Formatul general este:

this tip

unde specificatorul *tip* poate fi BYTE, WORD, DWORD, QWORD sau TBYTE pentru operanzi din memorie şi variabile, respectiv NEAR, FAR, PROC pentru etichete. El poate fi utilizat, de obicei, cu directivele *EQU* sau semnul =. Exemplul următor defineşte eticheta *param*:

param equ this byte

Efectul acestei declarații este acelaşi cu acela al utilizării directivei *LABEL*:

param label byte

Simbolul contor program $ (contor locații)

Simbolul $ reprezintă adresa curentă a contorului program, adică adresa relativă în cadrul segmentului curent a instrucțiunii sau datelor de asamblat. La începutul fiecărui segment contorul de locații este inițializat la zero şi este actualizat de asamblor pe măsura asamblării programului.

Exemple:

```
lista       dw      11, 22, 33, 0aah
lung_lista  equ     ($ – lista)/2
mesaj       db      ‘ mesaj $ ‘
lung_mesaj  equ     $ – mesaj
```

În ambele exemple s-a determinat numărul de elemente, respectiv de cuvinte/octeți, din cele două declarații utilizând acest simbol. Să nu se confunde simbolul $ cu caracterul „$“.

Directiva ORG

Cu această directivă, contorul de locații poate fi inițializat cu o anumită valoare pozitivă, astfel:

ORG < expresie >

unde expresia este evaluată modulo 64K şi nu trebuie să conțină referințe ulterioare.

Exemple:

```
org     100h            ; ce urmeaza in continuare va fi depus in
                        ; segmentul respectiv de la adresa relativa 100h
org     offset $ + 100  ; se rezerva, de la ultimul octet asamblat,
                        ; 100 octeti neinitializati
```

Directiva nu poate fi atribuită la o etichetă, adică o declarație de forma

```
start:  org             100h
```

va genera o eroare. În schimb putem proceda astfel:

```
        org         100h
        start:      ....
```

sau pentru date:

```
        date        label       byte
        org         $ + 100
```

şi în acest fel se rezervă 100 de octeți de la adresa specificată.

Observație: o declarație de forma

```
        org         offset $ – 100
```

nu are sens, deoarece înseamnă că datele care urmează după această directivă vor fi scrise peste ultimii 100 de octeți. (definiți anterior).