

ARHITECTURA SISTEMELOR DE CALCUL

– Seminar 2 –

1. Obținerea offsetului/valorii unei variabile
2. Ordinea de plasare a octeților în memorie
3. Instrucțiuni cu/fără semn
4. Instrucțiuni aritmetice pentru înmulțire/împărțire (cu/fără semn)
5. Instrucțiuni de conversie cu/fără semn
6. Instrucțiuni aritmetice care țin cont de transport
7. Instrucțiuni de lucru cu stiva

1. OBȚINEREA OFFSETULUI/VALORII UNEI VARIABILE

Dacă variabila `a` este un dublucuvânt (`a dd 12345678h`), atunci:

| Instrucțiunea | Efect |
|---------------------------|--|
| <code>mov eax, a</code> | EAX = OFFSET-ul (32 de biți) la care este stocat variabila <code>a</code> |
| <code>mov eax, [a]</code> | EAX = VALOAREA variabilei <code>a</code> (dublucuvântul care începe de la offset-ul <code>a</code>) |

2. ORDINEA DE PLASARE A OCTEȚILOR ÎN MEMORIE

Reprezentarea în memorie a datelor a căror dimensiune depășește un octet se poate realiza în două moduri distincte:

- plasarea little-endian, în care octetul cu cea mai mică adresă din locația de memorie respectivă va conține octetul cel mai puțin semnificativ al reprezentării (octetul "*end*" al reprezentării are adresa cea mai "*little*");
- plasarea big-endian, în care octetul cu cea mai mare adresă din locația de memorie respectivă va conține octetul cel mai puțin semnificativ al reprezentării (octetul "*end*" al reprezentării are adresa cea mai "*big*").

Discuție

Spre exemplu, dacă dorim să reprezentăm numărul $1025_{(10)}$ într-o locație de 4 octeți, procedăm astfel:

- convertim numărul în bazele 16 și 2:

$$1025_{(10)} = 00000401_{(16)} = 00000000 \ 00000000 \ 00000100 \ 00000001_{(2)}$$

- luând în considerare cele două moduri posibile, ordinea de plasare a octeților în memorie va fi:

| | | b | b + 1 | b + 2 | b + 3 |
|----------------------|--|----------------|----------------|----------------|----------------|
| Big-endian | Octetul " <i>end</i> " al reprezentării are adresa cea mai " <i>big</i> " | 00 00000000 | 00 00000000 | 04 00000100 | 01 00000001 |
| Little-endian | Octetul " <i>end</i> " al reprezentării are adresa cea mai " <i>little</i> " | 01 00000001 | 04 00000100 | 00 00000000 | 00 00000000 |

Modul de plasare a octeților în memorie poate să difere de la un sistem de operare la altul. Familia de sisteme de operare Windows utilizează plasarea little-endian.

Exemplu

Se dă următorul segment de date:

```
segment data use32 class=data
a1 db 2, 4, 6, 8
a2 dw 2, 4, 6, 8
a3 dd 2, 4, 6, 8
a4 db '2', '4', '6', '8'
```

```

a5 db 24h, 68h
a6 dw 24h, 68h
a7 dd 24h, 68h
a8 db '24', '68'
a9 dw '24', '68'
a10 dw '2', '4', '6', '8'

a11 db 2468h
a12 dw 2468h
a13 dd 2468h
a14 dd 02040608h, 01030507h

```

Cum va fi reprezentat în memorie segmentul de date de mai sus ?

Obs: In cazul initializarii unei zone de memorie cu valori de tip constante string (sizeof > 1) tipul de data utilizat in definire (dw, dd, dq) are rol doar de rezervare a spatiului dorit, ordinea de “umplere” a zonei de memorie respective fiind ordinea in care apar caracterele (octetii) in cadrul constantei de tip string.

3. INSTRUCȚIUNI CU/FĂRĂ SEMN

Dacă ținem cont de reprezentarea numerelor cu/fără semn, în arhitectura IA-32, există trei tipuri de instrucțiuni:

- instrucțiuni care nu țin cont de reprezentarea cu/fără semn a numerelor: *mov add sub*
- instrucțiuni care interpretează operanzii ca fiind numere fără semn: *div mul*
- instrucțiuni care interpretează operanzii ca fiind numere cu semn: *idiv imul cbw cwd cwde cdq*

Este important ca programatorul să fie consistent atunci când programează în limbajul IA-32:

- dacă consideră toate valorile numerice ca fiind pozitive, atunci trebuie să folosească doar instrucțiuni de tipul **a** și **b**;
- dacă consideră toate valorile numerice ca fiind numere cu semn, atunci trebuie să folosească doar instrucțiuni de tipul **a** și **c**.

Observații

Atunci când se folosesc instrucțiuni cu doi operanzi trebuie să se țină cont de următoarele:

- ambii operanzi trebuie să aibă aceeași dimensiune de reprezentare (de exemplu putem aduna un octet cu un alt octet, dar nu un octet cu un cuvânt sau un octet cu un dublucuvânt);
- cel puțin un operand trebuie să fie un registru de uz general sau o valoare imediată (constantă);
- dacă operandul este constantă, acesta nu poate să fie operandul destinație.

Discuție

Calculați suma valorilor a și b definite în segmentul de date. Analizați secvențele de instrucțiuni de mai jos și explicați efectul fiecărei linii.

```

; segmentul de date
segment data use32 class=data
    a db 10                ; 10 = 0Ah
    b db 11                ; 11 = 0Bh

; segmentul de date
segment code use32 class=code
    start:
        add [a], [b]      ; eroare la asamblare
        ...
        mov ax, [a]       ; instructiunea e corectă sintactic, dar incorectă logic
        add ax, [b]       ; instructiunea e corectă sintactic, dar incorectă logic
        ...

```

- instrucțiunea

```
add [a], [b]
```

va produce eroare la asamblare, iar fișierul executabil nu va fi creat, deoarece operanzii acestei instrucțiuni nu respectă cea de-a doua constrângere de mai sus;

– deși instrucțiunile:

```
mov ax, [a]
add ax, [b]
```

sunt incorecte logic, asamblorul nu va semnaliza eroare de sintaxă deoarece dimensiunea operandului sursă este dedusă din dimensiunea operandului destinație.

Astfel, după execuția primei instrucțiuni, în registrul **AX** vom avea cuvântul din memorie care începe la offset-ul a (cuvântul compus din octeții aflați la offset-ul a și a+1), adică **AX=0B0Ah**.

Cea de-a doua este incorectă logic pentru că se efectuează adunarea dintre un cuvânt (în registrul **AX**) și valoarea cuvântului care începe la adresa b, adică **0B0Ah + ??0Bh** (?? Semifică faptul că nu avem control asupra datelor care se regăsesc în segmentul de date la adresa b+1).

Secvențele de instrucțiuni care realizează corect suma valorilor a și b definite în segmentul de date sunt:

```
; segmentul de date
segment data use32 class=data
a db 10                ; 10 = 0Ah
b db 11                ; 11 = 0Bh
; segmentul de date
segment code use32 class=code
start:
...
mov al, [a]
add al, [b]
...
```

4. INSTRUCȚIUNI ARITMETICE PENTRU ÎNMULȚIRE/ÎMPĂRȚIRE (CU/FĂRĂ SEMN)

MUL – înmulțire fără semn

Sintaxa: mul op

unde op poate fi un registru sau o variabilă de tip octet, cuvânt sau dublucuvânt

Efect:

| Dimensiune op | | Registru implicit | | Rezultat operație |
|----------------------------|---|---|---|---|
| 1 octet (8 biți) | x | AL (1 octet = 8 biți) | = | AX (2 octeți = 16 biți) |
| 1 cuvânt (16 biți) | x | AX (1 cuvânt = 16 biți) | = | DX (cuvântul superior) AX (cuvântul inferior) |
| 1 dublucuvânt (32 biți) | x | EAX (1 dublucuvânt = 32 biți) | = | EDX (dublucuvântul superior) EAX (dublucuvântul inferior) |

Exemplu

Instrucțiunea

```
mul dx
```

va înmulți cuvântul aflat în registrul **DX** cu cuvântul aflat în registrul **AX**.

Rezultatul operației va fi un număr reprezentat pe 32 de biți (1 dublucuvânt) și va fi stocat în doi regiștri **DX:AX** din motive de compatibilitate cu arhitecturile Intel 8086 precedente.

Dacă presupunem că rezultatul înmulțirii este numărul 12345678h, atunci cuvântul inferior (cel mai puțin semnificativ) va fi stocat în registrul **AX** (**AX** = 5678h), iar cuvântul superior (cel mai semnificativ) va fi stocat în registrul **DX** (**DX** = 1234h).

DIV – împărțire fără semn

Sintaxa: div op

unde op poate fi un registru sau o variabilă de tip octet, cuvânt sau dublucuvânt

Efect:

| Deîmpărțit | | Dimensiune op | | Cât | | Rest |
|---|---|----------------------------|---|---|--|---|
| AX (1 cuvânt = 16 biți) | : | 1 octet (8 biți) | = | AL (1 octet = 8 biți) | | AH (1 octet = 8 biți) |
| DX:AX (1 dublucuvânt = 32 biți) | : | 1 cuvânt (16 biți) | = | AX (1 cuvânt = 16 biți) | | DX (1 cuvânt = 16 biți) |
| EDX:EAX (1 quadword = 64 biți) | : | 1 dublucuvânt (32 biți) | = | EAX (1 dublucuvânt = 32 biți) | | EDX (1 dublucuvânt = 32 biți) |

IMUL și **IDIV** reprezintă variantele cu semn a instrucțiunilor MUL și DIV (operandii sunt interpretați ca numere cu semn).

Exemplu

```
mov ax,0180h ; ax = 0180h
mul ah       ; ax = al * ah = 80h * 01h = 128 * 1 = 128 = 0080h

mov ax,0180h ; ax = 0180h
imul ah      ; ax = al * ah = 80h * 01h = -128 * 1 = -128 = FF80h

mov ax,0080h ; ax = 0080h
div al       ; ax / al = 0080h / 80h = 128 / 128 => al=01h si ah=00h

mov ax,0080h ; ax = 0080h
idiv al      ; ax / bl = 0080h / 80h = 128 / (-128) => al=FFh si ah=00h

mov ax,512   ; ax = 0200h
mov bl,1     ; bl=01h
div bl       ; division overflow 512 / 1 = 512 r 0 si 512 nu incapa pe un byte
```

5. INSTRUCȚIUNI DE CONVERSIE CU/FĂRĂ SEMN

5.a. Instrucțiuni de conversie fără semn

Nu există instrucțiuni de conversie fără semn.

Conversiile fără semn se realizează prin „zerorizarea” octetului, cuvântului sau dublucuvântului superior.

Exemple

```

; segmentul de date
segment data use32 class=data
a db 10
b dw 1122h
c dd 11223344h

; segmentul de cod
segment code use32 class=code
start:
    ; Să se calculeze a+b, a - byte, b - word
    ; BYTE -> WORD
    mov al, [a]      ; AL = 00001010b
    mov ah, 0        ; AX = AH:AL = 00000000:00001010b (extindere fără semn)
    add ax, [b]      ; AX = 000Ah + 1122h

    ; Să se calculeze b / 234h, b - word
    ; WORD -> DWORD
    mov ax, [b]      ; AX = 1122h
    mov dx, 0        ; DX:AX = 0000:1122h (extindere fără semn)
    mov bx, 234h     ; BX = 0234h
    div bx           ; AX = catul impartirii 00001122h / 0234h si
                    ; DX = restul impartirii 00001122h / 0234h

    ; Să se calculeze b+c, b - word, c - dword
    ; WORD -> DWORD EXTENDED
    mov eax, 0       ; EAX = 00000000h
    mov ax, [b]      ; EAX = 00001122h
    add eax, [c]     ; EAX = 00001122h + 11223344h

    ; Să se calculeze c / 45678h, c - doubleword
    ; DWORD -> QUADWORD
    mov eax, [c]     ; EAX = 11223344h
    mov edx, 0       ; EDX:EAX = 00000000:11223344h (extindere fără semn)
    mov ebx, 45678h ; EBX = 00045678h
    div ebx          ; EAX = catul impartirii 0000000011223344h / 45678h
                    ; EDX = restul impartirii 0000000011223344h / 45678h

```

5.b. Instrucțiuni de conversie cu semn

CBW

Sintaxa: cbw

Efect: convertește cu semn BYTE-ul din AL la un WORD în AX

Instrucțiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia $AL \rightarrow AX$.

Conversia se realizează prin extinderea reprezentării de pe 8 biți pe 16 biți, prin completarea cu bitul de semn în fața octetului inițial.

CWD

Sintaxa: cwd

Efect: convertește cu semn WORD-ul din AX la un DWORD în DX:AX

Instrucțiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia $AX \rightarrow DX:AX$.

Conversia se realizează prin extinderea reprezentării de pe 16 biți pe 32 biți, prin completarea cu bitul de semn în fața cuvântului inițial.

CWDE

Sintaxa: cwde

Efect: convertește cu semn WORD-ul din AX la un DWORD în EAX

Instrucțiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia $AX \rightarrow EAX$.

Conversia se realizează prin extinderea reprezentării de pe 16 biți pe 32 biți, prin completarea cu bitul de semn în fața cuvântului inițial.

CDQ

Sintaxa: cdq

Efect: convertește cu semn DWORD-ul din EAX la un QUADWORD în EDX:EAX

Instrucțiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia $EAX \rightarrow EDX:EAX$.

Conversia se realizează prin extinderea reprezentării de pe 32 biți pe 64 biți, prin completarea cu bitul de semn în fața dublucuvântului inițial.

Exemple:

```
mov AX, 0080h ; AX = 0080h
mov BX, -3    ; BX = FFFDh
cbw          ; AL -> AX => AX = FF80h
imul AH      ; AX = AL * AH = 80h * FFh = (-128) * (-1) = 128 = 0080h
cwd          ; DX:AX = 0000:0080h
idiv BX      ; AX = 00000080h / FFFDh = 128 / (-3) = -42 = FFD6h
              ; DX = 00000080h % FFFDh = 128 % (-3) = 2
```

6. INSTRUCȚIUNI ARITMETICE CARE ȚIN CONT DE TRANSPORT

Există situații în care valorile unor variabile/rezultate se găsesc jumătate într-un registru și jumătate în altul. În aceste cazuri, poate fi convenabil să adunăm/scădem pe două etape: adunăm/scădem mai întâi regiștrii care conțin parte inferioară a reprezentării, apoi pe cei care conțin parte superioară a acestora. Însă rezultatul operației nu va fi corect, dacă, în a doua etapă, nu ținem cont de un eventual transport/împrumut generat de operația efectuată în prima etapă.

CF (Carry Flag) este flagul de transport. CF va avea valoarea 1 dacă în cadrul ultimei operații efectuate (UOE) s-a efectuat transport în afara domeniului de reprezentare a rezultatului și valoarea 0 în caz contrar.

Instrucțiunile aritmetice care țin cont de transport sunt **ADC** și **SBB**.

ADC (Add with Carry)

Sintaxa: **adc** d, s

unde:

- d poate fi un registru sau o locație de memorie
- s poate fi o valoare imediată (constantă), un registru sau o locație de memorie

Efect: $d \leftarrow d + s + \mathbf{CF}$ (**Carry Flag**)

SBB (Subtract with Borrow)

Sintaxa: **sbb** d, s

unde:

- d poate fi un registru sau o locație de memorie
- s poate fi o valoare imediată (constantă), un registru sau o locație de memorie

Efect: $d \leftarrow d - s - \mathbf{CF}$ (**Carry Flag**)

Exemple: sem2_byte+word.asm, sem2_word+dword.asm, sem2_dword+qword.asm

7. INSTRUCȚIUNI DE LUCRU CU STIVA

Orice program care se execută utilizează o stivă de execuție (*execution stack, run-time stack*).

Stiva este o structură de date care funcționează pe principiul LIFO (Last-In-First-Out). Singurul element direct accesibil este cel aflat în vârful stivei.

Segmentul de memorie în care este localizată stiva este indicat de către registru **SS** (Stack Segment), iar offset-ul locației de memorie aflate în vârful stivei se găsește în registru **(E)SP** (Stack Pointer).

Cele mai utilizate instrucțiuni de lucru cu stiva sunt **PUSH** (pune un element în stivă) și **POP** (extrage un element din stivă).

PUSH

Sintaxa: **push** source

unde source poate fi o valoare imediată, un registru sau o locație de memorie pe 16 sau 32 biți

Efect:

a. dacă source este de tip cuvânt (16 biți):

$ESP \leftarrow ESP - 2$

$[SS : ((ESP + 1) : ESP)] \leftarrow source$

b. dacă source este de tip dublucuvânt (32 biți):

$ESP \leftarrow ESP - 4$

$[SS : ((ESP + 2) : ESP)] \leftarrow source$

POP

Sintaxa: **pop** destination

unde destination poate fi un registru sau o locație de memorie pe 16 sau 32 biți

Efect:

a. dacă destination este de tip cuvânt (16 biți):

$destination \leftarrow [SS : ((ESP + 1) : ESP)]$

$ESP \leftarrow ESP + 2$

b. dacă destination este de tip dublucuvânt (32 biți):

$destination \leftarrow [SS : ((ESP + 2) : ESP)]$

$ESP \leftarrow ESP + 4$

Observații:

- pe/din stivă pot fi puse/extrase doar cuvinte (16 biți) sau dublucuvinte (32 biți)
- stiva crește invers de la adrese de memorie mari către adrese de memorie mici
- (E)SP va indica offset-ul celui mai puțin semnificativ octet al elementului aflat în vârful stivei

Exemplu: sem2_utilizare_stiva.asm

EXERCITII

Scrieți un program în limbaj de asamblare care să calculeze expresia aritmetică, considerând domeniile de definiție ale variabilelor:

1. $x = [(a + b) * c] / d$, unde a, b, c, d – byte
2. $x = (a - b * c) / d$, unde a, b, c, d – byte
3. $x = (a * b) / d - c$, unde a, c, d – byte, b – word