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

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 *"end"* al reprezentării are adresa cea mai *"big"* | **00**  00000000 | **00**  00000000 | **04**  00000100 | **01**  00000001 |
|  |  |  |  |  |  |
| ***Little−endian*** | Octetul *"end"* al reprezentării are adresa cea mai *"little"* | **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 IN MEMORIE

a1 db 2, 4, 6, 8 02 04 06 08

a2 dw 2, 4, 6, 8 02 00 04 00 06 00 08 00

a3 dd 2, 4, 6, 8 02 00 00 00 04 00 00 00 06 00 00 ...

a4 db '2', '4', '6', '8' – caractere 32 34 36 38

a5 db 24h, 68h 24 68

a6 dw 24h, 68h 24 00 68 00

a7 dd 24h, 68h 24 00 00 00 68 00 00 00

a8 db '24', '68' 32 34 36 38

a9 dw '24', '68' 32 34 36 38

a10 dw '2', '4', '6', '8' 32 00 34 00 36 00 38 00

a11 db 2468h 68 <- valoare trunchiata

a12 dw 2468h 68 24

a13 dd 2468h 68 24 00 00

a14 dd 02040608h, 01030507h 08 06 04 02 07 05 03 01

3. INSTRUCȚIUNI CU/FĂRĂ SEMN

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

**a**. instrucțiuni care nu țin cont de reprezentarea cu/fără semn a numerelor: *mov add sub*

**b**. instrucțiuni care interpretează operanzii ca fiind numere fără semn: *div mul*

**c**. 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

Fie segmentul de date și secvențele de instrucțiuni de mai jos:

|  |
| --- |
| ; segmentul de date  segment data use32 class=data  a db 10  b db 11  ; segmentul de date  segment code use32 class=code  start:  add [a], [b] ; eroare la asamblare  ...  mov ax, [a] ; incorect pentru că AX are 16 biți  add ax, [b] ; incorect pentru că se adună AX (16 biți) + b (8 biți)  ... |

* 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, asamblorul nu va semnala 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).

Cea de-a doua este incorectă pentru că se efectuează adunarea dintre un cuvânt (în registrul AX) și un octet (valoarea variabilei 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 NU O CONSTANTA

*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 registru 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 *varianta cu semn a instrucțiunilor MUL si DIV* (operanzii sunt interpretați ca numere cu semn).

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:  ; BYTE -> WORD  mov al, [a] ; AL = 00001010  mov ah, 0 ; AX = AH:AL = 000000000:00001010 (extindere fără semn)  ; WORD -> DWORD  mov ax, [b] ; AX = 1122h  mov dx, 0 ; DX:AX = 0000:1122h (extindere fără semn)  ; DWORD -> QUADWORD  mov eax, [c] ; EAX = 11223344h  mov edx, 0 ; EDX:EAX = 00000000:11223344h (extindere fără semn) |

5.b. Instrucțiuni de conversie cu semn

**CBW**

*Sintaxa:* cbw

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

Instrucţiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia AL → AX.

Conversia se realizează prin extinderea reprezentării de pe 8 biţi pe 16 biţi, prin completarea cu bitul de semn în faţa octetului iniţial.

**CWD**

*Sintaxa:* cwd

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

Instrucţiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia AX → DX:AX.

Conversia se realizează prin extinderea reprezentării de pe 16 biţi pe 32 biţi, prin completarea cu bitul de semn în faţa cuvântului iniţial.

**CWDE**

*Sintaxa:* cwde

*Efect:* converteşte cu semn WORD-ul din AX la un DWORD în EA X

Instrucţiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia AX → EAX.

Conversia se realizează prin extinderea reprezentării de pe 16 biţi pe 32 biţi, prin completarea cu bitul de semn în faţa cuvântului iniţial.

**CDQ**

*Sintaxa:* cdq

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

Instrucţiunea nu are operanzi, deci va realiza ÎNTOTDEAUNA conversia EAX → EDX:EAX.

Conversia se realizează prin extinderea reprezentării de pe 32 biţi pe 64 biţi, prin completarea cu bitul de semn în faţa dublucuvântului iniţial.

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 acesteia. 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 operatii efectuate (UOE) s-a efectuat transport în afara domeniului de reprezentare a rezultatului și valoarea 0 in 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 ← d + s **+ CF (Carry Flag)**

**SBB (SuBstract 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 ← d - s **- CF (Carry Flag)**

Exemple: sem2\_byte+word.asm, sem2\_word+dword.asm, sem2\_dword+qword.asm

DX : AX CX:BX

push dx push cx ADD ax, bx SUB ax, bx

push ax push bx SAU ADC dx, cx SBB dx, cx

pop eax pop ebx

ADD eax, ebx

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 registrul SS (Stack Segment), iar offset-ul locației de memorie aflate în vârful stivei se găsește în registrul (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 ← ESP - 2

**[SS:**((ESP + 1):ESP)**]** ← source

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

ESP ← ESP - 4

**[SS:**((ESP + 2):ESP)**]** ← 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 ← **[SS:**((ESP + 1):ESP)**]**

ESP ← ESP + 2

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

destination ← **[SS:**((ESP + 2):ESP)**]**

ESP ← 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 vărful stivei

Exemplu: sem2\_stiva.asm

EXERCIȚII

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

bits 32

global start

extern exit

import exit msvcld.dll

segment data use 32 class = data

a db 1

b db 2

c db 3

d db 4

x resb 1 ; se rezerva un octet

segment code use 32 class = code

mov al, [a]

add al, [b]

mul **byte**[c] ; specificare dimensiune operatie

mov bl, [d] / = div **byte**[d]

div bl /

mov [x], al ; x = catul operatiei

2. x = (a − b \* c) / d, unde a, b, c, d − byte

bits 32

global start

extern exit

import exit msvcld.dll

segment data use 32 class = data

a db 1

b dw 2

c db 3

d db 4

x resb 1 ; se rezerva un octet

segment code use 32 class = code

mov al, [b]

mul byte[c]

push ax ; mov bx, ax

mov al, [a]

mov ah, 0

pop bx

sub ax, bx

div byte[d]

mov [x], ax

3. x = (a \* b) / d − c , unde a, c, d − byte, b − word