Arhitectura Sistemelor de Calcul

Lect. Dr. Șotropa Diana
diana.sotropa@ubbcluj.ro

Facultatea de Matematică și Informatică
Universitatea Babeș-Bolyai

Instrucțiuni

Instrucțiuni

Instrucțiuni de conversie (distructivă)

Instrucțiune	Efect
`CBW`	conversie octet conţinut în AL la cuvânt în AX (extensie de semn) `mov al, -1` ; AL = 0FFh `cbw` ;extinde valoarea octet -1 din AL în valoarea cuvânt -1 din AX (0FFFFh).
`CWD`	conversie cuvânt conţinut în AX la dublu cuvânt în DX:AX (extensie de semn) `mov ax,-10000` ; AX = 0D8F0h `cwd` ;obţine valoarea -10000 în DX:AX (DX = 0FFFFh ; AX = 0D8F0h) cwde ;obţine valoarea -10000 în EAX (EAX = 0FFFFD8F0h)
`CWDE`	conversie cuvânt din AX în dublucuvânt în EAX (extensie de semn)
`CDQ`	conversie cuvânt din EAX în dublucuvânt în EDX:EAX (extensie de semn)
`MOVZX d,s`	încarcă în d (REGISTRU !), de dimensiune mai mare decât s (registru sau operand din memorie!), conţinutul lui s fară semn (zero extension)
`MOVSX d,s`	încarcă în d (REGISTRU !), de dimensiune mai mare decât s (registru sau operand din memorie!), continutul lui s cu semn (sign extension) http://www.c-jump.com/CIS77/ASM/DataTypes/T77_0270_sext_example_movsx.htm

Instrucțiuni

Instrucțiuni de conversie (distructivă)

- Conversia fără semn se realizează prin zerorizarea octetului sau cuvântului superior al valorii de la care s-a plecat. (de exemplu, prin mov ah,0 sau mov dx,0 – efect similar se obţine prin aplicarea instr. MOVZX)

De ce coexistă CWD cu CWDE ?

- CWD trebuie să rămână din raţiuni de backwards compatibility şi din raţiuni de funcţionalitate a intrucţiunilor (I)MUL şi (I)DIV.
- Exemple:

```
MOV ah, 0c8h
MOVSX ebx, ah ; EBX = FFFFFFC8h
MOVZX edx, ah ; EDX = 000000C8h
MOVSX ax,[v] ; MOVSX ax, byte ptr DS:[offset v]
MOVZX eax, [v] ; syntax error – op.size not specified
```

- Atenţie ! NU sunt acceptate sintactic:

```
CBD         CWDE EBX, BX        MOVSX EAX, [v]
CWB         CWD EDX,AX          MOVZX EAX, [EBX]
CDW         MOVZX AX, BX        MOVSX dword [EBX], AH
CDB         MOVSX EAX, -1       CBW BL
```

Instrucțiuni

Operații aritmetice

- Operanzii sunt reprezentaţi în cod complementar. Microprocesorul realizează adunările şi scăderile "văzând" doar configuraţii de biţi şi nu numere cu semn sau fără.
- Regulile de efectuare a adunării şi scăderii presupun adunarea de configuraţii binare, fără a fi nevoie de a interpreta operanzii drept cu semn sau fără semn anterior efectuării operaţiei!
- Deci, la nivelul acestor instrucţiuni, interpretarea "cu semn" sau "fără semn" rămâne la latitudinea programatorului, nefiind nevoie de instrucţiuni separate pentru adunarea/scăderea cu semn faţă de adunarea/scăderea fără semn. Adunarea şi scăderea se efectuează întotdeauna la fel (adunând sau scăzând configuraţii binare) indiferent de semnul (interpretarea) acestor configuraţii! După cum vom vedea acest lucru nu este valabil şi pentru înmulţire şi împărţire.
- În cazul acestor operaţii trebuie să ştim apriori dacă operanzii vor fi interpretaţi drept cu semn sau fără semn. De exemplu, fie doi operanzi A şi B reprezentaţi fiecare pe câte un octet:

`A = 9Ch = 10011100b (= 156 în i.f.s şi -100 în i.c.s)`

`B = 4Ah = 01001010b (= 74 atât în i.f.s cât şi în i.c.s)`

Microprocesorul realizează adunarea `C = A + B` şi obţine

`C = E6h = 11100110b (= 230 în i.f.s şi -26 în i.c.s)`

- Se observă deci că simpla adunare a configuraţiilor de biţi (fără a ne fixa neapărat asupra unei interpretări anume la momentul efectuării adunării) asigură corectitudinea rezultatului obţinut, atât în interpretarea cu semn cât şi în cea fără semn.

Instrucțiuni

Impactul reprezentării little-endian asupra accesării datelor

- Dacă programatorul utilizează datele consistent cu dimensiunea de reprezentare stabilită la definire *(ex: accesarea octeţilor drept octeţi şi nu drept secvenţe de octeţi interpretate ca şi cuvinte sau dublucuvinte, accesarea de cuvinte ca şi cuvinte şi nu ca perechi de octeţi, accesarea de dublucuvinte ca şi dublucuvinte şi nu ca secvenţe de octeţi sau de cuvinte)* atunci instrucţiunile limbajului de asamblare vor ţine cont în mod AUTOMAT de modalitatea de reprezentare little-endian.
- Ca urmare, dacă se respectă această condiţie programatorul nu trebuie să intervină suplimentar în nici un fel pentru a asigura corectitudinea accesării şi manipulării datelor utilizate.
- Exemplu:

```
a db 'd', -25, 120
b dw -15642, 2ba5h
c dd 12345678h
…
mov al, [a]      ; se încarcă în AL codul ASCII al caracterului 'd'
mov bx, [b]      ; se încarcă în BX valoarea -15642
                 ; ordinea octeţilor în BX va fi însă inversată faţă de
                   reprezentarea în memorie, deoarece numai
                   reprezentarea în memorie foloseşte reprezentarea
                   little-endian! În regiştri datele sunt memorate
                   conform reprezentării structurale normale,
                   echivalente unei reprezentări big endian.
mov edx, [c]     ;se încarcă în EDX valoarea dublucuvânt 12345678h
```

Instrucțiuni

Impactul reprezentării little-endian asupra accesării datelor

- Dacă însă se doreşte accesarea sau interpretarea datelor sub o formă diferită faţă de modalitatea de definire atunci trebuie utilizate **conversii explicite de tip**.
- În momentul utilizării conversiilor explicite de tip programatorul trebuie să îşi asume însă întreaga responsabilitate a interpretării şi accesării corecte a datelor.
- În astfel de situaţii programatorul este obligat să conştientizeze particularităţile de reprezentare **little-endian** *(ordinea de plasare a octeţilor în memorie)* şi să utilizeze modalităţi de accesare a datelor în conformitate cu aceasta

Instrucțiuni

Impactul reprezentării little-endian asupra accesării datelor

```
segment data
      a dw 1234h      ;datorită reprezentării little-endian, în memorie octeţii sunt plasaţi astfel:
      b dd 11223344h  ;          34h 12h  44h 33h 22h 11h
                      ; adresa a     a+1 b    b+1 b+2 b+3
      c db -1
segment code
      mov al, byte [a+1]  ;accesarea lui a drept octet, efectuarea calculului de adresă a+1,
                           selectarea octetului de la adresa a+1 (octetul de valoare 12h) şi
                           transferul său în registrul AL.
      mov dx, word [b+2]  ;dx:=1122h
      mov dx, word [a+4]  ;dx:=1122h deoarece b+2 = a+4 , în sensul că aceste expresii de
                           tip pointer desemnează aceeaşi adresă şi anume adresa oct. 22h.
      mov dx, [a+4]       ;această instrucţiune este echivalentă cu cea de mai sus, nefiind
                           realmente necesară
                          ;utilizarea operatorului de conversie WORD
      mov bx, [b]         ;bx:=3344h
      mov bx, [a+2]       ;bx:=3344h, deoarece ca adrese b = a+2.
      mov ecx, dword [a]  ;ecx:=33441234h, deoarece dublucuvântul ce începe la adresa a
                           este format din octeţii 34h 12h 44h 33h care (datorită reprezentării
                           little-endian) înseamnă de fapt ;dublucuvântul 33441234h.
      mov ebx, [b]        ; ebx := 11223344h
      mov ax, word [a+1]  ; ax := 4412h
      mov eax, word [a+1]; eroare de sintaxa. Dacă era dword atunci eax := 22334412h
      mov dx, [c-2]       ; DX := 1122h deoarece c-2 = b+2 = a+4
      mov bh, [b]         ; bh := 44h
      mov ch, [b-1]       ; ch := 12h
      mov cx, [b+3]       ; CX := 0FF11h
```

Instrucțiuni

Constante de tip string. Reprezentare in memorie si utilizare in cadrul unor instructiuni de transfer.

- 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:

`a6 dd '123', '345','abcd'`	; se vor defini 3 dublucuvinte continutul lor fiind | 31h 32h 33h 00h | 33h 34h 35h 00h | 61h 62h 63h 64h |
`a6 dd '1234'`	; | 31h 32h 33h 34h |
`a6 dd '12345' , ‘abc’`	; | 31h 32h 33h 34h | 35h 00h 00h 00h | 61h 62h 63h 00h |
`a7 dw '23','45'`	; | 32h 33h | 34h 35h | - 2 cuvinte = 1 doubleword
`a7 dw '2345'`	; | 32h 33h | 34h 35h| - 2 cuvinte
`a7 dw '23456'`	; | 32h 33h | 34h 35h | 36h 00h | - 3 cuvinte
`a8 dw '1', '2', '3'`	; | 31h 00h | 32h 00h | 33h 00h | - 3 cuvinte
`a9 dw '123'`	; | 31h 32h | 33h 00h | - 2 cuvinte

Instrucțiuni

Constante de tip string. Reprezentare in memorie si utilizare in cadrul unor instructiuni de transfer.

- Urmatoarele definitii produc aceeasi configuratie de memorie

`dd 'ninechars'`	; constanta string doubleword
`dd 'nine','char','s'`	; 3 dublucuvinte
`db 'ninechars',0,0,0`	; "umplere" zona prin secventa de octeti

- Definitia din documentatia oficiala spune:
  A character constant with more than one byte will be arranged with little-endian order in mind: if you code `mov eax,'abcd' (EAX = 0x64636261)` *then the constant generated is not* `0x61626364`*, but* `0x64636261`*, so that if you were then to store the value into memory, it would read* `abcd` *rather than* `dcba`*. This is also the sense of character constants understood by the Pentium's CPUID instruction.*
- Esenta acestei definitii este ca VALOAREA asociata unei constante de tip string '`abcd`' este de fapt '`dcba`' (adica acesta este modul de STOCARE al acestei constante în TABELA DE CONSTANTE)

Instructiune	OllyDbg	Memorie
`Mov dword [a], '2345'`	`mov dword ptr DS:[401000],35343332`	| 32h 33h 34h 35h |

Instrucțiuni

Constante de tip string. Reprezentare in memorie si utilizare in cadrul unor instructiuni de transfer.

- Dar daca folosim a data definition like `a7 dd '2345'` the corresponding memory layout will be NO little-endian representation, but `|32h 33h 34h 35h|`
- Astfel, comparativ si in rezumat (constante tip string vs. constante numerice) :

`a7 dw '2345'`	`; | 32h 33h | 34h 35h|`
`a8 dd 12345678h`	`; | 78h 56h 34h 12h |`
`mov eax, '2345'`	`; EAX = '5432' = 35h 34h 33h 32h`
`mov ebx, [a7]`	`; EBX = '5432' = 35h 34h 33h 32h`
`mov ecx, 12345678h`	`; ECX = 12345678h (sizeof = 4 bytes)`
`mov dword [var1], '12345678'`	`; var1 = '1234' (31 32 33 34)` `; (sizeof '12345678' = 8 bytes)`
`mov edx, [a8]`	`EDX = 12345678h`

Instrucțiuni

Constante de tip string. Reprezentare in memorie si utilizare in cadrul unor instructiuni de transfer.

- In cazul in care se foloseste DB ca directiva de definire a datelor e normal ca ordinea octetilor data in specificarea constantei sa se regaseasca si in memorie in mod similar, deci acest caz nu comporta analiza si discutii suplimentare.

`a66 TIMES 4 db '13'`	; | 31h 33h 31h 33h 31h 33h 31h 33h | echiv cu …db '1','3'
`a67 TIMES 4 dw '13'`	; | 31h 33h 31h 33h 31h 33h 31h 33h | - cele doua moduri de definire diferite produc acelasi rezultat !!!
`a68 TIMES 4 dw '1','3'`	; | 31h 00h | 33h 00h | 31h 00h | 33h 00h | 31h 00h | 33h 00h | 31h 00 h | 33h 00h |
`a69 TIMES 4 dd '13'`	; | 31h 33h 00h 00h | 31h 33h 00h 00h | 31h 33h 00h 00h | 31h 33h 00h 00h |

- Deci, un șir constant în NASM se comportă ca si cum ar exista o „zonă de memorie” alocată anterior acestor constante (ea exista !! si se numeste TABELA DE CONSTANTE!!), unde acestea sunt stocate folosind reprezentarea little-endian !!
- Dpdv al REPREZENTARII valoarea asociată unei constant de tip string este INVERSA SA !!!! (vezi şi “definiţia oficială” de mai sus!)
- Practic, dacă iniţializăm o zonă de memorie cu o constantă de tip string (fie prin directive de definire a datelor, fie prin `mov [zona_de_memorie], constanta_string`) ordinea în care caracterele vor fi depuse în memorie este ordinea în care ele apar în scrierea pe hârtie a constantei de tip string !!!
- Dacă iniţializăm conţinutul unui registru cu o constantă de tip string, caracterele se vor depune în ordine inversă apariţiei lor în scrierea pe hârtie a constantei de tip string !

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSITATEA BABEȘ-BOLYAI

Str. Mihail Kogălniceanu nr. 1
Cluj-Napoca, Cluj, România

www.cs.ubbcluj.ro