Laborator 3 - Suport teoretic

Instrucțiuni de conversie

Conversie cu semn

ADC

Sintaxă:

```
adc <regd>,<regs>; <regd> ← <regd> + <regs> + CF
adc <reg>,<mem>; <reg> ← <reg> + <mem> + CF
adc <mem>,<reg>; <mem> ← <mem> + <reg> + CF
adc <reg>,<con>; <reg> ← <reg> + <con> + CF
adc <mem>,<con>; <mem> ← <mem> + <con> + CF
```

Semantică:

- Cei doi operanzi ai adunării trebuie să aibă același tip (ambii octeți, ambii cuvinte, ambii dublucuvinte);
- În timp ce ambii operanzi pot fi regiștri, cel mult un operand poate fi o locație de memorie.

Exemplu:

```
adc EDX,EBX; EDX ← EDX + EBX + CF
adc AX,[var]; AX ← AX + [var] + CF
adc [var],AX; [var] ← [var] + AX + CF
adc EAX,123456h; EAX ← EAX + 123456h + CF
adc BYTE [var],10; BYTE [var] ← BYTE [var] + 10 + CF
```

SBB

Sintaxă:

```
sbb <regd>,<regs>; <regd> ← <regd> - <regs> - CF
sbb <reg>,<mem>; <reg> ← <reg> - <mem> - CF
sbb <mem>,<reg>; <mem> ← <mem> - <reg> - CF
sbb <reg>,<con>; <reg> ← <reg> - <con> - CF
sbb <mem>,<con>; <mem> ← <mem> - <con> - CF
```

Semantică:

- Cei doi operanzi ai scăderii trebuie să aibă același tip (ambii octeți, ambii cuvinte, ambii dublucuvinte);
- În timp ce ambii operanzi pot fi regiștri, cel mult un operand poate fi o locație de memorie.

Exemplu:

```
sbb EDX,EBX; EDX ← EDX - EBX - CF
sbb AX,[var]; AX ← AX - [var] - CF
sbb [var],AX; [var] ← [var] - AX - CF
sbb EAX,123456h; EAX ← EAX - 123456h - CF
sbb byte [var],10; BYTE [var] ← BYTE [var] - 10 - CF
```

IMUL

Sintaxă:

```
imul <op8>; AX ← AL * <op8>
imul <op16>; DX:AX ← AX * <op16>
imul <op32>; EDX:EAX ← EAX * <op32>
```

Semantică:

- Rezultatul operației de înmulțire se păstrează pe o lungime dublă față de lungimea operanzilor;
- Instrucțiunea IMUL efectuează operația de înmulțire pentru întregi cu semn;
- Se impune ca primul operand și rezultatul să se păstreze în regiștri;
- Operandul explicit poate fi un registru sau o variabilă, dar nu poate fi o valoare imediată (constantă);
- La instrucțiunea IMUL cu doi operanzi, rezultatul înmulțirii celor doi operanzi este păstrat în primul operand;
- La instrucțiunea IMUL cu trei operanzi, rezultatul înmulțirii dintre al doilea operand și valoarea imediată este păstrat în primul operand.

Exemplu:

```
imul DH; AX ← AL * DH
imul mem8; AX ← AL * mem8
imul DX; DX:AX ← AX * DX
imul EBX; EDX:EAX ← EAX * EBX
```

IDIV

Sintaxă:

```
idiv <reg8>; AL ← AX / <reg8>, AH ← AX % <reg8>
idiv <reg16>; AX ← DX:AX / <reg16>, DX ← DX:AX % <reg16>
idiv <reg32>; EAX ← EDX:EAX / <reg32>, EDX ← EDX:EAX % <reg32>
idiv <mem8>; AL ← EAX / <mem8>, AH ← AX % <mem8>
idiv <mem16>; AX ← DX:AX / <mem16>, DX ← DX:AX % <mem16>
idiv <mem32>; EAX ← EDX:EAX / <mem32>, EDX ← EDX:EAX % <mem32>
```

Semantică:

- Instrucțiunea IDIV efectuează operația de împărțire pentru întregi cu semn;

- Se impune ca primul operand și rezultatul să se păstreze în regiștri;
- Primul operand nu se specifică și are o lungime dublă față de al doilea operand;
- Operandul explicit poate fi un registru sau o variabilă, dar nu poate fi o valoare imediată (constantă);
- Prin împărțirea unui număr mare la un număr mic, există posibilitatea ca rezultatul să depășească capacitatea de reprezentare. În acest caz, se va declanșa aceeași eroare ca și la împărțirea cu 0.

Exemplu:

```
idiv CL; AL ← AX / CL, AH ← AX % CL
idiv SI; AX ← DX:AX / SI, DX ← DX:AX % SI
idiv EBX; EAX ← EDX:EAX / EBX, EDX ← EDX:EAX % EBX
idiv DWORD [var]; EAX ← EDX:EAX / DWORD [var], EDX ← EDX:EAX % DWORD [var]
```

CBW

Sintaxă:

```
cbw
```

Semantică:

- convertește cu semn BYTE-ul din AL la WORD-ul AX;
- conversia se referă la extinderea reprezentării de pe 8 biți pe 16 biți, prin completarea cu bitul de semn în fața octetului inițial;
- instrucțiunea nu are operanzi specificați explicit deoarece este întotdeauna vorba despre conversia AL → AX.

Exemplu:

```
cbw ; dacă AL=01110111b atunci AX ← 00000000 01110111b
    ; dacă AL=11110111b atunci AX ← 11111111 11110111b
```

CWD

Sintaxă:

```
cwd
```

Semantică:

- convertește cu semn WORD-ul din AX la DWORD-ul DX:AX;
- conversia se referă la extinderea reprezentării de pe 16 biți pe 32 biți, prin completarea cu bitul de semn în fața cuvântului inițial;
- instrucțiunea nu are operanzi specificați explicit deoarece este întotdeauna vorba despre conversia AX → DX:AX.

Exemplu:

```
cwd ; dacă AX=00110011 11001100b atunci DX:AX ← 00000000 00000000 00110011
11001100b
    ; dacă AX=10110011 11001100b atunci DX:AX ← 11111111 11111111 10110011
11001100b
```

CWDE

Sintaxă:

```
cwde
```

Semantică:

- convertește cu semn WORD-ul din AX la DWORD-ul EAX;
- conversia se referă la extinderea reprezentării de pe 16 biți pe 32 biți, prin completarea cu bitul de semn în fața cuvântului inițial;
- instrucțiunea nu are operanzi specificați explicit deoarece este întotdeauna vorba despre conversia AX → EAX.

Exemplu:

```
cwde ; dacă AX=00110011 11001100b atunci EAX ← 00000000 00000000 00110011
11001100b
     ; dacă AX=10110011 11001100b atunci EAX ← 11111111 11111111 10110011
11001100b
```

CDQ

Sintaxă:

```
cdq
```

Semantică:

- convertește cu semn DWORD-ul din EAX la QWORD-ul EDX:EAX;
- conversia se referă la extinderea reprezentării de pe 32 biți pe 64 biți, prin completarea cu bitul de semn în fața cuvântului inițial;
- instrucțiunea nu are operanzi specificați explicit deoarece este întotdeauna vorba despre conversia EAX → EDX:EAX.

Exemplu:

```
cdq ; dacă EAX=00110011 11001100 00110011 11001100b atunci EDX:EAX ← 00000000
00000000 00000000 00000000 00110011 11001100 00110011 11001100b
     ; dacă EAX=10110011 11001100 10110011 11001100b atunci EDX:EAX ← 11111111
11111111 11111111 11111111 10110011 11001100 10110011 11001100b
```

Conversie fără semn

Conversie fără semn

- Nu există instrucţiuni de conversie fără semn;
- Conversiile fără semn se realizează în limbajul de asamblare prin „zerorizarea” octetului, cuvântului sau dublucuvântului superior.

Exemple

```
mov AH,0 ; pentru conversia AL → AX
mov DX,0 ; pentru conversia AX → DX:AX
mov EDX,0 ; pentru conversia EAX → EDX:EAX
```

Declararea variabilelor / constantelor

Declararea variabilelor cu valoare initiala

```
a DB 0A2h ;se declara variabila a de tip BYTE si se initializeaza cu valoarea
0A2h
b DW 'ab' ;se declara variabila a de tip WORD si se initializeaza cu valoarea
'ab'
c DD 12345678h ;se declara variabila a de tip DOUBLE WORD si se initializeaza
cu valoarea 12345678h
d DQ 1122334455667788h ;se declara variabila a de tip QUAD WORD si se
initializeaza cu valoarea 1122334455667788h
```

Declararea variabilelor fara valoare initiala

```
a RESB 1 ;se rezerva 1 octet
b RESB 64 ;se rezerva 64 octeti
c RESW 1 ;se rezerva 1 word
```

Definirea constantelor

```
zece EQU 10 ;se defineste constanta zece care are valoarea 10
```

Legendă

```
<op8>    - operand pe 8 biți
<op16>   - operand pe 16 biți
<op32>   - operand pe 32 biți

<reg8>   - registru pe 8 biți
<reg16>  - registru pe 16 biți
<reg32>  - registru pe 32 biți
<reg>    - registru
<regd>   - registru destinație
<regs>   - registru sursă

<mem8>   - variabilă de memorie pe 8 biți
<mem16>  - variabilă de memorie pe 16 biți
<mem32>  - variabilă de memorie pe 32 biți
<mem>    - variabilă de memorie

<con8>   - constantă (valoare imediată) pe 8 biți
<con16>  - constantă (valoare imediată) pe 16 biți
<con32>  - constantă (valoare imediată) pe 32 biți
<con>    - constantă (valoare imediată)
```

Laborator 3 – Exemple

Interpretare fără semn

```
; Scrieți un program în limbaj de asamblare care să rezolve expresia
aritmetică, considerând domeniile de definiție ale variabilelor
; a - doubleword; b, d - byte; c - word; e - qword
; a + b / c - d * 2 - e
bits 32 ;asamblare si compilare pentru arhitectura de 32 biti
; definim punctul de intrare in programul principal
global  start

extern  exit ; indicam asamblorului ca exit exista, chiar daca noi nu o vom
defini
import  exit msvcrt.dll; exit este o functie care incheie procesul, este
definita in msvcrt.dll
        ; msvcrt.dll contine exit, printf si toate celelalte functii C-runtime
importante
segment  data use32 class=data ; segmentul de date in care se vor defini
variabilele
        a dd 125
        b db 2
        c dw 15
        d db 200
        e dq 80
segment  code use32 class=code ; segmentul de cod
start:
        ;pentru a calcula b/c, convertim b de la byte la doubleword pentru a-l
putea imparti la word-ul c
        mov al, [b]
        mov ah, 0 ;conversie fara semn de la al la ax
        mov dx, 0 ;conversie fara semn de la ax la dx:ax
        ;dx:ax = b
        div word [c] ;impartire fara semn dx:ax la c
        ;ax=b/c
        ;catul impartirii este in ax (restul este in dx, dar mergem mai
departe doar cu catul)

        mov bx, ax ;salvam b/c in bx pentru a putea folosi ax la inmultirea
d*2
        mov al, 2
        mul byte [d] ;ax=d*2

        sub bx, ax ;bx = b / c - d * 2
        ; convertim la doubleword word-ul bx pentru a-l putea aduna cu
doubleword-ul a
        mov cx, 0 ; conversie fara semn de la bx la cx:bx
        ;cx:bx=b/c-d*2

        mov ax, word [a]
        mov dx, word [a+2] ;dx:ax=a

        add ax, bx
        adc dx, cx ;dx:ax = a + b / c - d * 2
```

```
        push dx
        push ax
        pop eax ;eax = a + b / c - d * 2

        mov edx, 0 ;edx:eax = a + b / c - d * 2
        sub eax, dword [e]
        sbb edx, dword [e+4] ;edx:eax = a + b / c - d * 2 - e

        push dword 0 ;se pune pe stiva codul de retur al functiei exit
        call [exit] ;apelul functiei sistem exit pentru terminarea executiei
programului
```

Interpretare cu semn

```
; Scrieți un program în limbaj de asamblare care să rezolve expresia
aritmetică, considerând domeniile de definiție ale variabilelor
; a - doubleword; b, d - byte; c - word; e - qword
; a + b / c - d * 2 - e
bits 32 ;asamblare si compilare pentru arhitectura de 32 biti
; definim punctul de intrare in programul principal
global  start

extern  exit ; indicam asamblorului ca exit exista, chiar daca noi nu o vom
defini
import  exit msvcrt.dll; exit este o functie care incheie procesul, este
definita in msvcrt.dll
        ; msvcrt.dll contine exit, printf si toate celelalte functii C-runtime
importante
segment  data use32 class=data ; segmentul de date in care se vor defini
variabilele
        a dd 125
        b db 2
        c dw 15
        d db 200
        e dq 80
segment  code use32 class=code ; segmentul de cod
start:
        ;pentru a calcula b/c, convertim b de la byte la doubleword pentru a-l
putea imparti la word-ul c
        mov al, [b]
        cbw ;conversie cu semn de la al la ax
        cwd ;conversie cu semn de la ax la dx:ax
        ;dx:ax = b
        idiv word [c] ;impartire cu semn dx:ax la c
        ;ax=b/c
        ;catul impartirii este in ax (restul este in dx, dar mergem mai
departe doar cu catul)

        mov bx, ax ;salvam b/c in bx pentru a putea folosi ax la inmultirea
d*2
        mov al, 2
        imul byte [d] ;ax=d*2

        sub bx, ax ;bx=b/c-d*2
```

```
        ; convertim la doubleword word-ul bx pentru a-l putea aduna cu
doubleword-ul a
        mov ax, bx
        cwd ; conversie cu semn de la ax la dx:ax
        ;dx:ax=b/c-d*2

        mov bx, word [a]
        mov cx, word [a+2] ;cx:bx=a

        add ax, bx
        adc dx, cx ;rezultatul a + b / c - d * 2 este in dx:ax

        push dx
        push ax
        pop eax ;eax = a + b / c - d * 2
        cdq ;edx:eax = a + b / c - d * 2

        mov edx, 0 ;edx:eax = a + b / c - d * 2
        sub eax, dword [e]
        sbb edx, dword [e+4] ;edx:eax = a + b / c - d * 2 - e

        push dword 0 ;se pune pe stiva codul de retur al functiei exit
        call [exit] ;apelul functiei sistem exit pentru terminarea executiei
programului
```

Laborator 3 - Probleme propuse

Scrieți un program în limbaj de asamblare care să rezolve expresia aritmetică, considerând domeniile de definiție ale variabilelor (in interpretarea cu semn si in interpretarea fara semn).

Adunări, scăderi

a - byte, b - word, c - double word, d - qword

1. c-(a+d)+(b+d)
2. (b+b)+(c-a)+d
3. (c+d)-(a+d)+b
4. (a-b)+(c-b-d)+d
5. (c-a-d)+(c-b)-a
6. (a+b)-(a+d)+(c-a)
7. c-(d+d+d)+(a-b)
8. (a+b-d)+(a-b-d)
9. (d+d-b)+(c-a)+d
10. (a+d+d)-c+(b+b)

---

a - byte, b - word, c - double word, d - qword

1. (c+b+a)-(d+d)
2. (c+b)-a-(d+d)
3. (b+b+d)-(c+a)
4. (b+b)-c-(a+d)
5. (c+b+b)-(c+a+d)
6. c-(d+a)+(b+c)
7. (c+c+c)-b+(d-a)
8. (b+c+d)-(a+a)
9. a-d+b+b+c
10. b+c+d+a-(d+c)

Înmulțiri, împărțiri

1. c+(a*a-b+7)/(2+a)
   a-byte; b-doubleword; c-qword
2. 2/(a+b*c-9)+e-d
   a,b,c-byte; d-doubleword; e-qword
3. (8-a*b*100+c)/d+x
   a,b,d-byte; c-doubleword; x-qword
4. (a*2+b/2+e)/(c-d)+x/a
   a-word; b,c,d-byte; e-doubleword; x-qword

---

5. (a+b/c-1)/(b+2)-x
   a-doubleword; b-byte; c-word; x-qword
6. x+a/b+c*d-b/c+e
   a,b,d-byte; c-word; e-doubleword; x-qword
7. (a-2)/(b+c)+a*c+e-x
   a,b-byte; c-word; e-doubleword; x-qword
8. 1/a+200*b-c/(d+1)+x/a-e
   a,b-word; c,d-byte; e-doubleword; x-qword
9. (a-b+c*128)/(a+b)+e-x
   a,b-byte; c-word; e-doubleword; x-qword
10. d-(7-a*b+c)/a-6+x/2
    a,c-byte; b-word; d-doubleword; x-qword
11. (a+b)/(2-b*b+b/c)-x
    a-doubleword; b,c-byte; x-qword
12. (a*b+2)/(a+7-c)+d+x
    a,c-byte; b-word; d-doubleword; x-qword
13. x-(a+b+c*d)/(9-a)
    a,c,d-byte; b-doubleword; x-qword
14. x+(2-a*b)/(a*3)-a+c
    a-byte; b-word; c-doubleword; x-qword
15. x-(a*b*25+c*3)/(a+b)+e
    a,b,c-byte; e-doubleword
16. x/2+100*(a+b)-3/(c+d)+e*e
    a,c-word; b,d-byte; e-doubleword; x-qword
17. x-(a*a+b)/(a+c/a)
    a,c-byte; b-doubleword; x-qword
18. (a+b*c+2/c)/(2+a)+e+x
    a,b-byte; c-word; e-doubleword; x-qword
19. (a+a+b*c*100+x)/(a+10)+e*a
    a,b,c-byte; e-doubleword; x-qword
20. x-b+8+(2*a-b)/(b*b)+e
    a-word; b-byte; e-doubleword; x-qword
21. (a*a/b+b*b)/(2+b)+e-x
    a-byte; b-word; e-doubleword; x-qword
22. a/2+b*b-a*b*c+e+x
    a,b,c-byte; e-doubleword; x-qword
23. (a*b-2*c*d)/(c-e)+x/a
    a,b,c,d-byte; e-word; x-qword
24. a-(7+x)/(b*b-c/d+2)
    a-doubleword; b,c,d-byte; x-qword
25. (a*a+b+x)/(b+b)+c*c
    a-word; b-byte; c-doubleword; x-qword
26. (a*a+b/c-1)/(b+c)+d-x
    a-word; b-byte; c-word; d-doubleword; x-qword
27. (100+a+b*c)/(a-100)+e+x/a
    a,b-byte; c-word; e-doubleword; x-qword
28. x-(a*100+b)/(b+c-1)
    a-word; b-byte; c-word; x-qword
29. (a+b)/(c-2)-d+2-x
    a,b,c-byte; d-doubleword; x-qword
30. a*b-(100-c)/(b*b)+e+x
    a-word; b,c-byte; e-doubleword; x-qword