Laborator 2 - Suport teoretic

Instrucțiuni aritmetice

ADD

Sintaxă:

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

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:

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

SUB

Sintaxă:

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

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:

```
sub EDX,EBX; EDX ← EDX - EBX
sub AX,[var]; AX ← AX - [var]
sub [var],AX; [var] ← [var] - AX
```

```
sub EAX,123456h; EAX ← EAX - 123456h
sub byte [var],10; BYTE [var] ← BYTE [var] - 10
```

MUL

Sintaxă:

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

Semantică:

- Rezultatul operației de înmulțire se păstrează pe o lungime dublă față de lungimea operanzilor;
- Instrucțiunea MUL efectuează operația de înmulțire pentru întregi fără semn;
- Se impune ca primul operand și rezultatul să se păstreze în regiștri;
- Deși operația este binară, se specifică un singur operand deoarece celălalt este întotdeauna fixat, la fel ca și locația rezultatului.
- Operandul explicit poate fi un registru sau o variabilă, dar nu poate fi o valoare imediată (constantă).

Exemplu:

```
mul DH; AX ← AL * DH
mul DX; DX:AX ← AX * DX
mul EBX; EDX:EAX ← EAX * EBX
mul BYTE [mem8]; AX ← AL * BYTE [mem8]
mul WORD [mem16]; DX:AX ← AX * WORD [mem8]
```

DIV

Sintaxă:

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

Semantică:

- Instrucțiunea DIV efectuează operația de împărțire pentru întregi fără 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:

```
div CL; AL ← AX / CL, AH ← AX % CL
```

```
div SI; AX ← DX:AX / SI, DX ← DX:AX % SI
div EBX; EAX ← EDX:EAX / EBX, EDX ← EDX:EAX % EBX
```

INC

Sintaxă:

```
inc <reg>; <reg> ← <reg> + 1
inc <mem>; <mem> ← <reg> + 1
```

Semantică:

- Incrementează conținutul operandului cu 1.

Exemplu:

```
inc DWORD [var]; DWORD [var] ← DWORD [var] + 1
inc EBX; EBX ← EBX + 1
inc DL; DL ← DL + 1
```

DEC

Sintaxă:

```
dec <reg>; <reg> ← <reg> - 1
dec <mem>; <mem> ← <reg> - 1
```

Semantică:

- Decrementează conținutul operandului cu 1.

Exemplu:

```
dec EAX; EAX ← EAX - 1
dec BYTE [mem]; [value] ← [value] - 1
```

NEG

Sintaxă:

```
neg <reg>; <reg> ← 0 - <reg>
neg <mem>; <mem> ← 0 - <mem>
```

Semantică:

- Are loc negarea aritmetică a operandului.

Exemplu:

```
neg EAX; EAX ← 0 - EAX
```

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 2 – Exemple

Exemple simple

1. Puneți o valoare în registrul AL. Analizați acea valoare făcând debug.

```
mov AL, 7
```

```
mov AL, 12
```

3. Puneți o valoare în registrul AX. Analizați acea valoare făcând debug.

```
mov AX, 256
```

```
mov AX, -1
```

5. Puneți o valoare în registrul EAX. Analizați acea valoare făcând debug.

```
mov EAX, 40000
```

```
mov EAX, -2
```

7. Puneți două valori în doi regiștri. Adunați cele două valori. Scădeți cele două valori. Analizați rezultatele obținute făcând debug.

```
mov AX, 300
mov BX, 256
 add AX, BX
```

```
sub AX, BX
```

11. Calculați 5 - 6. Analizați rezultatul obținut făcând debug.
12. Calculați AL * BL. Analizați rezultatul obținut făcând debug.
13. Calculați AX / BL. Analizați rezultatul obținut făcând debug.

Exemple complexe

```
; Scrieți un program în limbaj de asamblare care să rezolve expresia
aritmetică, considerând domeniile de definiție ale variabilelor
; a - byte, b - word
; (b / a * 2 + 10) * b - b * 15;
; ex. 1: a = 10; b = 40; Rezultat: (40 / 10 * 2 + 10) * 40 - 40 * 15 = 18 * 40
- 1600 = 720 - 600 = 120
; ex. 2: a = 100; b = 50; Rezultat: (50 / 100 * 2 + 10) * 50 - 50 * 15 = 12 *
50 - 750 = 600 - 750 = - 150
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  db 10
        b  dw 40
segment  code use32 class=code ; segmentul de cod
start:
        mov  AX, [b] ;AX = b
        div  BYTE [a] ;AL = AX / a = b / a și AH = AX % a = b % a

        mov  AH, 2 ;AH = 2
        mul  AH ;AX = AL * AH = b / a * 2

        add  AX, 10 ;AX = AX + b = b / a * 2 + 10

        mul  word [b] ;DX:AX = AX * b = (b / a * 2 + 10) * b

        push  DX ;se pune pe stiva partea high din double word-ul DX:AX
        push  AX ;se pune pe stiva partea low din double word-ul DX:AX
        pop  EBX ;EBX = DX:AX = (b / a * 2 + 10) * b

        mov  AX, [b] ;AX = b
        mov  DX, 15 ;DX = 15
        mul  DX ;DX:AX = b * 15

        push  DX ;se pune pe stiva partea high din double word-ul DX:AX
        push  AX ;se pune pe stiva partea low din double word-ul DX:AX
        pop  EAX ;EAX = DX:AX = b * 15

        sub  EBX, EAX ;EBX = EBX - EAX = (b / a * 2 + 10) * b - b * 15

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

Laborator 2 - Probleme propuse

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

Exerciții simple

Efectuați calculele și analizați rezultatele

1. 1+9
2. 1+15
3. 128+128
4. 5-6
5. 10/(-4)
6. 256*1
7. 256/1
8. 128+128
9. (-3)*4

Adunări, scăderi

a,b,c,d - byte

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,b,c,d - word

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

a,b,c - byte, d - word

1. ((a+b-c)*2 + d-5)*d
2. d*(d+2*a)/(b*c)
3. [-1+d-2*(b+1)]/a
4. –a*a + 2*(b-1) – d
5. [d-2*(a-b)+b*c]/2
6. [2*(a+b)-5*c]*(d-3)
7. [100*(d+3)-10]/d
8. (100*a+d+5-75*b)/(c-5)
9. 3*[20*(b-a+2)-10*c]+2*(d-3)
10. 3*[20*(b-a+2)-10*c]/5

---

a,b,c,d-byte, e,f,g,h-word

1. (a+b)-(c+d)
2. e-a*a
3. (a*b)/c
4. (a-c)*3+b*b
5. a-(b+c)+34
6. (e+f)*g
7. 2*(a+b)-e
8. ((a-b)*4)/c
9. (a+(b-c))*3
10. a*d+b*c
