**Laborator 3 - Suport teoretic**

**Instrucțiuni aritmetice**

**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 și restricții:***

* 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;
* La suma dintre cei doi operanzi se mai adună şi valoarea bitului de transport (Carry Flag).

***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 și restricții:***

* 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;
* Din diferenţa dintre cei doi operanzi se mai scade şi valoarea bitului de transport (Carry Flag).

***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 și restricții:***

* 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);

***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 și restricții:***

* 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]

**Instrucțiuni de conversie cu semn**

**CBW**

***Sintaxă:***

cbw

***Semantică și restricții:***

* converteşte cu semn BYTE-ul din AL la WORD-ul AX;
* conversia se referă la extinderea reprezentării de pe 8 biţi pe 16 biţi, prin completarea cu bitul de semn în faţa octetului iniţial;
* instrucţiunea nu are operanzi specificaţi 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 și restricții:***

* converteşte cu semn WORD-ul din AX la DWORD-ul DX:AX;
* conversia se referă la extinderea reprezentării de pe 16 biţi pe 32 biţi, prin completarea cu bitul de semn în faţa cuvântului iniţial;
* instrucţiunea nu are operanzi specificaţi 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 și restricții:***

* converteşte cu semn WORD-ul din AX la DWORD-ul EAX;
* conversia se referă la extinderea reprezentării de pe 16 biţi pe 32 biţi, prin completarea cu bitul de semn în faţa cuvântului iniţial;
* instrucţiunea nu are operanzi specificaţi 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 și restricții:***

* converteşte cu semn DWORD-ul din EAX la QWORD-ul EDX:EAX;
* conversia se referă la extinderea reprezentării de pe 32 biţi pe 64 biţi, prin completarea cu bitul de semn în faţa dublucuvântului iniţial;
* instrucţiunea nu are operanzi specificaţi 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 inițială

a DB 0A2h ;se declara variabila a de tip BYTE si se iniţializează cu valoarea 0A2h

b DW 'ab' ;se declara variabila a de tip WORD si se iniţializează cu valoarea 'ab'

c DD 12345678h ;se declara variabila a de tip DOUBLE WORD si se iniţializează cu valoarea 12345678h

d DQ 1122334455667788h ;se declara variabila a de tip QUAD WORD si se iniţializează cu valoarea 1122334455667788h

## Declararea variabilelor fără valoare inițială

a RESB 1 ;se rezerva 1 octet

b RESB 64 ;se rezerva 64 octeți

c RESW 1 ;se rezerva 1 word

## Definirea constantelor

zece EQU 10 ;se definește 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 biți

; 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 funcție care încheie procesul, este definita in msvcrt.dll

; msvcrt.dll conține exit, printf si toate celelalte funcții 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 împărți la word-ul c

mov al, [b]

mov ah, 0 ;conversie fără semn de la al la ax

mov dx, 0 ;conversie fără semn de la ax la dx:ax

;dx:ax = b

div word [c] ; împărţire fără semn dx:ax la c

;ax=b/c

;catul împărţirii 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 înmulțirea 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 fără 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 funcției exit

call [exit] ;apelul funcției sistem exit pentru terminarea execuției 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 biți

; definim punctul de intrare in programul principal

global start

extern exit ; indicam asamblorului că exit există, chiar daca noi nu o vom defini

import exit msvcrt.dll; exit este o funcție care încheie procesul, este definita in msvcrt.dll

; msvcrt.dll conține exit, printf si toate celelalte funcții 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 împărţi 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] ; împărţire cu semn dx:ax la c

;ax=b/c

;catul împărţirii 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 funcției exit

call [exit] ;apelul funcției sistem exit pentru terminarea execuției programului

**Adunări, scăderi**

**a - byte, b - word, c - double word, d - qword - Interpretare fără semn**

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*)
11. (*d*-*c*)+(*b*-*a*)-(*b*+*b*+*b*)
12. (*a*+*b*+*d*)-(*a*-*c*+*d*)+(*b*-*c*)
13. *d*-*b*+*a*-(*b*+*c*)
14. (*a*+*d*)-(*c*-*b*)+*c*
15. *a*+*b*-*c*+(*d*-*a*)
16. *c*-*a*-(*b*+*a*)+*c*
17. (*c*+*c*-*a*)-(*d*+*d*)-*b*
18. (*d*+*d*)-*a*-*b*-*c*
19. (*d*+*d*)-(*a*+*a*)-(*b*+*b*)-(*c*+*c*)
20. (*a*+*c*)-*b*+*a* + (*d*-*c*)
21. (*c* - *a*) + (*b* - *d*) + *d*
22. (*d* + *c*) - (*c* + *b*) - (*b* + *a*)
23. ((*a* + *a*) + (*b* + *b*) + (*c* + *c*)) - *d*
24. ((*a* + *b*) + (*a* + *c*) + (*b* + *c*)) - *d*
25. (*a* + *b* + *c*) - (*d* + *d*) + (*b* + *c*)
26. (*c* - *b* + *a*) - (*d* + *a*)
27. (*a* + *c*) - (*d* + *b*)
28. *d* - (*a* + *b*) + (*c* + *c*)
29. *d* + *c* - *b* + (*a* - *c*)
30. (*b* + *c* + *a*) - (*d* + *c* + *a*)

**a - byte, b - word, c - double word, d - qword - Interpretare cu semn**

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*)
11. *d* - (*a* + *b* + *c*) - (*a* + *a*)
12. (*a* - *b* - *c*) + (*d* - *b* - *c*) - (*a* - *d*)
13. (*b* - *a* + *c* - *d*) - (*d* + *c* - *a* - *b*)
14. *c* - *b* - (*a* + *a*) - *b*
15. *c* + *a* + *b* + *b* + *a*
16. (*d* - *a*) - (*a* - *c*) - *d*
17. (*c* + *d* - *a*) - (*d* - *c*) - *b*
18. (*d* - *b*) - *a* - (*b* - *c*)
19. (*d* + *a*) - (*c* - *b*) - (*b* - *a*) + (*c* + *d*)
20. *a* - *b* - (*c* - *d*) + *d*
21. *d* - *a* + (*b* + *a* - *c*)
22. *c* + *b* - (*a* - *d* + *b*)
23. *a* + *b* + *c* + *d* - (*a* + *b*)
24. (*a* + *b* + *c*) - *d* + (*b* - *c*)
25. (*a* + *b* - *c*) + (*a* + *b* + *d*) - (*a* + *b*)
26. (*c* - *d* - *a*) + (*b* + *b*) - (*c* + *a*)
27. (*d* + *d* - *c*) - (*c* + *c* - *a*) + (*c* + *a*)
28. *c* + *d* - *a* - *b* + (*c* - *a*)
29. (*a* + *a*) - (*b* + *b*) - (*c* + *d*) + (*d* + *d*)
30. *d* - *a* + *c* + *c* - *b* + *a*

**Înmulțiri, împărțiri - Interpretare fără semn şi interpretare cu semn**

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