LUCRAREA DE LABORATOR 2

NOŢIUNI DE BAZĂ

1. SCOPUL LUCRARII

Se prezinta problemele principale legate de conversii de date, reprezentarea datelor întregi, reprezentarea întregilor in format BCD, reprezentarea caracterelor si a şirurilor de caractere, reprezentarea valorilor reale, elemente de memorie, tipuri de date utilizate si modurile de adresare a operanzilor.

2. NOTIUNI TEORETICE

2.1. Reprezentarea datelor

Programele scrise in limbaje de asamblare, la fel ca si cele scrise in limbaje de nivel superior prelucrează date pentru a obţine rezultate. Pentru a fi prelucrate, datele in final sunt convertite in binar.

Conversiile ce se realizează sunt specifice tipului de date, de exemplu: întregul 15 este convertit in binar altfel decât valoarea reala 15.0 sau de şirul de caractere '15'. Astfel:

- întregul 15, reprezentat in binar, pe un octet se reprezintă astfel: 0000 1111 sau 0fh
- întregul 15, reprezentat in format BCD împachetat este: 0001 0101 sau 15h
- şirurile de caractere '15' se reprezintă, pe octeţi, astfel: 0011 0001 0011 0101 sau 3135h
- valoarea reala 15.0, in simpla precizie-virgula mobila, se reprezintă astfel:
  0100 0001 1110 0000 0000 0000 0000 0000 sau 41e0 0000h

Deşi calculatorul operează asupra datelor reprezentate in binar, pentru depanarea si vizualizarea datelor se prefera reprezentarea lor externa in hexazecimal. Folosirea sistemului hexazecimal de reprezentare a datelor se bazează pe:

- codurile binare sunt relativ lungi si dificil de lucrat cu ele. Codurile hexazecimale, asemănătoare celor zecimale, sunt mai uşor de citit.
- exista metode directe de trecere din binar in hexazecimal si invers.
- informaţia din calculator este organizata sub forma unor multipli de 16 (cuvinte, dublu cuvânt, double word, quad word, ten bytes) sau submultipli de 16 (octet, tetradă).

Conversia dintr-o baza in alta

Pentru a converti un număr din baza p in baza q se poate aplica una din metodele generale de conversie:

-conversia cu calcul in baza iniţială;

- conversia cu calcul in baza finala;

- conversia cu calcul in baza intermediara.

Pentru numere reale conversia din baza 10 într-o alta baza se face separat conversia pentru partea întreagă, respectiv pentru partea fracţionară. Pentru ***conversia părţi întregi*** a unui număr real, care acoperă si numerele întregi, se procedează astfel:

-se împarte numărul, respectiv câturile, la baza si se reţin resturile parţiale pana catul devine zero;

-se i-au resturile obţinute in ordine inversa calculului si se formează numărul in noua baza.

Pentru ***conversia părţii fracţionare*** se înmulţeşte aceasta cu baza si se retine partea întreagă ca fiind cifra a numărului in noua baza iar partea fracţionară se înmulţeşte din nou cu baza s.a.m.d. pana se obţine partea fracţionară zero s-au numărul de cifre a părţii fracţionare in noua baza dorit.

Pentru exemplificare folosind prima metoda se va converti numărul zecimal 348.6785 in baza 16.

Cifrele bazei 16 sunt: 0, 1, 2,3, 4, 5, 6, 7, 8, 9, 0, a, b, c, d,e, f.

Începem cu partea întreagă:

348:16= 21 rest 12, adică „c”

21:16=1 rest 5

1:16=0 rest 1

deci $348_{10}=15C_{16}$.

Convertim partea fracţionară:

0.6785*16=10.856 prima cifra a părţii fracţionare este 10 adică A

0.856*16=13.696 următoarea cifra a părţii fracţionare este 13 adică D

0.696*16=11.136 următoarea cifra a părţii fracţionare este 11 adică B

0.136*16= 2.176 următoarea cifra a părţii fracţionare este 2

.....

deci $0.6785_{10}=0.ADB2_{16}$

Conversia dintr-o alta baza in baza 10 se face cu calcul in baza finala, de exemplu:

$2a75.bdb3_{16}=2*16^3+a*16^2+7*16^1+5*16^0+b*16^{-1}+d*16^{-2}+b*16^{-3}+3*16^{-4}=$

$2*16^3+10*16^2+7*16^1+5*16^0+11*16^{-1}+13*16^{-2}+11*16^{-3}+3*16^{-4}=10869.7414703369141_{10}$

Conversia binar – hexazecimala directa.

Numerele reprezentate in binar se grupează in grupe de câte 4 biţi, de la dreapta la stânga. Fiecare grupa se înlocuieşte cu cifra hexazecimala corespunzătoare. De exemplu:

```
101 1011 0011 1011
---- ----- ------ -----
 5    B     3    B
```

daca e cazul tetrada cea mai din stânga se completează cu zerouri, deci vom avea:

$0101101100111011_2 = 5B3B_{16}$

In capul unui număr binar, care are si parte subunitara, se fac grupe de 4 biţi de la virgula spre dreapta, respectiv spre stânga. Ultimele grupe, daca nu sunt complete, se completează cu zerouri. Ex.:

```
001011010110.11010100
-----  ----- -----   ----  -----
  2     d     6     d     4
```

deci: $1011010110 . 110101_2 = 2d6 . d4_{16}$

Conversia hexazecimal – binar directa

Se realizează invers, fiecare cifra hexazecimala se înlocuieşte cu echivalentul binar format din 4 cifre binare.

Exemplu:

$5B2C_{16}$ va fi in binar:

```
0101 1011 0010 1100
-----  -----  ------  -----
  5     B      2       C
```

Deci: $5B2C_{16} = 0101101100101100_2$

sau:

$2C3 . 64C_{16} = 001011000011 . 011001001100_2$

2.1.1.Reprezentarea numerelor întregi in binar

In cazul numerelor binare cu semn se foloseşte convenţia prin care primul bit este bitul de semn, daca are valoarea 0 numărul este pozitiv, iar daca are valoarea 1 numărul este negativ.

Exista 3 metode de reprezentare a numerelor binare întregi cu semn:

-directa (in reprezentare apare semnul si valoarea absoluta)

-complement fata de unu

-complement fata de doi sau complement adevărat.

Metoda directa

In acest caz numărul cu semn se reprezintă prin semn si valoarea sa absoluta.

De exemplu: întregul 30 se reprezintă pe octet astfel: pe ultimi 7 biţi se reprezintă valoarea absoluta a numărului iar bitul 7 cel mai semnificativ este bitul de semn (1-minus si 0-plus):

00011110

iar - 30 se reprezintă:

10011110

Complement fata de unu

In complement fata de unu, numerele *pozitive* se reprezintă la fel ca in cazul metodei directe, adică după bitul de semn care este 0 urmează valoarea absoluta a numărului.

Daca numărul este *negativ* se reprezintă numărul in binar ca un număr pozitiv după care se face complementul fata de 1, adică se înlocuieşte fiecare bit 0 cu 1, respectiv fiecare bit 1 cu 0.

De exemplu numărul – 30 se reprezintă pe octet astfel:

- se reprezintă pe octet 30 (valoarea absoluta)

00011110_2

- se face complementul fata de 1, adică

11100001

deci $-30_{10} = 1110001_2 = 0e1_{16}$ in complement fata de 1 pe octet.

Daca dorim sa reprezentam –30 pe cuvânt, parcurgem paşii:

-reprezentam valoarea absoluta 30 pe cuvânt:

0000000000011110

-facem complementul fata de unu a întregului cuvânt

1111111111100001

si avem $-30_{10} = 1111\ 1111\ 1110\ 0001_2 = 0ffe1_{16}$ in complement fata de 1 pe cuvânt.

Complement fata de doi

De obicei in calculatoare pentru reprezentarea numerelor cu semn se foloseşte complementul fata de doi. Reprezentarea in complement fata de doi se realizează astfel:

- daca *numărul este pozitiv*, complement fata de doi este identic cu complement fata de unu
- daca *numărul este negativ* la reprezentarea in complement fata de unu se aduna 1

De exemplu -30_{10} reprezentat in complement fata de doi pe octet se face astfel:

-se reprezintă in binar pe octet valoarea absoluta a numărului

00011110

-se calculează complement fata de unu, adică:

11100001

-la valoarea calculata se aduna 1, deoarece numărul este negativ si avem

```
11100001 +
       1
---------------
11100010
```

Deci -30_{10} in complement fata de doi este $11100010_2 = 0e2_{16}$ pe octet.

Acelaşi număr reprezentat in complement fata de doi pe cuvânt va fi:

$30_{10} = 1111\ 1111\ 1110\ 0010_2 = 0ff\ e2_{16}$

2.1.2. Reprezentarea numerelor întregi in BCD (Binary Coded Decimal)

In acest sistem o cifra zecimala se reprezintă pe o tetradă (4 biţi), deci doua cifre BCD pe octet. Deoarece cu 4 biţi se pot codifica 16 valori distincte si sistemul BCD foloseşte doar 10 dintre acestea, rămân 6 valori ce nu pot fi folosite.

De exemplu: numărul zecimal 675 se va reprezenta pe 2 octeţi astfel: 0000 0110 0111 0101.

Procesoarele dispun de instrucţiuni ce operează cu numere reprezentate în BCD si de instrucţiuni care realizează corecţii ce trebuie făcute după, sau înaintea, operaţiilor in BCD.

Acest format de reprezentare se mai numeşte şi BCD împachetat, pentru al deosebi de formatul BCD despachetat, conform căruia o cifra BCD se reprezintă pe un octet. In acest caz numărul 675 va fi reprezentat pe 3 octeţi. Tetrada superioara a fiecărui octet din reprezentarea BCD despachetat se cere de către microprocesor sa aibă valoarea 0 (zero).

2.1.3. Reprezentarea caracterelor şi a şirurilor de caractere

In categoria datelor alfanumerice intra caracterele alfabetice, caracterele numerice, semnele de punctuaţie şi caracterele de control.

Codificarea acestora se face conform codului standard ASCII (American Standard Code for Information Interchange). Codul ASCII standard este un cod pe 7 biţi (128 combinaţii distincte). Ca atare un caracter ASCII se reprezintă pe un octet in care bitul cel mai semnificativ este zero. Domeniul de valori este de la 0 la 127 (in hexazecimal de la 00h la 07fh).

Exemplu:

Caracterul 'a' are codul 61h, 'A' are codul 41h, '5' are codul 35h, '+' are codul 02Bh.

Şirurile de caractere sunt reprezentate prin succesiuni de octeţi ce cuprind codurile caracterelor respective. De exemplu şirul de caractere “Aa4+b” se reprezintă pe 5 octeţi ce vor conţine 41 61 34 2B 42h.

2.1.4. Reprezentarea numerelor in virgula mobila

Valorile reale (fracţionare) se reprezintă de obicei in virgula mobila, adică sub forma:

$(-1)^s * m * b^e$

unde:

-***s*** este numit bit de semn si este 1 pentru numere negative şi 0 pentru numere pozitive;

-***m*** este numita mantisa şi este un număr pozitiv subunitar reprezentat in baza b

-***b*** este baza de reprezentare a numărului, de obicei 2 sau 16;

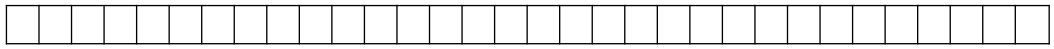
-***e*** este exponentul şi este un număr întreg cu semn.

In binar mantisa este normalizata daca partea întreagă a numărului are o singura cifra şi aceasta este 1. Întrucât in calculator numerele sunt reprezentate in binar b=2, sub forma normalizata prima cifra a mantisei normalizate este 1. Deci numărul 0.00101101 normalizat este $1.01101 * 2^{-3}$

Daca primul bit al mantisei in reprezentarea normalizata este întotdeauna 1, el poate fi subînţeles, deci nu se mai reprezintă, câştigându-se astfel un bit in spaţiul de memorare al mantisei. Acest bit se mai numeşte bit ascuns (hidden bit).

Exponentul fiind un număr întreg cu semn va trebui sa fie reprezentat semnul şi valoarea acestuia. Pentru a evita reprezentarea separata a semnului exponentului, acesta se reprezintă sub forma unei caracteristici, adică sub forma unei constante ***c=k+e*** astfel ca valoarea ***c*** sa fie întotdeauna o valoare pozitiva.

De obicei numerele reale sunt reprezentate in calculator in simpla precizie (pe 32 biţi) sau in dubla precizie (pe 64 biţi). In cazul reprezentării in simpla precizie, constanta k este 127 şi avem:



I s I caracteristica I m a n t i s a I

unde:

-s -bitul de semn al numărului (mantisei) : 0 -pozitiv, 1 -negativ

-caracteristica c = k + e, ($k = 127_{10} = 01111111_2$) se reprezintă pe 8 biţi fiind compusa intre 0 şi 255. Domeniul de reprezentare al exponentului va fi de la – 127 la 128;

- mantisa se reprezintă fizic pe 23 biţi, deci o mantisa de 24 biţi (23 + 1 bit ascuns)

Exemple:

1. Fie numărul real 0.0390625. Se cere reprezentarea lui in simpla precizie. Pe paşi trebuie făcute următoarele operaţii:

-se converteşte numărul dat in binar, adică:

$0.0390625_{10} = 0.0000101_2$

-se normalizează numărul binar, adică:

$0.0000101_2 = 1.01 \times 2^{-5}$

- se calculează semnul, caracteristica şi mantisa:

s = 0, numărul este pozitiv

$c = 127 - 5 = 122_{10} = 01111010_2$

mantisa m= 1.01, eliminând bitul ascuns şi reprezentat pe cei 23 biţi avem:

.01000000000000000000000

in final numărul real 0.0390625_{10} in simpla precizie va fi

0 01111011 0100. 0000000

sau 3d20 0000h

2.Sa se reprezinte in simpla precizie numărul: - 23.0390625

$23.0390625 = 10111.0000101_2$

$10111.0000101_2 = 1.01110000101 * 2^4$

şi avem

s = 1

$c = 127 + 4 = 123_{10} = 10000011_2$

mantisa m pe 23 biţi va fi: m = .01110000101000000000000

In final reprezentarea numărului in binar este :

1 10000011 01110000101000000000000

sau in hexazecimal: c1b85000

3. Ce număr real este reprezentat in virgula mobila
$bef40000_{16}$ = 1011 1110 1111 0100 0000 0000 0000 0000_2 ?

Se vor extrage elementele constitutive:

-semnul s = 1, numărul este negativ

-caracteristica $01111101_2 = 125_{10}$, deci exponentul e = c – 127=125 – 127 = -2

- la mantisa 111 0100 0000 0000 0000 0000

-se adaugă bitul ascuns şi avem 1.111 0100 0000 0000 0000 0000

care ne da $0.111101_2 = 1.2^0+1 . 2^{-1} +1.2^{-2}+1.2^{-3} +1.2^{-4} +0.2^{-5} +1.2^{-6} = 1.953125_{10}$

-înmulţindu-l cu 2^{-2} se obţine:

$0,48835625_{10}$

sau m = 1.111101 de unde valoarea obţinuta a numărului este $1.111101 \times 2^{-2} = 0.0111101_2 = 0.48835625_{10}$

Adăugând semnul, numărul real este: $- 0. 48835625_{10}$

4. Ce număr real este reprezentat in virgula mobila prin:
0100 0011 0101 1010 0000 0000 0000 0000 = $435a 0000_{16}$?

Extragem elementele constitutive:

-s = 0, număr pozitiv:

-caracteristica $c = 10000110_2 = 134$ de unde exponentul e = 134 – 127 = 7

-adăugam bitul ascuns la 10110100. . . . 0 şi obţinem mantisa m = 1. 110110100 . . .0

-valoarea absoluta a numărului este: $1.1101101 \times 2^7 = 11101101.0 = 237_{10}$

Numărul real este 237,00.

2.2. Elemente de memorie

Elementele de memorie sunt următoarele.

Bitul. Cel mai mic element de memorare a unei informaţii este bitul, în care se poate memora o cifra binara, 0 sau 1.

De obicei informaţia de prelucrat se reprezintă pe segmente contigue de biţi denumite tetrade, octeţi, cuvinte, dublu cuvinte, quadwords si tenbytes.

Tetrada. Tetrada este o secvenţă de 4 biţi, numerotaţi 0,1,2,3 de la dreapta la stânga, bitul 0 fiind cel mai puţin semnificativ, iar bitul 3 cel mai semnificativ:

1	0	1	1
3	2	1	0

Octetul (Byte). Octetul sau byte este un element de memorare, ce cuprinde o secvenţa de 8 biţi. Octetul este unul dintre cele mai importante elemente (celule) de memorare adresabile. Cei 8 biţi ai unui octet sunt numerotaţi cu 0,1,2,...7 de la dreapta la stânga:

0	1	1	0	0	0	0	1
7	6	5	4	3	2	1	0

Octetul este format din 2 tetrade, tetradă inferioara (din dreapta) conţine biţii 0, 1, 2, 3, iar cea superioara (din stânga) conţine biţii 4, 5, 6, 7 ai octetului.

Cuvântul(Word). Cuvântul este o secvenţă de 2 octeţi, respectiv 16 biţi, numerotaţi de la dreapta spre stânga, astfel 0, 1, 2 .....14, 15. Bitul cel mai semnificativ este bitul 15. Primul octet(inferior) din cuvânt conţine biţii 0, 1, 2, 3, 4, 5, 6, 7, iar al doilea octet(superior), biţii 7, 8, 9, 10, 11, 12, 13, 14, 15.

1	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Cuvântul poate fi reprezentat printr-un registru de 16 biţi sau în doi octeţi de memorie. In memorie, octetul inferior (biţii 0-7) este memorat la adresa mai mică, iar octetul superior (biţii 8-15) la adresa cea mai mare.

De exemplu cuvântul 4567h se reprezintă intr-un registru de 16 biţi sub forma 4567h, iar în memorie la adresa 1000 sub forma 6745 (octetul 67 la adresa 1000, iar octetul 45 la adresa 1001).

Dublu cuvânt (Double Word). O succesiune de 2 cuvinte (4 octeţi, 32 biţi), reprezintă un dublu cuvânt. Cei 32 de biţi ai unui dublu cuvânt sunt numerotaţi de la dreapta la stânga prin 0, 1, 2, ...30, 31. Bitul cel mai semnificativ este bitul 31, octetul cel mai puţin semnificativ conţine biţii 0-7, iar cel mai semnificativ octet (octetul 4) conţine biţii 23-31.

Un dublu cuvânt poate fi reprezentat într-un registru de 32 biţi sau pe 4 octeţi consecutivi de memorie. In memorie, octetul 1-cel mai puţin semnificativ este memorat la adresa cea mai mica, iar octetul 4-cel mai semnificativ la adresa cea mai mare (în ordinea *little-endian*).

De exemplu dublul cuvânt 12 34 56 78h, aflat la offset-ul 0000, va fi memorat astfel 78 56 34 12, cu octetul 78h la offset-ul 0000, iar octetul 12h la offset-ul 0003.

0000:	78
0001:	56
0002:	34
0003:	12

Quadword. Quadword (qword) este format din 2 dublu cuvinte (4 cuvinte, respectiv 8 octeţi succesivi de memorie). Cei 64 biţi ai unui qword sunt numerotaţi de la dreapta la stânga astfel: 0, 1, 2, ...62, 63. Bitul cel mai semnificativ este bitul 63. In memorie octetul 1 se reprezintă la adresa cea mai mica, iar octetul 8 la adresa cea mai mare.

Tenbyte (10 octeţi)

O succesiune de 10 octeţi formează un tenbyte (tb). Cei 80 de biţi ai elementului sunt numerotaţi de la dreapta la stânga cu 0, 1, 2,.....78, 79. In memorie octetul cel mai puţin semnificativ (biţii 0-7) se reprezintă la adresa cea mai mica, iar octetul 10 (biţii 73-80) la adresa cea mai mare.

2.3 Definirea datelor

În limbajele de asamblare 80x86 se poate opera cu anumite tipuri de date, recunoscute de procesor, acesta dispunând de directive (pseudoinstructiuni) specifice pentru definirea lor.

a) Byte (octet).

Acest tip de date ocupa 8 biţi, adică un octet (byte). Informaţia dintr-un octet poate fi: un *întreg fără semn* cuprins intre 0 si 225, *un întreg cu semn* cuprins intre –128 si 127, sau un *caracter* ASCII.

Definirea datelor de tip byte se face cu ajutorul directivelor BYTE şi SBYTE:

```
value1   BYTE    'A' ; character ASCII
value2   BYTE     0 ;  byte fără semn
value3   BYTE   255 ;  byte fără semn
value4   SBYTE  −128 ;  byte cu semn
value5   SBYTE   +127 ; byte cu semn
value6   BYTE   ?  ; byte nedefinit
```

Definirea datelor de tip byte se face şi cu ajutorul directivei DB (*Define Byte*):

Fie directivele:

```
alfa     DB  65, 72h, 75o, 11011b, 11h+22h, 0ach
         DB  -65, 'a', 'abc'
```

În memorie începând de la adresa simbolica *alfa* (offset, etichetă de date), se va genera secvenţa de octeţi, reprezentata in hexazecimal :

41	72	3d	1b	33	ac	bf	61	61	62	63	
alfa +0	+1	+2	+3	+4						+10	

Valoarea binara 11011b va fi generata la adresa alfa+3.

Definirea şirurilor de caractere:

```
salutare1   BYTE   "Good afternoon",0
greeting2   BYTE   'Good night',0
```

Utilizarea operatorului DUP (duplicate):

```
BYTE   20   DUP(0) ; 20 bytes, toate încărcate  cu zero
BYTE   20   DUP(?) ; 20 bytes, nedefiniţi
BYTE    4   DUP("STACK") ; 20 bytes: "STACKSTACKSTACKSTACK"
```

b) WORD (cuvânt).

Un cuvânt ocupa doi octeţi (16 biţi) si poate fi reprezentat intr-un registru de 16 biţi sau in 2 octeţi consecutivi de memorie. Numerotarea biţilor in cadrul unui cuvânt se face de la 0 la 15 (bitul 15 e bitul cel mai semnificativ al cuvântului, iar bitul 0 este bitul cel mai puţin semnificativ), numerotarea se face de la dreapta la stânga:

Informaţia memorata intr-un cuvânt poate fi :

-un întreg pe 16 biţi cu semn (bitul 15 este bitul de semn), cuprins intre -2^{15} si $2^{15}-1$,

- un întreg pe 16 biţi fără semn, cuprins intre 0 si 2^{16}

- o adresa de memorie de 16 biţi.

Reprezentarea celor 2 octeţi ai cuvântului in memorie se face astfel încât octetul cel mai puţin semnificativ este memorat la adresa cea mai mica. De exemplu: daca valoarea 2345h este memorata la adresa 2000h, atunci octetul 45h se va afla la adresa 2000h, iar octetul 23h la adresa 2001h.

Generarea datelor de tip cuvânt se poate face folosind directivele de tip WORD şi SWORD:

```
word1   WORD     65535 ;  întreg pe 16 biţi fără semn
word2   SWORD   -32768 ; întreg pe 16 biţi cu semn
word3   WORD ?           ; neiniţializat
```

Generarea datelor de tip cuvânt se poate face şi cu directiva de tip DW (*Define Word*):

Fie secvenţa de directive:

```
beta    DW     4567h,   0bc4ah,   1110111011b,   2476o
        DW    -7683,    7683,     'ab'
```

In memorie de la adresa “beta” se vor genera octeţii:

67	45	4a	bc	bb	03	3e	05	fd	e1	03	e1	62	61
beta		+2		+4		+6		+8				+12	

Constanta octala 2476o este generată de la adresa beta +6.

c) **Double WORD**(dublu cuvânt)

Un dublu cuvânt ocupa 2 cuvinte sau 4 octeţi (32 biţi) si poate fi reprezentat in memorie pe 4 octeţi consecutivi, într-o pereche de registre de 16 biţi sau într-un registru de 32 biţi (la procesoarele de 32 biţi).

Informaţia memorata într-un dublu cuvânt poate fi:

- un întreg pe 32 biţi, cu sau fără semn;
- un număr real in simplă precizie;
- sau o adresă fizică de memorie de 32 biţi.

Generarea datelor de tip dublu cuvânt se poate face folosind directivele DWORD şi SDWORD:

```
val1   DWORD     12345678h ; fără semn
val2   SDWORD   −21474836 ; cu semn
val3   DWORD  20  DUP(?) ; fără semn
```

Generarea datelor de tip dublu cuvânt se face şi cu directiva DD (*Define Double Word*):

Reprezentarea celor doua cuvinte a unui dublu cuvânt de memorie se face astfel încât cuvântul cel mai puţin semnificativ este memorat la adresa cea mai mica. De exemplu dublul cuvânt 12345678 h, aflat la adresa 2000h se memorează astfel: cuvântul 5678h se memorează la adresa 2000h, iar cuvântul 1234h la adresa 2002h.

Secvenţa de directive :

```
val1   DD   12345678h ; fără semn
val2   DD   −21474836 ; cu semn
```

d) QUAD – WORD (8 octeţi)

Tipul Quad – word (QWORD) ocupa 8 octeţi şi este reprezentat in memorie pe 64 biţi sau într-o pereche de registre de 32 biţi (în cazul procesoarelor de 32 biţi), sau într-un registru pe 64 biţi.

Informaţia stocata intr-un qword poate fi: un întreg cu sau fără semn pe 64 biţi, sau un număr real în dublă precizie.

Generarea unor date de tip qword se face cu ajutorul directivei QWORD:

```
quad1 QWORD 1234567812345678h
```

Generarea unor date de tip qword se face şi cu directiva DQ (*Define Quad – word*):

```
quad1 DQ 1234567812345678h
```

Reprezentarea in memorie a celor 8 octeţi ai unui qword se face astfel încât octetul cel mai puţin semnificativ este memorat la adresa cea mai mica.

e) Ten Bytes

Valorile Ten – byte (tbyte) ocupă 10 octeţi consecutivi de memorie, sau unul din registrele coprocesorului matematic.

Informaţia stocata intr-un tbyte poate fi: un număr întreg reprezentat ca o secvenţa de 18 cifre (9 octeţi) BCD (format împachetat) cu sau fără semn, sau un număr real in precizie extinsa. Octetul superior conţine semnul numărului, dacă are valoarea 80h – numărul este negativ, iar dacă are valoarea 00h- este pozitiv.

Generarea unor date de tip tbyte se face cu directiva TBYTE, de ex. valoarea zecimală -1234:

```
intVal    TBYTE   80000000000000001234h
```

Generarea datelor de tip tbyte se face şi cu directiva DT (*Define Ten Bytes*):

```
intVal    DT  80000000000000001234h
```

În format BCD împachetat fiecare cifra zecimală se reprezintă pe o tetradă (4 biţi), deci 2 cifre BCD pe octet. Un întreg BCD se poate reprezenta cu maxim 19 cifre zecimale, care ocupă 76 biţi. Ultima tetradă aflată la adresa cea mai mare este destinată memorării semnului.

f) Definirea datelor în virgulă mobilă

Definirea datelor în virgulă mobilă se face cu directivele:

- REAL4 – defineşte variabile în virgulă mobilă, în simpla precizie pe 32 biţi;
- REAL8 – defineşte variabile în virgulă mobilă, în dubla precizie pe 64 biţi;

- REAL10 – defineşte variabile în virgulă mobilă, cu precizie extinsă pe 80 biţi.

```
rVal1   REAL4   -1.2
rVal2   REAL8   3.2E-260
rVal3   REAL10   4.6E+4096
ShortArray   REAL4 20 DUP(0.0)
```

Definirea datelor în virgulă mobilă se face şi cu directivele:

```
rVal1   DD   -1.2
rVal2   DQ   3.2E-260
rVal3   DT   4.6E+4096
```

Gama valorilor definite pot fi:

REAL4 - 1.18 $x10^{-38}$ până 3.40 $x10^{38}$
REAL8 - 2.23$x10^{-308}$ până 1.79$x10^{308}$
REAL10 - 3.37$x10^{-4932}$ până 1.18$x10^{4932}$

Pentru exemplificarea utilizarii directivelor de generare de date, consideram urmatorul program:

Exemplul 1.

```
INCLUDE Irvine32.inc
.data
alfa        byte      40+25,72h,75o,11011b,15+22h,0ach
            sbyte     -65
            byte      'a','abc'
beta        word      4567h,0bc4ah,1110111011b,2476o
            dw        -7683,7683,'ab'
gama        dword     12345678h,1
            sdword    -1,-1.0,-0.23828125
            dword     1.0,0.0390625
gama1       dword     gama
qw          qword     2,-2,2.5,-2.5
t1          tbyte     888888888777777777h,8055555555555555555h
tb          tbyte     456712345678901234 56h,8045671234567890 1234h
f           tbyte     3345,-3345,1.0,-1.0

.code
      main proc

      exit
main ENDP
END main
```

Fisierul listing obtinut (Project.lst) dupa asamblare arata astfel (de la rândul 765):

```
                                C .LIST
                                 C
00000000                .data
00000000 41 72 3D 1B 31 alfa            byte    40+25,72h,75o,11011b,15+22h,0ach
       AC
00000006  BF                            sbyte   -65
00000007  61 61 62 63                   byte    'a','abc'
0000000B 4567 BC4A 03BB beta            word    4567h,0bc4ah,1110111011b,2476o
       053E
00000013  E1FD 1E03 6162                dw      -7683,7683,'ab'
00000019 12345678               gama    dword   12345678h,1
       00000001
```

```
00000021  FFFFFFFF                    sdword   -1,-1.0,-0.23828125
          BF800000
          BE740000
0000002D  3F800000                    dword    1.0,0.0390625
          3D200000
00000035 00000019 R       gama1       dword    gama
00000039                  qw          qword    2,-2,2.5,-2.5
          0000000000000002
          FFFFFFFFFFFFFFFE
          4004000000000000
          C004000000000000
00000059                  t1          tbyte    88888888888777777777h,80555555555555555555h
          08888888888777777777
          80555555555555555555
0000006D                  tb          tbyte    45671234567890123456h,80456712345678901234h
          45671234567890123456
          80456712345678901234
00000081                  f           tbyte    3345,-3345,1.0,-1.0
          00000000000000000D11
          FFFFFFFFFFFFFFFF2EF
          3FFF8000000000000000
          BFFF8000000000000000

00000000                  .code
00000000                      main proc

                              exit
00000000  6A 00    *      push   +000000000h
00000002  E8 00000000 E   *       call   ExitProcess
00000007                  main ENDP
                      END main
_Microsoft (R) Macro Assembler Version 12.00.31101.0          02/16/15 19:52:47
..\lab21.asm                                Symbols 2 - 1
```

Prima coloana reprezinta adresa relativa (offset) în interiorul segmentului, adresa exprimata in hexazecimal pe 8 cifre hexa. Urmeaza apoi continutul campului de date corespunzator unei linii. Valorile intregi, negative se reprezinta în complement faţă de doi, iar cele reale sunt reprezentate în virgulă mobilă.

Prezenta caracterului R în aceasta zona indică faptul că este vorba despre deplasamentul 0019h, adică adresa relativa a variabilei *gama*.

La intregii BCD de la variabilele t1, tb se poate observa semnul (bitul de semn).
In fisierul .lst nu apare pozitia fizica a octetilor in memorie.

2.4 Moduri de adresare

Modurile de adresare specifica modul in care se calculeaza adresa operatorului aflat in memorie. Adresa fizica a unui operand este formata din adresa de segment si adresa efectiva (offsetul). Adresa de segment este furnizata de unul din cele 4 registre de segment si indica inceputul segmentului in care se gaseste operandul. Adresa efectiva calculata pe baza modurilor de adresare, dau offsetul (deplasamentul) operatorului in cadrul segmentului respectiv. In cazul instructiunilor cu 2 operanzi, obligatoriu unul din operanzi este continut intr-un registru general.

a) Modurile de adresare la procesoarele de 16 biti

1. **Adresare imediata**. Operatorul este o constanta si apare in formatul instructiunii. De exemplu:

```
mov   bx , 55h; transfera in bx valoarea hexazecimala 55
add    bx, 10011b; aduna la Bx valoarea binara 10011
```

2. **Adresare directa**. Adresa efectiva a operandului este data de un deplasament in segmentul curent. Deplasamentul este parte componenta a instructiunii. Exemplu:

```
alfa     db   167o
.....
mov    al, alfa; transfera in al continutul variabilei alfa
mov    dx, [135]; transfera in cx continutul adresei 135
```

Acest mod de adresare foloseste registrul DS implicit ca registru de segment.

3)**Adresare indirecta**

Adresa efectiva a operandului este data de continutul registrelor: bp, bx, si sau di. Registrul segment implicit este DS.

```
mov  al, [si] ; transfera in al continutul octetului de la adresa continuta de registrul si
mov  [bx] , ax ; transfera continutul lui ax in locatia a carei adresa se afla in bx.
sub  byte  ptr [si] , 56h ; scade 56h din octetul aflat la adresa data de si
```

4) **Adresare bazata sau indexata**

Adresa efectiva se obtine adunand la unul din registrele de baza bx sau bp, sau la unul din registrele de index SI sau DI un deplasament pe 8 sau 16 biti. Registrul segment implicit este DS (daca se folosesc registrele BX, SI sau DI) si registrul SS(daca se foloseste BP).

In scrierea operanzilor adresati sub aceasta forma, se pot folosi diverse modalitati, echivalente din punct de vedere al mecanismului de adresare.

Exemplu:

```
alfa   dw   0a4bh, 35, 64, 157o
.....
1.   mov   bx, 100
     mov   ax, alfa [bx]
     mov   ax, bx [alfa]
     mov   ax, [bx + alfa]
     mov   ax, [bx] . alfa

2.     sir   db   20 dup(?)
          .....
              xor  si, si
              .....
          add  al, sir[si] ; echivalent cu si[sir] sau [sir+si] etc.
          inc si
```

5) **Adresare bazata si indexata**

Adresa efectiva este formata prin adunarea unuia din registrele de baza (BX sau BP) cu unul din registrele index (SI si DI) si cu un deplasament de 8 sau 16 biti. Registrele segment implicite sint DS (daca se foloseste BX cu SI sau cu DI) sau SS (daca se foloseste BP cu SI sau DI). Deplasamentul poate fi si nul. Registrele BX, BP, SI si DI sint incluse intre paranteze drepte [ ]. Exemplu:

```
mov   ax, alfa [bx] [si]
mov   ax, [bx + si + 8]
add    ax, [bx + di] . 88h
adc    ax, [bp] [di] [79]
```

Folosirea prefixelor de segment permite asignarea unui alt registru segment decit cel implicit. Prefixul de segment este un octet care apare inaintea codului instructiunii si care identifica explicit

registrul de segment folosit. In textul sursa se scrie un registru de segment, urmat de doua puncte inaintea operandului aflat in memorie. Exemplu:

```
mov   ax,  ds : [bp] [si]
adc    ax,  cs : alfa [di] [bx + 55h]
mov   bx,  ss : [bp]
```

Tabelul urmator sintetizeaza modurile de adresare si registrul segment implicit:

Modul de adresare	Formatul operandului	Registrul segment implicit
imediat	constanta	-----
direct	variabila	DS
registru indirect	BX BP SI DI	DS SS DS DS
bazat	variabila [BX] variabila [BP]	DS SS
indexat	variabila [SI] variabila [DI]	DS DS
bazat indexat	variabila [BX] [SI] variabila [BX + DI] variabila [BP + SI] variabila [BP + DI]	DS DS SS SS

b) *Moduri de adresare pe 32 de biti*

Posibilitatile de specificare ale operanzilor in cazul procesoarelor pe 32 biti sint mult mai multe. In acest caz deplasamentul este o valoare imediata pe 8 sau 32 de biti, registrul de baza este orice registru general de 32 biti, registru de index poate fi orice registru generat de 32 biti, cu exceptia registrului EPS. In modurile de adresare pe 32 biti apare notiunea de factor de scala, care este un intreg ce poate avea valoarea 1, 2, 4 sau 8, valoare cu care poate fi inmultit indexul. Cele 9 moduri de adresare sint:

Adresare directa – adresa efectiva a operandului face parte din instructiune, putind fi reprezentata pe 8, 16, sau 32 biti. Exemplu:

```
dec    dword    ptr   [1265h]
adc     eax , alfa
```

Adresare indirecta prin registre – adresa efectiva a operandului este continuta intr-unul din registrele generale. Exemplu:

```
mov   [eax] ,   ecx
sub    [ebx] ,  edx
```

Adresare bazata – adresa efectiva a operandului este formata din continutul unui registru de baza la care se poate adauga un deplasament. Ex.:

```
add     ecx ,   [eax + 0a4h]
```

Adresare indexata – adresa efectiva a operandului este formata din continutul unui registru de index la care se poate adăuga un deplasament. Ex.:

```
inc   word  ptr  alfa[edi]
```

Adresare indexata cu factor de scala – adresa efectiva a operandului se obtine din continutul unui registru de index, inmultit cu un factor de scala la care se poate aduna un deplasament. Exemplu:

```
mul  byte ptr sir [edi* 4] [126 h]
```

Adresare bazata indexata – adresa efectiva a operandului se formeaza prin adunarea continutului unui registru de baza la continutul unui registru de indexare. Ex.:

```
add    eax ,    [esi] [ebx]
```

Adresare bazata indexata cu factor de scala – adresa efectiva a operandului este formata din continutul unui registru de baza la care se adauga continutul unui registru de index inmultit cu un factor de scala. Ex.:

```
sub   eax ,   [ecx] [edi * 8]
```

Adresare bazata, indexata cu deplasament – adresa efectiva a operandului se obtine prin adunarea continutului unui registru de baza cu continutul unui registru de indexare si a unui deplasament. Ex.:

```
adc   edx ,   [esi] [ebp + 2163 h]
```

Adresare bazata indexata cu factor de scala si depasament – adresa efectiva a operandului se obtine prin adunarea continutului unui registru de baza la continutul unui registru de index multiplicat cu un factor de scala si a unui deplasament. Ex.:

```
mov   alfa[esi * 8] [edp + 0a3bh] , eax
```

Factorul de scala este util la preluarea tablourilor de date, permitind ca indicele tabloului sa coincida cu continutul unui registru de indexare. De ex. instructiunea in C:

```
long  int  sir [25] ;
for  ( i = 0 ; i < 25 ; i + + )
     sir [i] + = 65 h ;
```

se poate scrie in asamblare :

```
      mov cx,25
      xor esi,esi
reia: add sir[esi*4],65h
      inc esi
      loop reia
```

Registrul *esi* are aceeasi semnificatie ca si indicele i din programul C.

2.5 Aritmetica binara

Aceste instrucţiuni modifica conţinutul registrului FLAGS.

a) Instrucţiunea ADD (Add)

Forma generală:

ADD dest, sursa ; [dest] ← [dest] + [sursa]

unde:

- *dest* poate fi un registru general sau o locaţie de memorie;
- *sursa* poate fi un registru general, o locaţie de memorie sau o constantă.

Operanzii au aceeaşi structură ca la instrucţiunea MOV. Cei doi operanzi nu pot fi simultan locaţii de memorie.

Operaţia se poate efectua pe 8,16, 32 sau pe 64 biţi. Cei doi operanzi trebuie sa aibă aceeaşi dimensiune (acelaşi tip). În caz de ambiguitate se va folosi operatorul PTR.

Indicatorii afectaţi sunt: AF, CF, PF, SF, ZF şi OF

Exemple:

```
add  ax, 5
add  bl, 5
add  ax, bx
add  word ptr [bx], 75
add  alfa, ax
add  alfa, 5
add  byte ptr [si], 75
add  byte ptr  alfa, 75
```

```
.data
var1 DWORD 10000h
var2 DWORD 20000h
.code
```

```
mov eax,var1 ; EAX = 10000h
add eax,var2 ; EAX = 30000h
```

```
.data
sum qword 0

.code
    mov rax,5
    add  rax,6
    mov   sum,rax
```

b) Instrucţiunea ADC (Add with Carry)

Forma generală:

ADC dest, sursa ; [dest) ← [dest] + [sursa] + [CF]

Unde *dest* şi *sursa* au aceeasi semnificaţie ca la instrucţiunea ADD, iar CF este Carry Flag.
Instrucţiunea adună conţinutul *dest* cu conţinutul *sursei* şi cu bitul de transport CF. Indicatorii afectaţi sunt aceeaşi de la instrucţiunea ADD.

Operaţia ADC se foloseşte la adunări de operanzi pe mai multe cuvinte, operaţie în care poate apărea transport de care trebuie să se ţină seama.

Exemplu 1. Să se adune doua numere op1, op2 pe 2 cuvinte (dword).

```
op1  dword    12345678h
op2  dword    0abcdefgh
rez   dword    ?
.....................................
mov ax, word ptr op1
add  ax, word ptr op2
mov word ptr rez, ax
mov ax, word ptr op1+2
adc  ax, word ptr op2+2;      se considera eventualul transport
mov word ptr rez+2, ax
```

c) Instrucţiunea SUB (Substrat)

Forma generală SUB(scăderea):

SUB dest, sursa ; (dest) ← (dest) – (sursa)

unde *dest* şi *sursa* au aceeaşi semnificaţie ca la instrucţiunea ADD Indicatorii afectaţi sunt cei specificaţi la ADD. Structura operanzilor ca la instrucţiunea MOV.

```
.data
var1 DWORD 30000h
var2 DWORD 10000h
.code
mov eax,var1 ; EAX = 30000h
sub eax,var2 ; EAX = 20000h
```

d) Instrucţiunea SBB (Substrat with Borrow)

Forma generală SBB (scădere cu eventualul împrumut):

SBB dest, sursa ; [det] ← [dest] – [sursa] – [CF]

unde semnificaţia *dest*, *sursa* şi CF sunt cele prezentate la ADC. Instrucţiunea SBB ia în considerare eventualul împrumut. Exemplu:

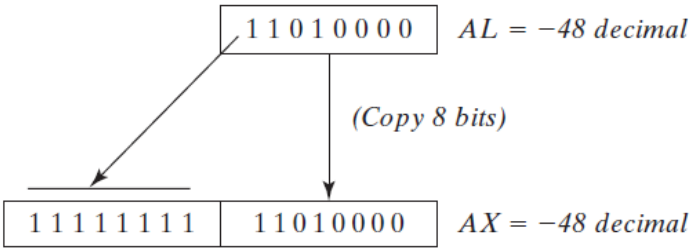
```
Op1 dword    12345678h
Op2 dword    0abcdefgh90h
Rez dword    ?
. . . . . .
mov ax, word  ptr  op1
sub  ax,  word  ptr  op2
```

```
mov word  ptr  rez, ax
mov ax, word  ptr  op1 + 2
sbb  ax, word  ptr  op2 + 2 ; se considera eventualul împrumut
mov word  ptr  rez + 2, ax
```

i) Instrucţiunea CBW (Convert Byte to Word)

Are ca efect extinderea bitului de semn (AL_7) din AL la întreg registru AH, adică:

daca bitul de semn $AL_7 = 0$ atunci [ah] ← 00h
altfel [ah] ← 0ffh.



Instrucţiunea nu are operanzi şi nu afectează indicatorii de condiţie.

Exemplu. Se cere să se adune un număr întreg cu semn reprezentat pe octet cu un număr întreg cu semn pe cuvânt.

```
a     sbyte     -75
b     sword   -188
c     sword     ?
. . . . . .
mov al, a
cbw ;             converteşte octetul la cuvânt
add  ax, b
mov c, ax
. . . . . .
```

j) Instrucţiunea CWD (Convert Word to Double Word)

Are ca efect extinderea bitului de semn din AX (AX_{15}) la întreg registrul DX, obţinându-se astfel AX pe 32 de biţi, adică:

daca semnul $[AX_{15}] = 0$ atunci [dx] ← 0000h,
altfel [dx] ← 0ffffh.

Instrucţiunea nu are operanzi şi nu afectează indicatorii de condiţie.

Exemplu. Se cere diferenţa dintre un operand reprezentat pe 2 cuvinte (do) şi unul reprezentat pe cuvânt (so)

```
do   dword    12345678h
so   word      0abcdh
rez  dword    ?
. . . . . .
mov ax, so
cwd        ; operandul so reprezentat în DX : AX
mov bx, ax     ; salvează ax în bx
mov ax, word  ptr do
sub  ax, bx
mov word  ptr  rez, ax
mov ax, word  ptr  do + 2
sbb  ax, dx     ; ia în considerare eventualul transport
mov word  ptr  rez + 2
```

Instrucţiunea CDQ (convert doubleword to quadword)

Are ca efect extinderea bitului de semn din EAX (EAX_{15}) la întreg registrul EDX, obţinându-se astfel EAX pe 64 de biţi, adică:

daca semnul [EAX_{15}] = 0 atunci [EDX] ← 00000000h,
altfel [edx] ← 0ffffffffh.

Instrucţiunea nu are operanzi şi nu afectează indicatorii de condiţie.

k) Instrucţiunea MUL (Multiply)

Forma generală:

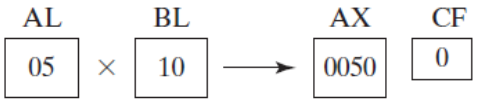
```
MUL reg/mem8
MUL reg/mem16
MUL reg/mem32
```

unde *reg* poate fi un registru sau o locaţie de memorie *mem* de 8, 16, 32 biţi. Rezultatul se obţine pe un număr dublu de biţi (16, 32, 64). Operaţia realizată este produsul intre acumulator şi sursa cu depunerea rezultatului în acumulatorul extins. Cei doi operanzi se consideră numere fără semn.
Dacă *sursa* este pe octet avem:

[AX] ← [AL] * [*reg/mem8*]

```
mov al,5h
mov bl,10h
mul bl ; AX = 0050h, CF = 0
```

Diagrama ilustrează interacţiunea dintre registre:

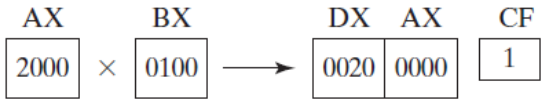


Dacă sursa este pe cuvânt avem:

[DX:AX] ← [AX] * [*reg/mem16*]

```
.data
val1 WORD 2000h
val2 WORD 0100h
.code
mov ax,val1 ; AX = 2000h
mul val2 ; DX:AX = 00200000h, CF = 1
```

Diagrama ilustrează interacţiunea dintre registre:



Dacă sursa este pe 32 biţi avem:

[EDX:EAX] ← [EAX] * [*reg/mem32*]

```
mov eax,12345h
mov ebx,1000h
mul ebx ; EDX:EAX = 0000000012345000h, CF = 0
```

Diagrama ilustrează interacţiunea dintre registre:

iar dacă sursa este pe 64 biţi avem:

[RDX:RAX] ← [RAX] * [*reg/mem64*]

```
mov rax,0FFFF0000FFFF0000h
mov rbx,2
mul rbx ; RDX:RAX = 0000000000000001FFFE0001FFFE0000
```

Afectează indicatorii CF şi OF, ceilalţi sunt nedefiniţi.

l) Instrucţiunea IMUL (Integer Multiply)

Instrucţiunea IMUL semnifică înmulţirea cu semn. Instrucţiunea poate avea 1, 2, sau 3operanzi. Afectează indicatorii CF şi OF, restul sunt nedefiniţi.

Structura cu un operand:

```
IMUL reg/mem8 ; AX = AL * reg/mem8
IMUL reg/mem16 ; DX:AX = AX * reg/mem16
IMUL reg/mem32 ; EDX:EAX = EAX * reg/mem32
```

Structura cu doi operanzi. Structura cu doi operanzi **trunchiază** produsul la lăţimea registrului de destinaţie. Dacă cifrele semnificative sunt pierdute, se setează indicatorii CF şi OF.

```
IMUL reg16,reg/mem16
IMUL reg16,imm8
IMUL reg16,imm16

IMUL reg32,reg/mem32
IMUL reg32,imm8
IMUL reg32,imm32
```

Structura cu 3 operanzi - op1=op2*op3 (**trunchiază** produsul):

```
IMUL reg16,reg/mem16,imm8
IMUL reg16,reg/mem16,imm16

IMUL reg32,reg/mem32,imm8
IMUL reg32,reg/mem32,imm32
```

Exemplu cu 2 operanzi:

```
   .data
word1 SWORD 4
dword1 SDWORD 4
.code
mov ax,-16 ; AX = -16
mov bx,2 ; BX = 2
imul bx,ax ; BX = -32
imul bx,2 ; BX = -64
imul bx,word1 ; BX = -256
mov eax,-16 ; EAX = -16
mov ebx,2 ; EBX = 2
imul ebx,eax ; EBX = -32
imul ebx,2 ; EBX = -64
imul ebx,dword1 ; EBX = -256
```

Exemplu cu 3 operanzi:

```
.data
```

```
word1 SWORD 4
dword1 SDWORD 4
.code
imul bx,word1,-16 ; BX = word1 * -16
imul ebx,dword1,-16 ; EBX = dword1 * -16
imul ebx,dword1,-2000000000 ; signed overflow!
```

m) Intructiunea DIV (Divide)

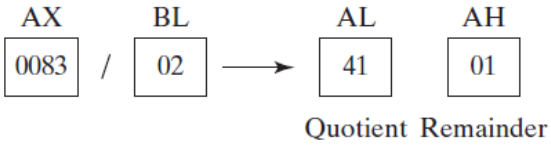
Forma generală:

DIV sursa

unde *sursa* este un registru sau o locaţie de memorie, reprezentata pe octet, cuvânt sau 32 biţi. Instrucţiunea realizează împărţirea fără semn intre deîmpărţit şi împărţitor. Daca împărţitorul (sursa) este reprezentat pe octet atunci deîmpărţitul este AX şi rezultatul este: câtul în **al** iar restul în **ah**, adică:

DIV sursa ; [al] ←[ax]/ [sursa]
; [ah] ←restul împărţirii [ax]/ [sursa]

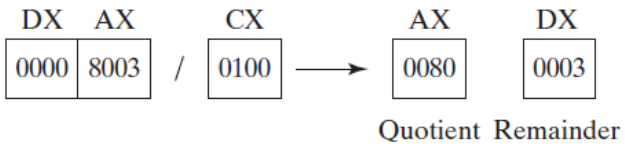
```
mov ax,0083h ; deîmpărţitul
mov bl,2 ; împărţitorul
div bl ; AL = 41h-catul, AH = 01h-restul
```



Daca împărţitorul (sursa) este reprezentat pe cuvânt atunci deîmpărţitul este considerat în DX şi AX, câtul se obţine în AX iar restul în DX, adică

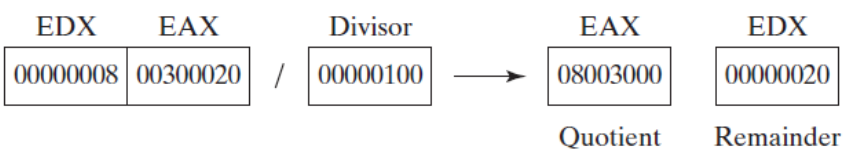
DIV sursa ; (ax) ←câtul împărţirii (dx:ax)/(sursa)
; (dx) ←restul împărţirii (dx:ax)/(sursa)

```
mov dx,0 ; clear deîmpartitul, high
mov ax,8003h ; deîmpartitul, low
mov cx,100h ; împartitorul
div cx ; AX = 0080h-câtul, DX = 0003h-restul
```



Daca împărţitorul (sursa) este reprezentat pe 32 biţi atunci deîmpărţitul este considerat în EDX şi EAX (64 biţi), câtul se obţine în EAX iar restul în EDX

```
.data
dividend QWORD 0000000800300020h
divisor DWORD 00000100h
.code
mov edx,DWORD PTR dividend + 4 ; high doubleword
mov eax,DWORD PTR dividend ; low doubleword
div divisor ; EAX = 08003000h, EDX = 00000020h
```



Daca împărţitorul (sursa) este reprezentat pe 64 biţi atunci deîmpărţitul este considerat în RDX şi RAX (64 biţi), câtul se obţine în RAX iar restul în RDX

```
.data
dividend_hi QWORD 0000000000000108h
dividend_lo QWORD 0000000033300020h
divisor QWORD 0000000000010000h
.code
mov rdx, dividend_hi
mov rax, dividend_lo
div divisor ; RAX = 0108000000003330
; RDX = 0000000000000020
```

Toţi indicatorii nu sunt definiţi. Operaţia de împărţire poate conduce la depăsiri, dacă câtul depăşeşte valoarea maximă reprezentabilă pe 8, respectiv pe 16 biţi sau daca împărţitorul este 0.

n) Instrucţiunea IDIV (Integer Divide)

Forma generală:

IDIV sursa

Semnificaţia instrucţiunii şi a operandului *sursa* este aceeasi ca la Instrucţiunea DIV, cu o singură diferenţă importantă – deîmpărţitul cu semn trebuie să fie extins înainte ca împărţirea să fie executată. Extinderea semnului se execută cu instrucţiunile ***CBW, CWD, CDQ.***

Indicatorii sunt nedefiniţi. Operaţia poate conduce la depăsiri.

Ex. Împărţim -48 la +5

```
.data
byteVal SBYTE –48 ; D0 hexadecimal
.code
mov al,byteVal ; partea inferioară a deîmpărţitului
cbw ; extindem AL in AH
mov bl,+5 ; împărţitorul
idiv bl ; AL = –9 – câtul, AH = –3 – restul
```

3. Desfasurarea lucrarii de laborator.

3.1. Se cere obţinerea fisierului executabil pentru următoarea porţiune de cod si rularea apoi pas cu pas.

```
INCLUDE Irvine32.inc
.data
alfa   DW    3 DUP(?)
.code
main proc
mov    ax,17  ; adresare indirecta a operandului sursa
mov    ax,10101b   ;
mov    ax,11b      ;
mov    ax,21o      ;
mov    alfa,ax       ; Adresare directa a operandului destinatie
mov    cx,ax      ; Interschimba registrele ax si bx
mov    ax,bx      ; Folosind registrul cx
mov    ax,cx      ;
xchg   ax,bx      ; Interschimba direct cele 2 registre.
mov    si,2
mov    alfa[si],ax  ; Adresare relativa cu registrul si
mov    esi,2
mov    ebx,offset alfa ; Adresare imediata a operandului sursa
lea    ebx,alfa       ; Acelasi efect
mov    ecx,[ebx][esi]  ; Adresare bazata indexata a sursei
mov    cx,alfa[2]   ; Acelasi efect.
mov    cx,[alfa+2]  ; Acelasi efect
mov    di,4
mov    byte ptr [ebx][edi],55h  ;
```

```
mov     esi,2
mov     ebx,3
mov     alfa[ebx][esi],33h  ; Adresare bazata indexata relativa a
                            ; destinatiei
mov     alfa[ebx+esi],33h   ; Notatii echivalente
mov     [alfa+ebx+esi],33h
mov     [ebx][esi]+alfa,33h

exit
main ENDP
END main
```

3.2. Sa se calculeze expresia aritmetica: e=((a+b*c-d)/f+g*h)/i. Se cere obţinerea fisierului executabil si rularea apoi pas cu pas.

```
INCLUDE Irvine32.inc

; Sa se calculeze expresia aritmetica:  e=((a+b*c-d)/f+g*h)/i
; se considera    a, d, f – cuvant  b, c, g, h, i –byte
; ca sa putem executa impartirea cu f convertim impartitorul la dublucuvânt
; ne vor interesa doar caturile impartirilor,  rezultatul va fi de tip octet

.data
      a dw 5
      b db 6
      cd db 10
      d dw 5
      f dw 6
      g db 10
      h db 11
      i db 10
      interm dw ?
      rez db ?

.code
 main proc
      mov eax,0
      mov al, b
      imul cd                 ; in ax avem b*c
      add ax, a         ; ax=b*c+a
      sub ax, d         ; ax=b*c+a-d
      cwd         ; am convertit cuvantul din ax, in dublu cuvantul , retinut in dx:ax
      idiv f              ; obtinem câtul în ax  si restul în dx ax=(a+b*c-d)/f
      mov interm, ax; interm=(a+b*c-d)/f
      mov al, g
      imul h                  ; ax=g*h
      add ax, interm          ; ax=(a+b*c-d)/f+g*h
      idiv i              ; se obtine catul în al si restul în ah
      mov rez, al

      exit
      main ENDP
      END main
```

*; Date de test : vom obţine rezultatul ((a+b*c-d)/f+g*h)/i=((5+6*10-5)/6+10*11)/10= 12*

4 Probleme propuse

Să se calculeze expresiile :

1. *z=15/(a*a+b*b-5)+24/(a*a-b*b+4)*
2. *z=a+b*b-(36/(b*b)/(1+(25/(b*b)))*
3. *z=(3+(c*c))/(6-(b*b))+((a*a-b*b)/(a*a+c*c))*
4. *z=(a*3+b*b*5)/(a*a+2*a*b)-a-b*
5. *z=(a+b+c+1) *(a+b+c+1) /((a-b+d)*(a-b+d))*
6. *z=(a-b*c/d)/(c+2-a/b)+5*
7. *z=(5*a-b/7)/(3/b+a*a)*
8. $z=(a+b+c+1)^3/(a-b*c+d)^2$
9. $z=((a+1)*(a+1)+2)^2/(b*b+c*c)$
10. *z=(a*a+b*b)/(a*a-b*b-5).*
11. *z=(5*a-b/7)/(3/b+a*a).*
12. *z=(2+1/a)/(3+1/(b*b))-1/(c*c)*
13. *z=((a+b)/c + 2*d)/e*
14. *z= ((a*c-b*d)/f +(a+b)*c/d)/h*
15. *z=((a+b*c-d)/f+h)/g*

5 Continutul referatului

Referatul va contine codurile din p.3.1, 3.2 şi una din problemele propuse (punctul 4):

- enuntul problemei;
- fişierele listing (de la rândul 765) comentate din 3.1., 3.2, şi codul sursă *.asm comentat (punctul 4);
- concluzii.