Seminar 1 ASC

Introducere în limbajul de asamblare IA-32. Conversia numerelor între bazele de numerație 2, 10, 16. Reprezentarea numerelor întregi în memorie. Instrucțiuni cu și fără semn.

1 Conținut

Arhitectura microprocesorului IA-32 a fost introdusă de Intel în 1985 pentru microprocesoarele 80386. Este un model abstract ce specifică elementele microprocesorului, structura și setul de instrucțiuni. IA-32 este o arhitectură pe 32 de biți (elementele principale au dimensiunea de 32 de biți) și se bazează pe versiunea precedentă Intel 8086.

1.1 Elementele limbajului de asamblare IA-32

Un algoritm este o secvență de pași / operații necesare pentru a rezolva o problemă specifică. De exemplu, algoritmul care rezolvă ecuația de gradul doi $ax^2 + bx + c = 0$ are pașii:

1. Determină Δ;
2. dacă $\Delta \geq 0$, calculează soluţiile x_1şi x_2.

În afară de această secvență de pași/operații, un algoritm include un set de date/entităţi care sunt manipulate de aceşti paşi/operaţii. Pentru exemplul nostru, datele manipulate de algoritm sunt: a, b, c, Δ, x_1şi x_2.
În consecință un algoritm manipulează un set de date/entități cu ajutorul unei secvenţe de paşi/operaţii.
Un algoritm poate fi descris folosind limbajul natural sau într-un limbaj de programare (ex. C, Java, etc.).
Când se specifică un algoritm într-un limbaj de programare, acest algoritm este un program. În mod similar un program conține:

- un set de date / entități;
- un set de operații / instrucțiuni.

Pe parcursul acestui semestru vom studia limbajul de asamblare IA-32. Vom descrie datele manipulate de limbaj și operațiile (instrucțiunile) limbajului.
Toate datele utilizate într-un program IA-32 sunt numerice (numere întregi) și există trei tipuri de date (figura 1 prezintă tipurile de date):

1. *byte* (octet) – tip de data reprezentat pe opt biți;
2. *word* (cuvânt) – tip de dată reprezentat pe 16 biți (2 octeti);
3. *doubleword* (dublucuvânt) – tip de data reprezentat pe 32 de biți (4 octeti);
4. *quadword* – tip de data reprezentat pe 64 de biti (8 octeti).

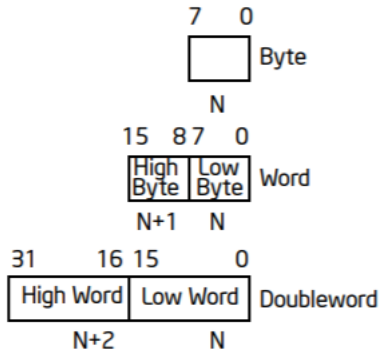

Figura 1. Tipuri fundamentale de date[1]

În cadrul limbajului de asamblare IA-32 avem date care nu își modifică valoarea pe parcursul execuției programului (*date constante* sau *constante*) și date care își modifică valoarea pe parcursul execuției programului (*date variabile* sau *variabile*).

1.1.1 Date constate

În cadrul IA-32 avem trei tipuri de constante:

- numere (naturale sau întregi) scrise în baza:
  - 2: ex. 101b, 11100b;
  - 10: ex. 23, 14, -1
  - 16: ex. 34FFh, 0FFh

Acestea pot fi stocate in memorie ca byte, word, doubleword sau quadword

- caracter, ex. 'a', 'B', 'c';

Acesta se stocheaza in memorie ca bye

- string (secvență de caractere), ex. 'aSc', 'abcd'.

Acesta se stocheaza in memorie ca secvente de bytes, words, doublewords sau quadwords.

1.1.2 Variabile

În limbajul de asamblare IA-32 se pot folosi două tipuri de variabile: predefinite și definite de utilizator. O variabilă are un nume, tip de dată (*byte*, *word*, *doubleword* sau *quadword*), o valoare curentă și o locație de memorie.

Variabile predefinite (regiştrii de uz general ai procesorului)

Regiștrii procesorului sunt locații de memorie pe procesor folosite pentru diferite calcule. Figura 2 prezintă regiștrii generali pentru programare (utilizați în cadrul cursului de ASC) pentru arhitectura IA-32, aceștia sunt:

1. regiștrii de uz general (de tipul dublucuvânt)
   a. **EAX, EBX, ECX, EDX – aceștia pot fi utilizați ca și tip de data dublucuvânt, cuvânt și octet (de exemplu, cuvântul cel mai puțin semnificativ din EAX se poate referi cu AX, octetul superior din AX se poate referi cu AH iar octetul inferior din AX se poate referi cu AL), prezentați în figura 3;**

      Fiecare dintre regiştrii EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP au capacitatea de 32 biţi. Fiecare dintre ei poate fi privit în acelaşi timp ca fiind format prin concatenarea (alipirea) a doi (sub)regiştri de câte 16 biți. Subregistrul superior, care conţine cei mai semnificativi 16 biţi ai registrului de 32 biţi din care face parte, nu are denumire și nu este disponibil separat.

[1] Manual Intel, vol 1 https://software.intel.com/sites/default/files/managed/a4/60/253665-sdm-vol-1.pdf

Subregistrul inferior poate însă fi accesat individual, având astfel regiștrii de 16 biți AX, BX, CX, DX, SP, BP, DI, SI, IP. Dintre aceștia, regiștrii AX, BX, CX, si DX sunt fiecare la rândul lor, formați din câte doi alți subregiștri a câte 8 biți. Există astfel regiștrii AH, BH, CH, DH, conținând cei 8 biți superiori (partea HIGH a regiștrilor AX, BX, CX și DX), respectiv AL, BL, CL, DL, conținând 2 tipuricei 8 biți inferiori (partea LOW).

Registrul EAX este registrul acumulator. El este folosit de către majoritatea instrucţiunilor ca unul dintre operanzi.
Registrul EBX - registru general
Registrul ECX - registru de numărare (registru contor) pt instr care au nevoie de indicaţii numerice.
Registrul EDX - registru de date. Împreună cu EAX se foloseşte în calculele ale căror rezultate depăşesc un dublucuvânt (32 biţi).

b. **ESP, EBP, ESI, EDI – aceștia pot fi utilizați ca și date de tip dublucuvânt și cuvânt (de exemplu, cuvântul cel mai puțin semnificativ din dublucuvântul ESP se referă cu SP), prezentați în figura 3.**

Regiştrii ESP şi EBP sunt regiştri destinaţi lucrului cu stiva. O stivă se defineşte ca fiind o zonă de memorie în care se pot depune succesiv valori, extragerea lor ulterioară făcându-se în ordinea inversă depunerii.
Registrul ESP (Stack Pointer) punctează spre elementul ultim introdus în stivă (elementul din vârful stivei).
Registrul EBP (Base pointer) punctează spre primul element introdus în stivă (indică baza stivei).
Regiştrii EDI şi ESI sunt regiştrii de index utilizaţi de obicei pentru accesarea elementelor din şiruri de octeţi sau de cuvinte. Denumirile lor (Destination Index şi Source Index) precum şi rolurile lor vor fi clarificate în cap. 4.

2. regiștrii segment (de tipul cuvânt): CS, DS, ES, SS, FS, GS – aceștia nu sunt folosiți în program. Arhitectura x86 permite folosirea a patru tipuri de segmente cu roluri diferite:
   a. segment de cod, care conţine instrucţiuni maşină;
   b. segment de date, care conţine date asupra cărora se acţionează în conformitate cu instrucţiunile;
   c. segment de stivă;
   d. segment suplimentar de date (extrasegment).

   Fiecare program este compus din unul sau mai multe segmente, de unul sau mai multe dintre tipurile de mai sus. În fiecare moment al execuţiei este activ cel mult câte un segment din fiecare tip. Regiştrii CS (Code Segment), DS (Data Segment), SS (Stack Segment), ES (Extra Segment) din BIU reţin valorile selectorilor segmentelor active, corespunzător fiecărui tip. Deci regiştrii CS, DS, SS şi ES determină adresele de început şi dimensiunile segmentelor active: de cod, de date, de stivă şi suplimentar. Regiştrii FS şi GS pot reţine selectori indicând către segmente suplimentare, fără însă a avea roluri predeterminate. Datorită utilizării lor, CS, DS, SS, ES, FS şi GS poartă denumirea de regiştri de segment (sau regiştri selectori). Registrul EIP conţine offsetul instrucţiunii curente în cadrul segmentului de cod curent, el fiind manipulat exclusiv de către BIU.

3. alți regiștrii (de tip dublucuvânt): EIP și Eflags.

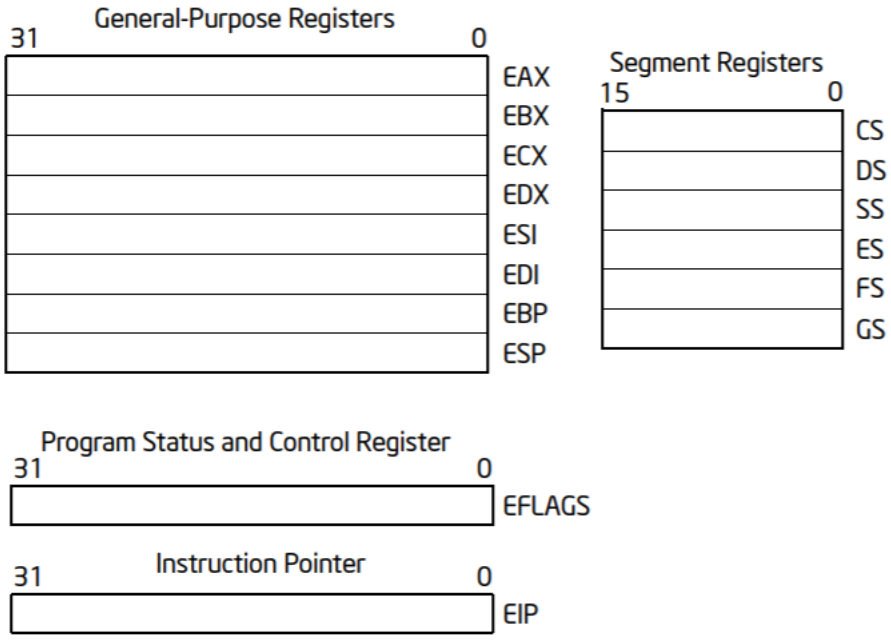

Figura 2. Regiștrii generali pentru programare[2]

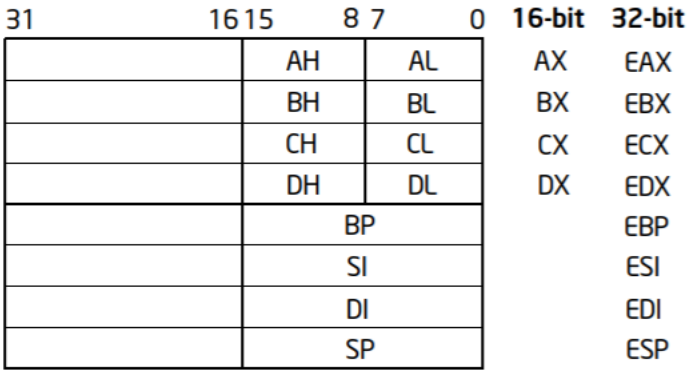

Figura 3. Regiștrii de uz general[2]

Variabile definite de utilizator

Pentru variabilele definite de utilizator, programatorul trebuie sa definească numele (opțional), tipul de data și o valoare inițiala. Exemple:

```
a   db  23      ; definește variabila cu numele "a", tipul de data octet (db - define byte) și
                ;valoarea inițiala 23
b   dw  23h     ; definește variabila cu numele "b", tipul de data cuvânt (db - define word) și
                ;valoarea inițiala 23h
c   dd  -1      ; definește variabila cu numele "c", tipul de data dublucuvânt (db - define
                ;doubleword) și valoarea inițiala -1
```

Declararea variabilelor fara valoare initiala

```
a RESB 1 ; se rezerva un octet
```

[2] Manual Intel, vol 1 https://software.intel.com/sites/default/files/managed/a4/60/253665-sdm-vol-1.pdf

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

Constante (literali) :

```
Zece EQU 10
```

1.1.3 Instrucțiuni

O instrucţiune maşină x86 reprezintă o secvenţă de 1 până la 15 octeţi, care prin valorile lor specifică o operaţie de executat, operanzii asupra cărora va fi aplicată, precum şi modificatori suplimentari care controlează modul în care aceasta va fi executată.

O instrucţiune maşină x86 are maximum doi operanzi. Pentru cele mai multe dintre instrucţiuni, cei doi operanzi poartă numele de sursă, respectiv destinaţie. Dintre cei doi operanzi, maximum unul se poate afla în memoria RAM. Celălalt se află fie într-un registru al EU, fie este o constantă întreagă. Astfel, o instrucţiune are forma:

```
numeinstrucţiune  destinaţie, sursă
```

MOV – asignare
Sintaxă: mov dest, source
(unde *dest* si *source* sunt regiștrii, variabile sau constante, ambii operanzi trebuie să fie de același tip - octet, cuvânt sau dublucuvânt, dest nu poate fi o constantă)
Efect: dest := source
Exemple:

```
mov ax, 2
mov [a], eax
```

ADD – adunare
Sintaxă: add dest, source
(unde *dest* si *source* sunt regiștrii, variabile sau constante, ambii operanzi trebuie să fie de același tip - octet, cuvânt sau dublucuvânt, dest nu poate fi o constantă)
Efect: dest := dest + source
Exemple:

```
add bx, cx
add [a], 101b
```

SUB – scădere
Sintaxă: sub dest, source
(unde *dest* si *source* sunt regiștrii, variabile sau constante, ambii operanzi trebuie să fie de același tip - octet, cuvânt sau dublucuvânt, dest nu poate fi o constantă)
Efect: dest := dest - source
Exemple:

```
sub ax, 2
sub [a], eax
```

Exercitii:

1. 1+2
2. 1-2
3. a+b, a, b – byte

4. a-b , a, b – byte
5. a+b, a, b – word
6. a-b , a, b – word
7. a+b, a, b – dw
8. a-b , a, b – dw
9. (a+b) – (c+10), a,b,c – byte
10. (a+b) – (c+10), a,b,c – word
11. (a+b) – (c+10), a,b,c – dw

1.2 Primul program în limbaj de asamblare

```
; Comentariile sunt precedate de simbolul ';'.
; Aceasta linie este un comentariu (este ignorata de asamblor)
; Acest program calculeaza expresia: x:= a + b – c = 3 + 4 – 2 = 5.
bits 32
; se declara punctul de intrare in program
; (o eticheta catre prima instructiune din program)
global start

; se declara functiile exterioare necesare programului
extern exit                 ; indica nasm ca functia exit exista
                            ; functia exit inchide procesul
import exit msvcrt.dll      ; exit este definita in msvcrt.dll

segment data use32 class=data
; segmentul de date, datele sunt declarate aici
; (variabile utilizate de program)
        a dw 3
        b dw 4
        c dw 2
        x dw 0

segment code use32 class=code
; segmentul care contine codul
start:
        mov ax, [a]     ; ax := a = 3
        add ax, [b]     ; ax := ax + b = 3+4 = 7
        sub ax, [c]     ; ax := ax – c = 7 – 2 = 5
        mov [x], ax     ; x := ax = 5
        ; exit(0)
        push  dword 0 ; pune pe stiva parametrul pentru exit
        call  [exit]    ; apelul functiei exit
```

1.3 Conversia numerelor între bazele de numerație 2, 10, 16

Un număr este convertit dintr-o baza de numerație *sursă* într-o bază de numerație *destinație*.

Conversia unui număr într-o bază de numerație folosind împărțiri succesive

- Algoritmul este folositor atunci când se vrea conversia unui număr din baza 10 în altă bază (calculele se fac în baza de numerație sursă, baza 10);
- Numărul inițial este împărțit succesiv la baza destinație până când restul împărțirii este zero (numărul inițial este împărțit la baza destinație, câtul obținut este împărțit la baza destinație și tot așa până când câtul este zero).

Exemple: $(23)_{10} = (10111)_2, (28)_{10} = (1C)_{16}$

Conversia unui număr într-o bază de numerație prin înmulțiri succesive

- Algoritmul este folositor când se vrea conversia unui număr dintr-o bază de numerație diferită de baza 10 în baza 10;
- Fie un număr reprezentat în baza *s* în următorul mod $a_n a_{n-1} \dots a_1 a_0$, reprezentarea acestui număr în baza de numerație *d* se obține în următorul mod:
$$a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s^1 + a_0 s^0$$
unde calculele sunt efectuate în baza *d*.

Exemple:
$(10111)_2 = (23)_{10}$, reprezentarea numărului în baza 10 se obține astfel
$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 23,$$
$(1C)_{16} = (28)_{10}$, reprezentarea numărului în baza 10 se obține astfel
$$1 \cdot 16^1 + 12 \cdot 16^0 = 28.$$

Deoarece o cifră în baza 16 se scrie folosind patru cifre în baza doi următorul tabel este folositor pentru a converti rapid numere între bazele 2, 10 și 16.

Baza 2	**Baza 10**	**Baza 16**
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Binary digit = **Bit** (cea mai mica cantitate de informație)

1.4 Reprezentarea numerelor întregi în memoria calculatorului

Fie următoarea instrucțiune

```
mov ax, 7
```

care instruiește microprocesorul de a seta valoarea registrului ax la 7. Apare întrebarea: cum reprezintă microprocesorul numere întregi în memorie și regiștrii?

În arhitectura IA-32 microprocesorul reprezintă un număr întreg pe 1, 2, 4 sau 8 octeți. Un număr poate fi reprezentat în două moduri: *cu semn* și *fără semn*. CPU alege una din aceste reprezentări în funcție de instrucțiunea executată.

Reprezentarea fără semn a numerelor

- În reprezentarea fără semn a numerelor putem reprezenta doar numere pozitive;
- reprezentarea unui număr pozitiv presupune reprezentarea numărului în baza doi;
- ex. pe 8 biți în reprezentare fără semn $(17)_{10} = (00010001)_2$, $(39)_{10} = (00100111)_2$.

Reprezentarea numerelor cu semn

- în reprezentarea cu semn putem reprezenta numere pozitive și negative;
- reprezentarea unui număr cu semn pozitiv este identică cu reprezentarea unui număr fară semn;
- reprezentarea numerelor negative este reprezentarea numărului respectiv în *cod complementar față de 2* a acelui număr. **Pentru a obține codul complementar față de 2 a unui număr negativ se scade modulul numărului reprezentat în baza 2 din 1 urmat de atâtea cifre de 0 câte sunt necesare pentru a reprezenta valoarea absolută a numărului;**
- bitul cel mai semnificativ din reprezentarea în baza doi a numărului reprezintă bitul de semn (1 = număr negativ și 0 = număr pozitiv);
- ex. reprezentarea cu semn a numărului 17 pe 8 biți în baza doi este 00010001, numărul este pozitiv deoarece bitul cel mai semnificativ (0) are valoarea zero;
- ex. reprezentarea cu semn a numărului -17 pe 8 biți în baza doi este 11101111, interpretat cu semn numărul este negativ deoarece bitul cel mai semnificativ (1) are valoarea unu

```
1 0000 0000 –
     1 0001
1110 1111
```

- ex. reprezentarea cu semn a numărului -39 pe 16 biți este 1111111111011001

```
1 0000 0000 0000 0000–
            10 0111
1111 1111 1101 1001
```

Reprezentarea numerelor

Numerele intregi fara semn sunt valori care pot fi continute intr-un byte [0,255], word [0, 65535], doubleword [0, 2^32-1] sau quadword [0, 2^64-1].

Numerele intregi cu semn sunt valori carepot fi reprezentate intr-un byte [-128, +127], word [-32768, +32767], doubleword [-2^31, +2^31-1] sau quadword [-2^63,+2^63-1].

Bitul de semn este localizat in bitul 7 intr-un byte, bitul 15 intr-un word, bitul 1 intr-un doubleword si bitul 63 intr-un quadword.

Table 4-1. Signed Integer Encodings

Class		Two's Complement Encoding	
		Sign	
Positive	Largest	0	11..11
		.	.
		.	.
	Smallest	0	00..01
Zero		0	00..00
Negative	Smallest	1	11..11
		.	.
		.	.
	Largest	1	00..00
Integer indefinite		1	00..00
		Signed Byte Integer: Signed Word Integer: Signed Doubleword Integer: Signed Quadword Integer:	← 7 bits → ← 15 bits → ← 31 bits → ← 63 bits →

Interpretarea numerelor

Fie valoarea 11101111 din registrul AL, se execute următoarea instrucțiune:

```
mul bl
```

Instrucțiunea *mul* multiplică valoarea din registrul AL cu valoarea din registrul BL și stochează rezultatul în registrul AX. Când CPU execută această instrucțiune trebuie să răspundă la următoarea întrebare: *ce număr întreg reprezintă secvența de biți din registrul AL (11101111) în baza 10?* CPU trebuie să *interpreteze* secvența de biți din AL într-un număr pentru a putea calcula rezultatul operației de înmulțire.

Numărul poate fi interpretat în două moduri: cu/fără semn. În exemplul de mai sus secvența de opt biți din registrul AL, 11101111, poate fi interpretată ca un număr:

- fără semn – în acest caz se știe că în reprezentarea fără semn sunt doar numere pozitive, secvența de biți reprezintă un număr pozitiv (bitul cel mai semnificativ contribuie la valoarea numărului) reprezentat în baza doi, numărul în baza 10 este

$$1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 128 + 64 + 32 + 0 + 8 + 4 + 2 + 1 = 239$$

- cu semn – în acest caz bitul cel mai semnificativ este bit de semn, acesta are valoarea 1 ceea ce înseamnă că numărul din AL este negativ (registrul AL conține codul complementar față de doi a numărului negativ); pentru a obține *codul direct* (reprezentarea în baza 2 a modulului numărului) se folosește următoarea regulă: *se parcurg toți biții ai codului complementar de la dreapta la stânga și se păstrează neschimbați până se întâlnește un bit cu valoarea 1, inclusiv acesta, restul biților se complementează*. Pentru exemplul de mai sus, 11101111, codul direct este $(00010001)_2 = (17)_{10}$. Secvența de opt biți 11101111 din memorie interpretată cu semn este numărul -17.

1.5 Instrucțiuni cu și fără semn

Ținând cont de reprezentarea numerelor cu și fără semn, pe arhitectura IA-32 există trei tipuri de instrucțiuni:

1. instrucțiuni care nu țin cont de reprezentarea cu/fără semn a numerelor: *mov, add, sub*;
2. instrucțiuni care interpretează operanzii ca fiind numere fără semn: *div, mul*;
3. instrucțiuni care interpretează operanzii ca fiind numere cu semn: *idiv, imul, cbw, cwd, cwde*.

Este important ca programatorul să fie consistent atunci când programează în limbajul IA-32: trebuie să considere toate valorile numerice ca fiind pozitive (în acest caz să folosească doar instrucțiuni din clasa 1 și 2) sau toate valorile numerice sunt numere cu semn (în acest caz să folosească doar instrucțiuni din clasa 1 si 3).