

#### 2.6.4. Registrii de adresă și calculul de adresă

*Adresa unei locații* – nr. de **octeți** consecutivi dintre începutul memoriei RAM și începutul locației respective.

O succesiune continuă de locații de memorie, menite să deservească scopuri similare în timpul execuției unui program, formează un *segment*. În consecință, un segment reprezintă o diviziune logică a memoriei unui program, caracterizată prin *adresa de bază* (început), *limita* (dimensiune) și *tipul* acesteia. Atât adresa de bază cât și dimensiunea unui segment au valori reprezentate pe 32 biți.

In the family of 8086-based processors, the term **segment** has two meanings:

1. A block of memory of discrete size, called a *physical segment*. The number of bytes in a physical memory segment is
  - o (a) 64K for 16-bit processors
  - o (b) 4 gigabytes for 32-bit processors.
2. A variable-sized block of memory, called a *logical segment* occupied by a program's code or data.

Vom numi *offset* sau *deplasament* adresa unei locații față de începutul unui segment, sau, cu alte cuvinte, numărul de octeți aflați între începutul segmentului și locația în cauză. Un offset se consideră valid dacă și numai dacă valoarea sa numerică, pe 32 biți, nu depășește limita segmentului la care se raportează.

Vom numi *specificare de adresă* sau *adresă logică* o pereche formată dintr-un *selector de segment* și un offset. Un *selector de segment* este o valoare numerică de 16 biți care identifică (indică/selectează) în mod unic segmentul accesat și caracteristicile acestuia. În scriere hexazecimală o adresă se exprimă sub forma:

**S<sub>3</sub>S<sub>2</sub>S<sub>1</sub>S<sub>0</sub> : 0706050403020100**

În acest caz, selectorul  $s_3s_2s_1s_0$  indică accesarea unui segment a cărui adresă de bază este de forma  $b_7b_6b_5b_4b_3b_2b_1b_0$  și având o limită  $l_7l_6l_5l_4l_3l_2l_1l_0$ . Baza și limita sunt determinate de către procesor în urma aplicării mecanismului de segmentare.

Pentru a fi permis accesul către locația specificată, este necesar să fie îndeplinită condiția:

$$0706050403020100 \leq b_7b_6b_5b_4b_3b_2b_1b_0.$$

Determinarea *adresei de segmentare* din specificarea de adresă se face conform formulei:

$$a_7a_6a_5a_4a_3a_2a_1a_0 := b_7b_6b_5b_4b_3b_2b_1b_0 + 0706050403020100$$

unde  $a_7a_6a_5a_4a_3a_2a_1a_0$  este adresa calculată (scrisă în hexazecimal). Adresa rezultată din calculul de mai sus, poartă numele de *adresă liniară*.

O specificare de adresă mai poartă și numele de adresă FAR (îndepărtată). Atunci când o adresă se precizează doar prin offset, spunem ca este o adresă NEAR (apropiată).

Un exemplu concret de specificare de adresă este:

**8:1000h**

Pentru a calcula adresa liniară ce-i corespunde acestei specificări, procesorul va proceda după cum urmează:

1. Verifică dacă segmentul ce corespunde valorii de selector 8 a fost definit de către sistemul de operare și se blochează accesul dacă nu a fost definit un astfel de segment;
2. Extrage adresa de bază (B) și limita acestui segment (L), de exemplu, ca rezultat am putea avea  $B = 2000h$  și  $L = 4000h$ ;
3. Verifică dacă offsetul depășește limita segmentului:  $1000h > 4000h$  ? în caz de depășire accesul ar fi fost blocat;
4. Adună offsetul cu B, obținând în cazul nostru adresa liniară  $3000h$  ( $1000h + 2000h$ ). Acest calcul este efectuat de către componenta **ADR** din **BIU**.

Acest mecanism de adresare poartă numele de *segmentare*, vorbind astfel despre *modelul de adresare segmentată*.

În cazul în care segmentele încep la adresa 0 și au dimensiunea maximă posibilă (4GiB), orice offset este automat valid și segmentarea nu contribuie efectiv în calculul adreselor. Astfel, având  $b_7b_6b_5b_4b_3b_2b_1b_0 = 00000000$ , calculul de adresă pentru adresa logică  $s_3s_2s_1s_0 : 0706050403020100$  va rezultă în adresa liniară:

$$\mathbf{a_7a_6a_5a_4a_3a_2a_1a_0 := 00000000 + 0706050403020100}$$

$$\mathbf{a_7a_6a_5a_4a_3a_2a_1a_0 := 0706050403020100}$$

⇒

Acest mod particular de utilizare a segmentării, folosit de către majoritatea sistemelor de operare moderne poartă numele de *model de memorie flat*.

Procesoarele x86 suportă și un mecanism de control al accesului la memorie numit *paginare*, independent de adresarea segmentată. Paginarea implică împărțirea memoriei *virtuale* în *pagini*, care sunt asociate (translatate) memoriei fizice disponibile.

Configurarea și controlul mecanismelor de segmentare și paginare sunt sarcina sistemului de operare. Dintre cele doua, doar segmentarea intervine în specificarea de adrese, paginarea fiind complet transparentă din perspectiva programelor de utilizator.

Atât calculul de adrese cât și folosirea mecanismelor de segmentare și paginare sunt influențate de *modul de execuție* al procesorului, procesoarele x86 suportând următoarele moduri de execuție mai importante:

- *mod real*, pe 16 biți (folosind cuvânt de memorie de 16 biți și având memoria limitată la 1MiB);

- *mod protejat pe 16 sau 32 biți*, caracterizat prin folosirea paginării și segmentării;
- *mod virtual 8086*, permite rulare programelor de tip mod real alături de cele de mod protejat;
- *long mode*, pe 64 sau 32 biți, unde paginarea este obligatorie în timp ce segmentarea este dezactivată.

În cadrul cursului ne vom concentra asupra arhitecturii și comportamentului procesoarelor x86 în modul protejat pe 32 de biți.

Arhitectura x86 permite folosirea a patru tipuri de segmente cu roluri diferite:

- *segment de cod*, care conține instrucțiuni mașină;
- *segment de date*, care conține date asupra cărora se acționează în conformitate cu instrucțiunile;
- *segment de stivă*;
- *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 execuției este activ cel mult câte un segment din fiecare tip. Regiștrii **CS** (*Code Segment*), **DS** (*Data Segment*), **SS** (*Stack Segment*), **ES** (*Extra Segment*) din **BIU** rețin valorile selectorilor segmentelor active, corespunzător fiecărui tip. Deci regiștrii CS, DS, SS și ES determină adresele de început și dimensiunile segmentelor active: de cod, de date, de stivă și suplimentar. Regiștrii **FS** și **GS** pot reține selectori indicând către segmente suplimentare, fără însă a avea roluri predeterminate. Datorită utilizării lor, CS, DS, SS, ES, FS și GS poartă denumirea de *regiștri de segment* (sau *regiștri selectori*). Registrul **EIP** (care oferă și posibilitatea accesării cuvântului său inferior prin subregistrul **IP**) conține offsetul instrucțiunii curente în cadrul segmentului de cod curent, el fiind manipulat exclusiv de către **BIU**.

Cum noțiunile asociate adresării sunt fundamentale înțelegerii funcționării procesoarelor x86 și programării în limbaj de asamblare, este foarte importantă cunoașterea acestora. Pentru aceasta, le recapitulăm pe scurt în vederea clarificării:

Noțiune	Reprezentare	Descriere
Specificare de adresă, adresă logică, adresă FAR	Selector <sub>16</sub> :offset <sub>32</sub>	Definește complet atât segmentul cât și deplasamentul în cadrul acestuia
Selector	16 biți	Identifică unul dintre segmentele disponibile. Ca valoare numerică acesta codifică poziția descriptorului de segment selectat în cadrul unei tabele de descriptori.
Offset, adresă NEAR	Offset <sub>32</sub>	Definește doar componenta de offset (considerând segmentul cunoscut ori folosirea modelului de memorie flat)
Adresă liniară (adresă de segmentare)	32 biți	Inceput segment + offset, reprezintă rezultatul calculului de segmentare
Adresă fizică efectivă	Cel puțin 32 biți	Rezultatul final al segmentării plus, eventual, paginării. Adresa finală obținută de către BIU, indicând în memoria fizică (hardware)

### 2.6.5. Reprezentarea instrucțiunilor mașină

O instrucțiune mașină x86 reprezintă o secvență de 1 până la 15 octeți, care prin valorile lor specifică o operație de executat, operanzii asupra cărora va fi aplicată, precum și modificatori suplimentari care controlează modul în care aceasta va fi executată.

O instrucțiune mașină x86 are maximum doi operanzi. Pentru cele mai multe dintre instrucțiuni, cei doi operanzi poartă numele de *sursă*, respectiv *destinație*. 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 instrucțiune are forma:

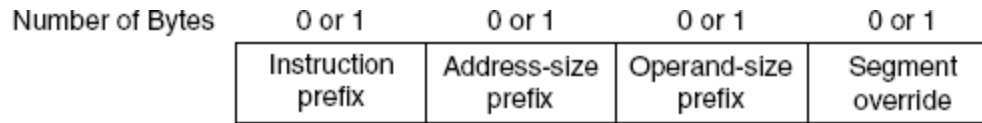
*numeinstrucțiune destinație, sursă*

Formatul intern al unei instrucțiuni este variabil, el putând ocupa între 1 și 15 octeți, având următoarea formă generală de reprezentare (*Instructions byte-codes from OllyDbg*):

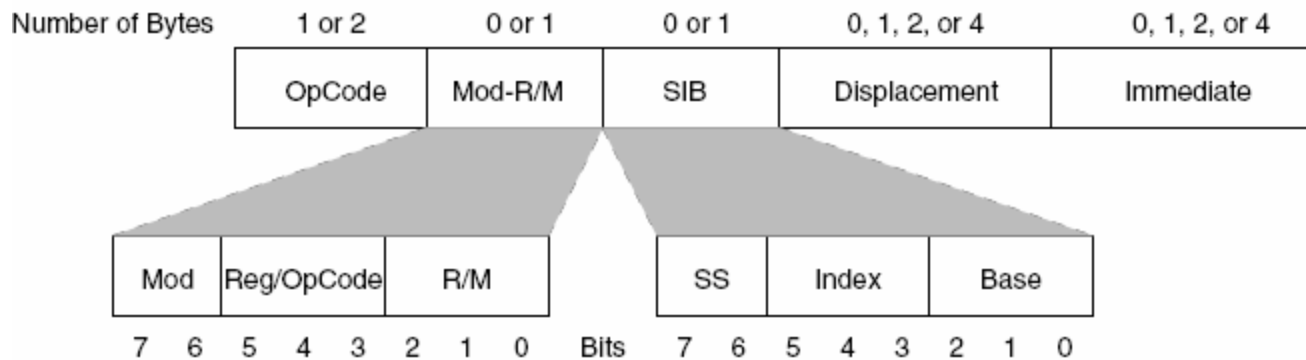
$[prefixe] + cod + [ModR/M] + [SIB] + [deplasament] + [imediat]$

*Prefixele* controlează modul în care o instrucțiune se execută. Acestea sunt opționale (0 până la maximum 4) și ocupă câte un octet fiecare. De exemplu, acestea pot solicita execuția repetată (în buclă) a instrucțiunii curente sau pot bloca magistrala de adrese pe parcursul execuției pentru a nu permite accesul concurent la operanzi și rezultate.

Operația care se va efectua este identificată prin intermediul a 1 sau 2 octeți de *cod* (opcode), aceștia fiind singurii octeți obligatoriu prezenți, indiferent de instrucțiune.



(a) Optional instruction prefixes



(b) General instruction format

Although the diagram seems to imply that instructions can be up to 16 bytes long, in actuality the x86 will not allow instructions greater than 15 bytes in length.

Octetul *ModR/M* (mod registru/memorie) specifică pentru unele dintre instrucțiuni natura și locul operanzilor (registru, memorie, constantă întreagă etc.). Acesta permite specificarea fie a unui registru, fie a unei locații de memorie a cărei adresă este exprimată prin intermediul unui offset.

Pentru cazuri mai complexe de adresare decât cele codificabile direct prin ModR/M, combinarea acestuia cu octetul SIB permite următoarea formulă generală de definire a unui offset:

$$[\text{bază}] + [\text{index} \times \text{scală}] + [\text{constantă}]$$

unde pentru bază și index vor fi folosite valorile a doi regiștri iar scală este 1, 2, 4 sau 8. Regiștrii permiși ca bază sau / și index sunt: EAX, EBX, ECX, EDX, EBP, ESI, EDI. Registrul ESP este disponibil ca bază însă nu poate fi folosit cu rol de index.

Majoritatea instrucțiunilor folosesc pentru reprezentare fie numai campul de cod, fie cod urmat de ModR/M.

*Deplasament* (displacement) apare în cazul unor forme de adresare particulare (operanzi din memorie) și urmează direct după ModR/M sau SIB, când SIB este prezent. Acest câmp poate fi codificat fie pe octet fie pe cuvânt dublu (32 biți).

Ca și consecință a imposibilității prezenței mai multor câmpuri de ModR/M, SIB și deplasament într-o instrucțiune, arhitectura nu permite codificarea a două adrese de memorie în aceeași instrucțiune.

*Valoare imediată* oferă posibilitatea definirii unui operand ca fiind o constantă numerică pe 1, 2 sau 4 octeți. Când este prezent, acest câmp apare întotdeauna la sfârșitul instrucțiunii.

#### **2.6.6. Adrese FAR și NEAR**

Pentru a adresa o locație din memoria RAM sunt necesare două valori: una care să indice segmentul, alta care să indice offsetul în cadrul segmentului. Pentru a simplifica referirea la memorie, microprocesorul derivă, în lipsa unei alte specificări, adresa segmentului din unul dintre regiștrii de segment CS, DS, SS sau ES. Alegerea implicită a unui registru de segment se face după niște reguli proprii instrucțiunii folosite.

Prin definiție, o adresă în care se specifică doar offsetul, urmând ca segmentul să fie preluat implicit dintr-un registru de segment poartă numele de *adresă NEAR* (adresă apropiată). O adresă NEAR se află întotdeauna în interiorul unuia din cele patru segmente active.



O adresă în care programatorul indică explicit un selector de segment poartă numele de *adresă FAR* (adresă îndepărtată). O adresă FAR este deci o SPECIFICARE COMPLETA DE ADRESA și ea se poate exprima în trei moduri:

- $s_3s_2s_1s_0$  : specificare\_offset unde  $s_3s_2s_1s_0$  este o constantă;
- registru\_segment : specificare\_offset, registru segment fiind CS, DS, SS, ES, FS sau GS;
- FAR [variabilă], unde variabilă este de tip QWORD și conține cei 6 octeți constituind adresa FAR.

Formatul intern al unei adrese FAR este: la adresa mai mică se află offsetul, iar la adresa mai mare cu 4 (cuvântul care urmează după dublucuvântul curent) se află cuvântul ce conține selectorul care indică segmentul.

Reprezentarea adreselor respectă principiul reprezentării little-endian expus în capitolul 1, paragraf 1.3.2.3: partea cea mai puțin semnificativă are adresa cea mai mică, iar partea cea mai semnificativă are adresa cea mai mare.

### **2.6.7. Calculul offsetului unui operand. Moduri de adresare**

În cadrul unei instrucțiuni există 3 moduri de a specifica un operand pe care aceasta îl solicită:

- *modul registru*, dacă pe post de operand se află un registru al mașinii;
- *modul imediat*, atunci când în instrucțiune se află chiar valoarea operandului (nu adresa lui și nici un registru în care să fie conținut);
- *modul adresare la memorie*, dacă operandul se află efectiv undeva în memorie. În acest caz, adresa offsetului lui se calculează după următoarea formulă:

$$adresa\_offset = [ bază ] + [ index \times scală ] + [ constanta ]$$

Deci *adresa\_offset* se obține din următoarele (maxim) patru elemente:

- conținutul unuia dintre regiștrii EAX, EBX, ECX, EDX, EBP, ESI, EDI sau ESP ca bază;
- conținutul unuia dintre regiștrii EAX, EBX, ECX, EDX, EBP, ESI sau EDI drept index;
- factor numeric (scală) pentru a înmulți valoarea registrului index cu 1, 2, 4 sau 8
- valoarea unei constante numerice, pe octet sau dublucuvânt.

De aici rezultă următoarele moduri de adresare la memorie:

- *directă*, atunci când apare numai *constanta*;
- *bazată*, dacă în calcul apare unul dintre regiștrii bază;
- *scalat-indexată*, dacă în calcul apare unul dintre regiștrii index;

Cele trei moduri de adresare a memoriei pot fi combinate. De exemplu, poate să apară adresare directă bazată, adresare bazată și scalat-indexată etc

La instrucțiunile de salt mai apare și un alt tip de adresare numit adresare *relativă*.

*Adresa relativă* indică poziția următoarei instrucțiuni de executat, în raport cu poziția curentă. Poziția este indicată prin numărul de octeți de cod peste care se va sări. Arhitectura x86 permite atât adrese relative scurte (SHORT Address), reprezentate pe octet și având valori între -128 și 127, cât și adrese relative apropiate (NEAR Address), pe dublucuvânt cu valori între -2147483648 și 2147483647.