2.6.4. Regiştrii de adresă şi calculul de adresă

Adresa unei locaţii – nr. de **octeţi** consecutivi dintre începutul memoriei RAM şi începutul locaţiei respective.

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

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
   - (a) 64K for 16-bit processors
   - (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 locaţii faţă de începutul unui segment, sau, cu alte cuvinte, numărul de octeţi aflaţi între începutul segmentului şi locaţia în cauză. Un offset se consideră valid dacă şi numai dacă valoarea sa numerică, pe 32 biţi, nu depăşeşte limita (dimensiunea) segmentului la care se raportează.

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

$$s_3s_2s_1s_0 : o_7o_6o_5o_4o_3o_2o_1o_0$$

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

Pentru a fi permis accesul către locaţia specificată, este necesar să fie îndeplinită condiţia:

$$o_7o_6o_5o_4o_3o_2o_1o_0 \leq l_7l_6l_5l_4l_3l_2l_1l_0.$$

Determinarea *adresei de segmentare* din specificarea de adresă se face printr-un *calcul de adresă* cf. formulei:

$$a_7a_6a_5a_4a_3a_2a_1a_0 := b_7b_6b_5b_4b_3b_2b_1b_0 + o_7o_6o_5o_4o_3o_2o_1o_0$$

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 (sau adresă de segmentare)*.

O specificare de adresă mai poartă şi 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 şi se blochează accesul dacă nu a fost definit un astfel de segment;
2. Extrage adresa de bază (B) şi limita acestui segment (L), de exemplu, ca rezultat am putea avea B = 2000h şi L = 4000h; (este o operaţie la ale cărei detalii NU avem acces, ea derulându-se exclusiv între procesor şi SO)
3. Verifică dacă offsetul depăşeşte limita segmentului: 1000h > 4000h ? în caz de depăşire accesul ar fi fost blocat (*memory violation error*);
4. Adună offsetul cu B, obţinând 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 şi au dimensiunea maximă posibilă (4GiB), orice offset este automat valid şi 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 $s_3s_2s_1s_0 : o_7o_6o_5o_4o_3o_2o_1o_0$ va rezultă în adresa liniară:

$$a_7a_6a_5a_4a_3a_2a_1a_0 := 00000000 + o_7o_6o_5o_4o_3o_2o_1o_0$$

$$a_7a_6a_5a_4a_3a_2a_1a_0 := o_7o_6o_5o_4o_3o_2o_1o_0$$

⇨

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 şi un mecanism de control al accesului la memorie numit *paginare*, independent de adresarea segmentată. Paginarea implică împărţirea memoriei *virtuale* în *pagini,* care sunt asociate (translatate) memoriei fizice disponibile (1 page = 4 KB = 2^12 bytes = 4096 bytes).

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

Atât calculul de adrese cât şi folosirea mecanismelor de segmentare şi paginare sunt influenţate de *modul de execuţie* al procesorului, procesoarele x86 suportând următoarele moduri de execuţie mai importante:

- *mod real*, pe 16 biţi (folosind cuvânt de memorie de 16 biţi şi având memoria limitată la 1MB);
- ***mod protejat pe 16 sau 32 biţi*, caracterizat prin folosirea paginării şi segmentării**;
- *mod virtual 8086*, permite rulare programelor de tip mod real alături de cele de mod protejat;
- *long mode*, pe 64 sau 32 biţi, unde paginarea este obligatorie in timp ce segmentarea este dezactivată.

În cadrul cursului nostru ne vom concentra asupra arhitecturii şi comportamentului procesoarelor din familia Intel x86 în modul protejat pe 32 de biţi.

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

- *segment de cod*, care conţine instrucţiuni maşină;
- *segment de date*, care conţine date asupra cărora se acţionează în conformitate cu instrucţiunile;
- *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 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** conţin 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** (care oferă şi posibilitatea accesării cuvântului său inferior prin subregistrul **IP**) conţine offsetul instrucţiunii curente în cadrul segmentului de cod curent, el fiind manipulat exclusiv de către **BIU**.

Cum noţiunile asociate adresării sunt fundamentale înţelegerii funcţionării procesoarelor x86 şi programării în limbaj de asamblare, este foarte importantă cunoaştera acestora. Pentru aceasta, le recapitulăm pe scurt în vederea clarificării:

Noţiune	**Reprezentare**	**Descriere**
Specificare de adresă, adresă logică,adresă FAR	$Selector_{16}:offset_{32}$	Defineşte complet atât segmentul cât şi deplasamentul în cadrul acestuia
Selector de segment	16 biţi	Identifică unul dintre segmentele disponibile. Ca valoare numerică acesta codifică poziţia descriptorului de segment selectat în cadrul unei tabele de descriptori.
Offset, adresă NEAR	$Offset_{32}$	Defineşte doar componenta de offset (considerând segmentul cunoscut ori folosirea modelului de memorie flat)
Adresă liniară (adresă de segmentare)	32 biţi	Inceput segment + offset, reprezintă rezultatul calculului de adresă
Adresă fizică efectivă	Cel puţin 32 biţi	Rezultatul final al segmentării plus, eventual, paginării. Adresa finală obţinută de către BIU, indicând în memoria fizică (hardware)