**Regiștrii de adresă și calcul de adresă**

**Adresa unei locații** = numărul de octeți conseecutivi de la începutul memoriei  
 RAM până la locația respective.

**Segmentul** = succesiune continuă de locații de memorie, menite să deservească  
 scopuri similar în timpul execuției unui program.

**Offset** = adresa unei locații față de începutul unui segment sau numărul de octeți  
 dintre începutul segmentului și locația în cauza. Se consideră valid dacă  
 nu depășește limita segmentului la care se raportează.

**Selector de segment** = valoare numerică de 16 biți care identifică în mod unic  
 segmental accesat și caracteristicile acestuia. Un selector  
 de segment este definit și furnizat de către sistemul de  
 operare.

**Adresa logică** = pereche formată dintr-un selector de segment și un offset.  
 s3s2s1s0 : o7o6o5o4o3o2o1o0

**Adresă liniară** = rezultatul calcului a7a6a5a4a3a2a1a0 = b7b6b5b4b3b2b1b0 (adresa de  
 bază) + o7o6o5o4o3o2o1 (offset), efectuat de **ADR** și **BIU**.

**Adresă FAR** = adresă logică (selector de segment indicat explicit de programator)

**Adresă NEAR** = adresă precizată doar prin offset (segmentul luat implicit dintr-un  
 registru de segment)

**Model de memorie flat** = mod particular de utilizare a segmentării, segmentele  
 încep de la adresa 0 și au dimensiunea maximă posibilă  
 de 4GB (b7b6b5b4b3b2b1b0 = 0)

**Tipuri de segmente (arhitectura x86)**:  
 - **CS (code segment)**: segment de cod (instrucțiuni mașină)  
 - **DS (data segment)**: segment de date (date asupra cărora se acționează)  
 - **SS (stack segment)**: segment de stivă  
 - **ES (extra segment)**: segment suplimentar de date  
 **Regiștrii CS, DS, SS, ES** din **BIU** conțin valorile selectorilor de segment active, **regiștrii FS și GS** pot contine selectori care indică segmente suplimentare, **registrul EIP** contine offsetul instrucțiunii curente din segmental de cod.

**Reprezentarea instrucțiunilor masina**

**Instrucțiune x86** = operație specifică de executat, are maxim 2 operanzi care în  
 majoritatea operațiilor poartă numele de **sursă** și **destinație**.  
 Maxim unul poate fi din RAM, celălalt aflându-se într-un  
 registru. Ex: **nume\_instrucțiune destinație, sursă**

**Formatul unei instrucțiuni** = ocupă între 1 și 15 octeți și este de forma: **[prefixe] +**   **cod + [Mod R/M] + [SIB] + [deplasament] + [imediat]**

**Prefixe** = controlează modul în care o instrucțiune se execută.

**Mod R/m** = mod registru / memorie, specifică pentru unele instrucțiuni natura și  
 locul operanzilor (registru sau memorie)

**SIB** **= [bază] + [index \* scală]**, bază, index regiștrii. scală = {1, 2, 4, 8}, index ≠ ESP

**Deplasament** = apare în cazul unor forme de adresare particulare

**Valoare imediată** = oferă posibilitatea definirii unui operand ca fiind o constantă  
 numerică pe 1, 2 sau 4 octeți.

**Arhitectura microprocesorului 80x86**

**EU (executive unit)** = execută instrucțiuni masina prin intermediul componentei  
 **ALU (Arithmetic and Logic Unit)**

**BIU (bus interface unit)** = pregătește instrucțiunile mașină să fie executate

\* Componentele microprocesorului lucrează în paralel.

**Regiștri generali EU**- **EAX** – **registru acumulator**, folosit în majoritatea instrucțiunilor ca fiind unul  
 dintre operatori  
 - **EBX** – **registru de bază**, base register, cu acest rol pe 16 biți  
 - **ECX** – **registru contor**, folosit pentru instrucțiuniile cu indicații numerice  
 - **EDX** – **registru de date**, împreună cu EAX pentru calculele ce depășesc dword  
 - **ESP** – **stack pointer**, punctează spre elementul din vârful stivei (ultimul introdus)  
 - **EBP** – **base pointer**, punctează spre primul element din stivă (primul introdus)  
 - **EDI** – **source index**, registru index pentru șirurile sursă  
 - **ESI** – **destination index**, registru index pentru șirurile destinați

**Stivă** = zonă de memorie în care se pot depune și extrage valori

**Flag** = indicator pe un bit. O configurație a registrului de flaguri este un rezumat  
 sintetic a execuției fiecărei instrucțiuni. Pentru x86, registrul EFLAGS are 32  
 de biți din care sunt folosiți uzual doar 9

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X 31 | X 30 | ... | **OF** 11 | **DF** 12 | **IF** 9 | **TF** 8 | **SF** 7 | **ZF** 6 | X 5 | **AF** 4 | X 3 | **PF** 2 | X 1 | **CF** 0 |

**Carry Flag (CF)** = flagul de transport, are valoarea 1 dacă în cadrul UOE s-a  
 efectuat transport în afara domeniului de reprezentare și 0 în caz  
 contrar. Semnalează depășirea în interpretarea FĂRĂ SEMN  
**Parity Flag (PF)** = valoarea lui se stabilește împreună cu numărul de biți de 1 din  
 octetul cel mai puțin semnificativ al reprezentării UOE să rezulte  
 un număr impar de cifre de 1  
**Auxiliary Flag (AF)** = indică valoarea transportului de la bitul 3 la bitul 4 al UOE.  
**Zero Flag (ZF)** = primește valoarea 1 dacă rezultatul UOE este 0, valoarea 0 altfel  
**Sign Flag (SF)** = primește valoarea 1 dacă rezultatul UOE are bitul de semn 1,  
 valoarea 0 altfel  
**Trap Flag (TF)** = flag de depanare, dacă are valoarea 1 atunci mașina se oprește  
 după fiecare instrucțiune  
**Interrupt Flag (IF)** = flag de întrerupere  
**Direction Flag (DF)** = pentru operarea șirurilor de octeți sau de cuvinte. Dacă  
 valoarea lui este 0, deplasarea se face la început spre sfârșit,  
 dacă are valoarea 1, deplasarea se face în sens contrar.  
**Overflow Flag (OF)** = flag pentru depășirea CU SEMN. Dacă rezultatul UOE în  
 interpretarea cu semn a operanzilor nu a încăput în spațiul  
 rezervat operanzilor, atunci flagul va avea valoarea 1, 0 altfel

**Instrucțiuni flaguri**  
 - **CLC** 🡪 CF = 0 **STC** 🡪 CF = 1 **CMC** 🡪 complementarea CF  
 - **CLD** 🡪 DF = 0 **STD** 🡪 DF = 1  
 - **CLI** 🡪 IF = 0 **STI** 🡪 IF = 1 (acces doar pe 16 biți)

Elemente de bază ale limbajului de asamblare

Bla blab la daca pica ftai venele