Universitatea de Vest din Timisoara, Facultatea de Matematica si Informatica

ARHITECTURA CALCULATOARELOR

Informatica, an I, 2021-2022

Dr. Mafteiu-Scai Liviu Octavian

Universitatea de Vest din Timisoara, Facultatea de Matematica si Informatica
ARHITECTURA CALCULATOARELOR, Informatica, an I, 2021

CURS 11++

Limbajul de asamblare pentru procesoare x86 partea a 3-a

Dr. Mafteiu-Scai Liviu Octavian

ARM PROCESSOR VS. INTEL PROCESSOR

INTEL - CISC (Complex Instruction Set Computing) :

- Mai multe operatii/instructiuni decat ARM
- Mai multe moduri de adresare decat ARM
- Mai putini registrii generali decat ARM

RISC (Reduced instruction set Computing) :

- Un set simplificat/redus de instructiuni (100 sau mai putine)
- Mai multi registrii de uz general decat CISC
- ARM foloseste instructiuni care pot opera doar pe registrii
- Foloseste un model de memorie de tip Load/Store memory, adica doar aceste instructiuni pot accesa memoria RAM

Familia de microprocesoare x86 :

- Produse Intel respectiv compatibile (IBM, AMD, Cyrix, ...)
- Echipeaza in general sisteme IBM-PC si compatibile
- vs. Motorola ce echipeaza Mac-PC (60xxx)
- Procesoare 8088, 8086, 80286, 80386, 80486, Pentium....samd
- 8086 este baza familiei
- Z80

Registrii:

- reprezintă locaţii de memorie speciale aflate direct pe cipul microprocesorului.

-> cel mai rapid tip de memorie.

- fiecare registru are un scop bine precizat, asigurand anumite funcţionalităţi

Categorii de regiştri:

- regiştrii de uz general,
- registrul indicatorilor de stare (flags)
- regiştrii de segment
- registrul pointer de instrucţiune.

ARHITECTURA MICROPROCESORULUI 8086

Structura microprocesorului

Microprocesorul 8086 este format din două componente principale:

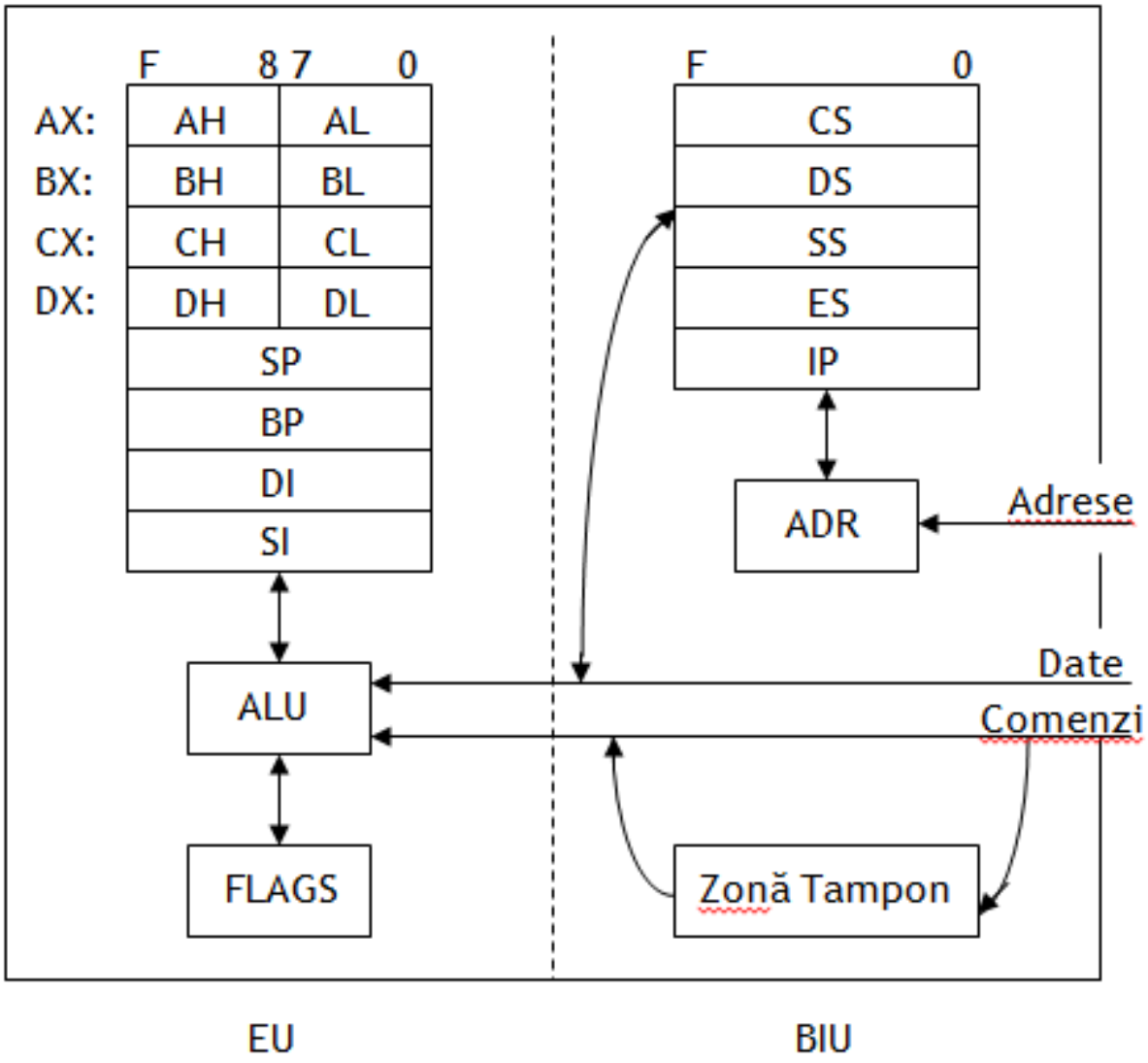
- **EU** (*Executive Unit*) - execută instr. maşină prin intermediul componentei **ALU** (*Aritmetic and Logic Unit*).
- **BIU** (*Bus Interface Unit*) - pregăteşte execuţia fiecărei instrucţiuni maşină. Citeşte o instrucţiune din memorie, o decodifică şi calculează adresa din memorie a unui eventual operand. Configuraţia rezultată este depusă într-o zonă tampon cu dimensiunea de 6 octeţi, de unde va fi preluată de **EU**.

EU si **BIU** lucrează în paralel –

- **EU** execută instrucţiunea curentă,
- **BIU** pregăteşte instrucţiunea următoare.

Cele două acţiuni sunt sincronizate - cea care termină prima aşteaptă după cealaltă.

Structura microprocesorului

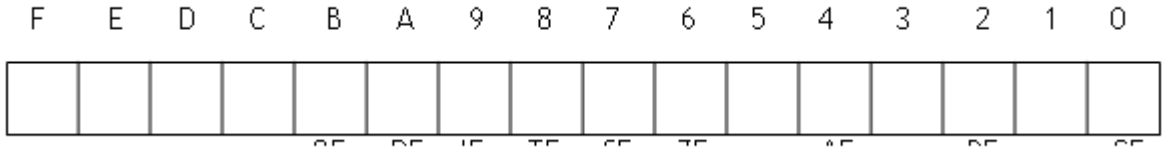


Regiştrii generali EU

- Registrul **AX** este *registrul acumulator*. El este folosit de către majoritatea instrucţiunilor ca unul dintre operanzi.
- Registrul **BX** - *registru de bază*. Folosit de obicei in adresare.
- Registrul **CX** - *registru de numărare (registru contor)* pt instr care au nevoie de indicaţii numerice.
- Registrul **DX** - *registru de date*. Împreună cu AX se foloseşte în calculele ale căror rezultate depăşesc un cuvânt.
- Fiecare dintre regiştrii AX, BX, CX, DX au capacitatea de 16 biţi. Fiecare dintre ei poate fi privit în acelaşi timp ca fiind format prin concatenarea (alipirea) a doi (sub)regiştri. Subregistrul superior conţine cei mai semnificativi 8 biţi (partea HIGH) ai registrului de 16 biţi din care face parte. Avem astfel regiştrii **AH, BH, CH, DH**. Subregistrul inferior conţine cei mai puţin semnificativi 8 biţi (partea LOW) ai registrului de 16 biţi din care face parte. Există astfel regiştrii **AL, BL, CL, DL**.
- Regiştrii **SP** şi **BP** 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 **SP** (*Stack Pointer*) punctează spre elementul ultim introdus în stivă (elementul din *vârful stivei*).
- Registrul **BP** (*Base pointer*) punctează spre primul element introdus în stivă (indică *bazei stivei*).
- Regiştrii **DI** şi **SI** sunt *regiştrii de index* utilizaţi de obicei pentru accesarea elementelor din şiruri de octeţi sau de cuvinte.

Flagurile

Un flag este un indicator reprezentat pe un bit. O configuraţie a registrului de flaguri indică un rezumat sintetic a execuţiei fiecărei instrucţiuni. Pentru 8086 registrul FLAGS are 16 biţi dintre care sunt folosiţi numai 9.



Flagurile

CF (Carry Flag) este flagul de transport. Are valoarea 1 în cazul în care în cadrul ultimei operatii efectuate (UOE) s a efectuat transport în afara domeniului de reprezentare a rezultatului si valoarea 0 in caz contrar. De exemplu, pt

1001 0011 +
0111 0011 — rezultă un transport de cifră semnificativă si
1 0000 0110 — valoarea 1 este depusă automat în CF

PF (Parity Flag) - Valoarea lui se stabileşte a.î. împreună cu numărul de biţi 1 din reprezentarea rezultatului UOE să rezulte un număr impar de cifre 1.

AF (Auxiliary Flag) indică valoarea transportului de la bitul 3 la bitul 4 al rezultatului UOE. De exemplu, în adunarea de mai sus transportul este 0.

ZF (Zero Flag) primeşte valoarea 1 dacă rezultatul UOE este egal cu zero şi valoarea 0 la rezultat diferit de zero.

SF (Sign Flag) primeşte valoarea 1 dacă rezultatul UOE este un număr strict negativ şi valoarea 0 în caz contrar.

TF (Trap Flag) este un flag de depanare. Dacă are valoarea 1, atunci maşina se opreşte după fiecare instrucţiune.

IF (Interrupt Flag) este flag de întrerupere. Asupra acestui flag vom reveni în capitolul 7.

DF (Direction Flag) - pt operare asupra şirurilor de octeţi sau de cuvinte. Dacă are valoarea 0, atunci deplasarea în şir se face de la început spre sfârşit, iar dacă are valoarea 1 este vorba de deplasări de la sfârşit spre început.

OF (Overflow Flag) este flag pentru depăşire. Dacă rezultatul ultimei instrucţiuni nu a încăput în spaţiul rezervat operanzilor, atunci acest flag va avea valoarea 1, altfel va avea valoarea 0.

Reprezentarea instrucţiunilor maşină

- O instrucţiune maşină 8086 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*

- Formatul intern al unei instrucţiuni este variabil, el putând ocupa între 1 şi 6 octeţi. Primul octet, numit *cod* identifică instrucţiunea de executat. Al doilea octet, numit *octetmod* specifică pentru unele dintre instrucţiuni natura şi locul operanzilor (registru, memorie, constantă întreagă etc.). Majoritatea instrucţiunilor folosesc pentru reprezentare fie numai octetul cod, fie octetul cod urmat de octetmod (deci 1-2 octeti). Următorii maximum patru octeţi, dacă apar, identifică fie o adresă de memorie, fie o constantă reprezentată pe mai mult de un octet.

Organizarea memoriei la Arhitectura ISA x86

- memoria este divizata în regiuni numite segmente
- adresa unei locaţii de memorie se exprima printr-o pereche de adrese:

<adresă de segment>:<adresă de offset>

  - adresa de segment indică punctul de început al unei zone de memorie
  - adresa de offset (relativa) este adresa relativă a locaţiei căutate în raport cu începutul de segment.

Avantaje adresarea pe segmente:

- adresa relativă în cadrul segmentului (adresa de offset) se exprimă pe un număr mai mic de biţi; astfel instrucţiunile care conţin adrese sunt mai scurte
- încărcarea şi reamplasarea programelor şi a datelor este mai simplă deoarece adresele relative rămân aceleaşi
- există posibilitatea protejării diferitelor zone de memorie pe bază de conţinut şi funcţie de drepturile de acces
- se pot încărca în memorie numai acele segmente ale unei aplicaţii care sunt strict necesare
- un sistem de operare de tip multi-tasking sau multi-utilizator are nevoie în mod obligatoriu de un sistem de partajare şi de protecţie a zonelor de memorie

Organizarea memoriei la Arhitectura ISA x86

Q: Ce este ISA? A: Instruction Set Architecture

La ISA x86 tehnica de segmentare folosită depinde de modul de lucru al procesorului:

- modul real
- modul protejat

La primele variante de procesoare exista numai modul real; ulterior, pentru a oferi suport pentru multitasking şi pentru a creşte spaţiul de adresare s-a introdus modul protejat.

În majoritatea cazurilor programele scrise de programatorii obişnuiţi folosesc modelul iniţial de segmentare, deşi procesorul lucrează în modul protejat. Acest lucru este posibil prin faptul că programul poate să ruleze într-un al treilea mod, modul virtual, care este o simulare a modului real pe un procesor aflat în modul protejat.

În mod uzual operaţiile care afectează funcţionarea în modul protejat sunt efectuate numai de rutine ale sistemului de operare.

Alocarea memoriei este responsabilitatea sistemului de operare: utilizatorul obişnuit are senzaţia lucrului în modul real, mod care este mult mai simplu.

Organizarea memoriei în modul real

- În modul real spaţiul maxim de adresare al memoriei este de 1Moctet. …. vezi sisteme 80286
- Memorie este împărţită în segmente de lungime fixă de 64Kocteţi.
- Adresa de început a unui segment se păstrează în unul din cele patru registre segment.
- Adresa fizică a unei locaţii de memorie se calculează ca o sumă între adresa de segment şi o adresă de offset.

Calcul adresa:

- Adresa de segment se obţine prin multiplicarea conţinutului registrului segment cu 16 (deplasarea la stânga cu 4 poziţii binare).
- Adresa de ofset se calculează pe baza modului de adresare şi eventual a adresei conţinute in codul de instrucţiune.
- Prin adunare se obţine o adresă fizică pe 20 de biţi, suficientă pentru adresarea unui spaţiu de 1 Mocteţi ($1M=2^{20}$).

Exemplu: (valorile de adrese sunt exprimate în hexazecimal)

1 2 3 4 0	Adresa de offset
5 6 7 8	Adresa de segment
1 7 9 B 8	Adresa fizica

? La ce “distanta” se afla offset-ul fata de segment? Dar fata de adresa fizica?

Organizarea memoriei în modul real

Consecintele acestui mod de calcul a adresei fizice:

- spaţiul maxim de adresare este 1Mo
- un segment trebuie să înceapă la o adresa multiplu de 16
- un segment are maxim 64ko
- segmentele se pot suprapune parţial sau total
- aceeaşi locaţie fizică se poate exprima prin mai multe variante de perechi de adrese (segment:offset)
- există puţine posibilităţi de protejare a zonelor de memorie
- orice program poate adresa orice locaţie de memorie, neputându-se impune restricţii (lucru nedorit într-un sistem multitasking)

Sunt folosite in general urmatorii registrii (de segment):

- CS – pentru adresarea instrucţiunilor
- DS – pentru adresarea datelor
- SS – pentru adresarea stivei
- ES – în anumite cazuri speciale pentru adresarea şirurilor

Programatorul poate solicita în mod explicit utilizarea unui anumit registru segment prin amplasarea unui prefix cu numele registrului (ex: ES:) în faţa operandului adresat. In mod uzual însă compilatorul este cel care detectează dacă utilizarea explicită a unui registru segment este necesară (de exemplu o variabilă nu se afla în segmentul indicat de registrul DS).

Organizarea memoriei în modul protejat

Modul protejat a fost introdus odată cu procesorul 80386 şi perfecţionat la 80486

Acest mod a fost necesar pentru a soluţiona limitările modului real, în special în ceea ce priveşte spaţiul limitat de adresare şi posibilităţile reduse de protecţie.

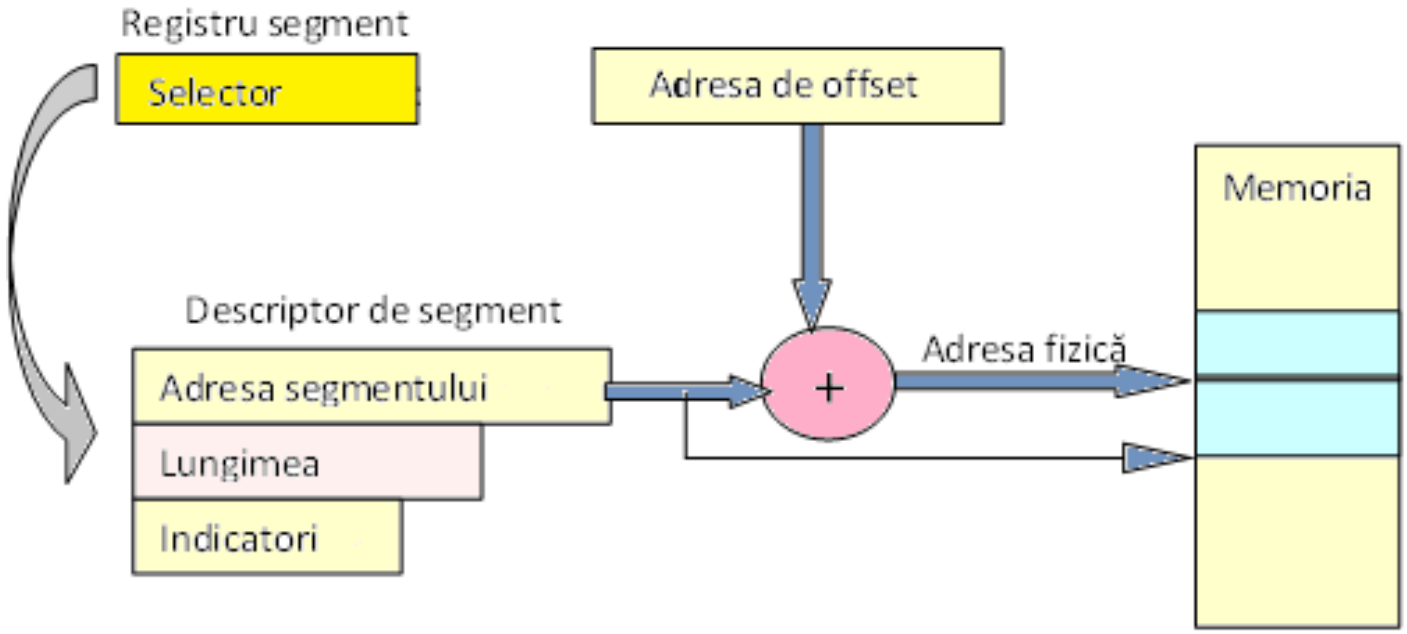
În modul protejat exprimarea adresei se face la fel prin adresă de segment şi adresa de offset, însă cu particularitatile:

- un registru segment păstrează un selector de segment şi nu adresa de început a segmentului;
- selectorul este un indicator care arată locul unde se află o structură de date care descrie un segment şi care poarta numele de descriptor de segment
- un descriptor de segment conţine: adresa segmentului (pe 32 de biţi) lungimea segmentului (pe 20 de biţi), indicatori pentru determinarea drepturilor de acces şi indicatori care arată tipul şi modul de utilizare a segmentului
- adresa de offset se exprimă pe 32 de biţi

Organizarea memoriei în modul protejat

Consecinţe:

- spaţiul maxim de adresare al memoriei se extinde la 4Gocteţi (4G = 2^{32})
- un segment are o lungime variabilă, în interval larg de la 1 octet la 4Gocteti
- se definesc trei nivele de protecţie (0, cel mai prioritar)
- un segment este accesibil numai taskului alocat şi eventual sistemului de operare
- anumite segmente pot fi blocate la scriere (ex: segmentele de cod)
- rezultă un mecanism complex de alocare şi de protecţie a zonelor de memorie



Calculul adresei fizice în modul protejat

Adrese FAR si NEAR

Adresa NEAR: o adresa in care se specifica doar offsetul, urmand ca segmentul sa fie preluat implicit dintr-un registru de segment poarta numele de *adresa NEAR* (adresa apropiata). O adresa NEAR se afla intotdeauna in interiorul unuia din cele patru segmente active.

Adresa FAR: o adresa in care programatorul specifica explicit si adresa de inceput a segmentului (adresa indepartata).

Formatul intern al unei adrese FAR: la adresa mai mica se afla cuvantul care indica offsetul, iar la adresa mai mare cu 2 (cuvantul care urmeaza) se afla cuvantul care indica segmentul.

Faptul ca o instructiune foloseste o adresa FAR sau NEAR se reflecta in continutul octetilor care codifica instructiunea.

Calculul offsetului unui operand. Moduri de adresare

- *modul registru*, daca pe post de operand se afla un registru al masinii;
- *modul imediat*, atunci cand in instructiune se afla chiar valoarea operandului (nu adresa lui si nici un registru in care sa fie continut);
- *modul adresare la memorie*, daca operandul se afla efectiv undeva in memorie. In acest caz, adresa offsetului lui se calculeaza cu formula:

***adresa_offset* = [BX | BP] + [SI | DI] + [*constanta*]**

adica , *adresa_offset* se obtine adunand urmatoarele elemente (sau numai unele dintre ele):

- continutul unuia dintre registrii BX sau BP
- continutul unuia dintre registrii SI sau DI;
- valoarea unei constante.

➔

urmatoarele moduri de adresare la memorie:

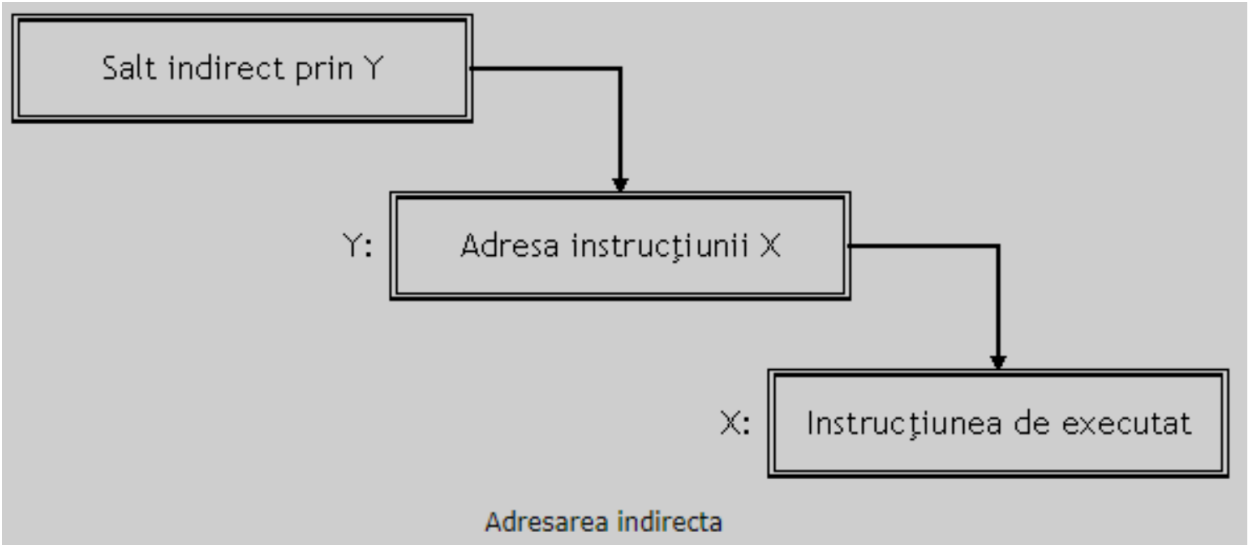
- *directa*, atunci cand apare numai *constanta*;
- *bazata*, daca in calcul apare BX respectiv BP;
- *indexata*, daca in calcul apare SI respectiv DI;

Calculul offsetului unui operand. Moduri de adresare

! La instructiunile de salt mai apar doua tipuri de adresari.

- *Adresarea relativa* indica pozitia urmatoarei instructiuni de executat, in raport cu pozitia curenta. Pozitia este indicata prin numarul de octeti de cod peste care se va sari, numar ce ia valori intre -128 si 127. O astfel de adresa mai poarta numele de *adresa scurta (SHORT Adress)*.

- *Adresarea indirecta* apare atunci cand locul viitoarei instructiuni este indicat printr-o adresa, aflata intr-o locatie, a carei adresa este data ca operand instructiunii de salt.

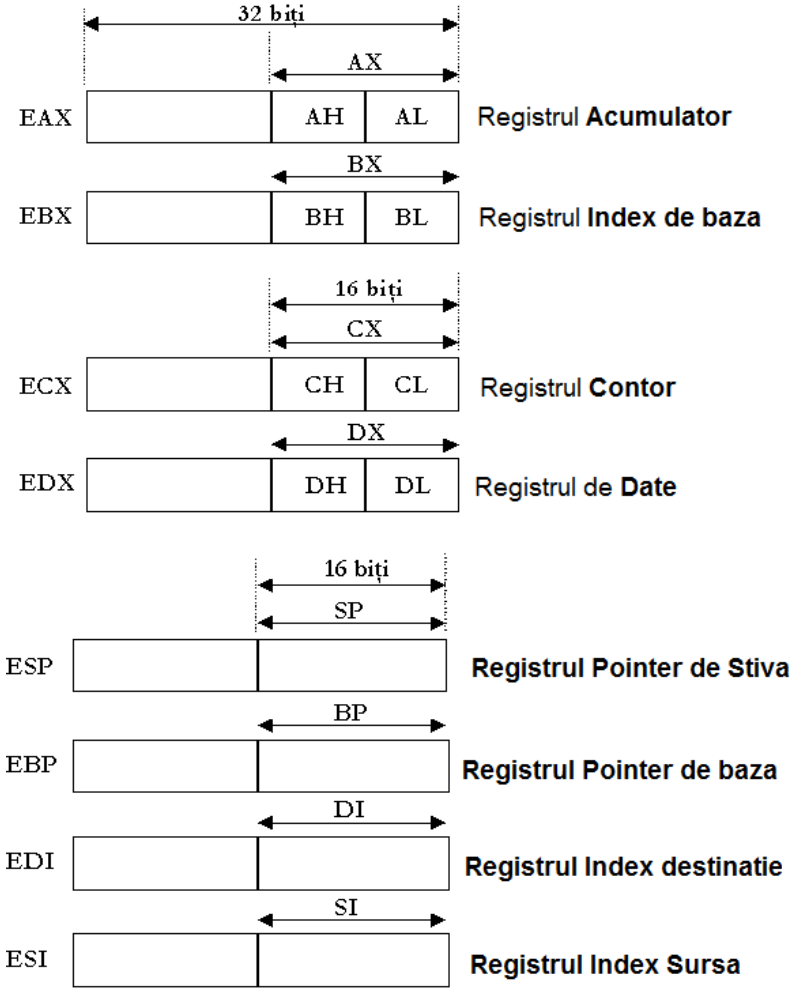


Adresarea indirecta

Adresarea indirecta asigura o mai mare flexibilitate in controlul succesiunii instructiunilor. De exemplu, in figura, continutul locatiei Y poate sa difere de la un moment la altul a executiei programului, ceea ce permite executia la momente diferite a unor instructiuni diferite in urma saltului indirect prin Y.

!!! Adresarea indirecta poate fi la randul ei adresare indirecta NEAR sau adresare indirecta FAR.

Registrii de uz general : 8



! Registrii pe 32 de biti apar doar de la 80386

! Prefixare cu E indica registru pe 32 biti si poate fi vazut ca doi registrii distincti (EAX si AX) fiecare de cate 16 biti

In continuare vom considera doar operatii pe 16 biti

In secventa:

MOV BX, 1

MOV DX, 2

ADD BX, DX

Se incarca valorile 1 si 2 in registrii BX si DX iar suma acestora se incarca in registrul BX

Cum arata in ARM assembler aceiasi secventa?

MOV R1, #1

MOV R2, #2

ADD R1, R1, R2

Diferenta intre cele doua instructiuni de adunare?.....

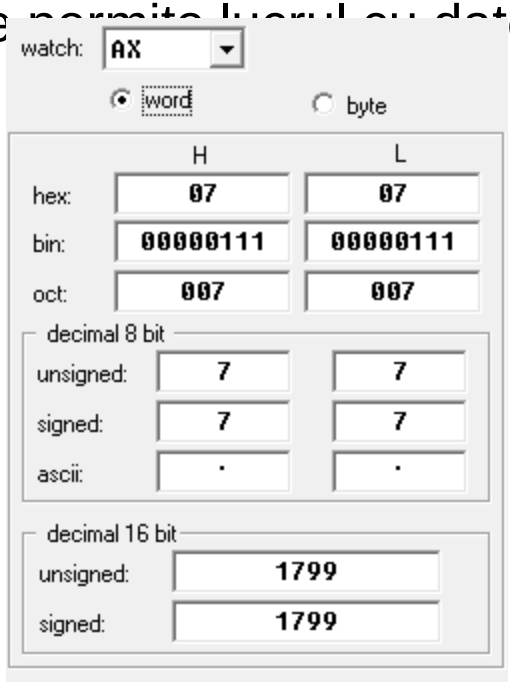
! Dar, pe langa caracteristicile generale ale registrilor generali in x86 assembler, fiecare registru are si caracteristici specifice!

Registrul AX (EAX) - acumulator

- operaţii aritmetice, logice şi de deplasare de date.
- operaţiile de înmulţire şi împărţire sunt optimizate pentru a se executa mai rapid
- folosit şi pentru toate transferurile de date de la/către porturile de Intrare/Ieşire.
- poate fi accesat pe porţiuni de 8, 16 sau 32 de biţi, fiind referit: AL (cei mai puţin semnificativi 8 biţi din AX), AH (cei mai semnificativi 8 biţi din AX), AX (16 biţi) sau EAX (32 de biţi), fapt ce permite lucrul cu date pe un octet

Exemplu:

MOV AH, 5
INC AH
MOV AL, AH
INC AH
INC AL



In AX vom avea la final 0707 (AH = AL = 07).

Registrul BX - registrul de bază (Base register)

- pentru stocare adrese ce fac referiri la structuri de date (vectori).

- pentru în operaţii aritmetico-logice

Exemplu: **MOV AX, 0**

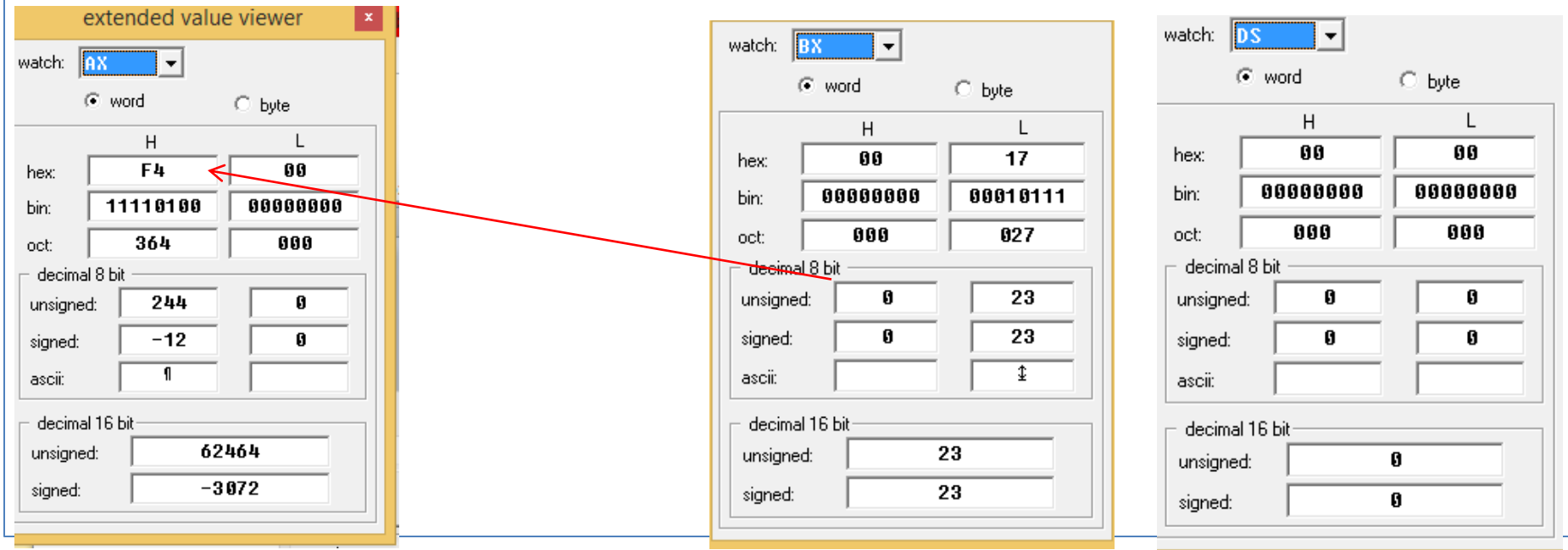
MOV DS, AX

MOV BX, 23

MOV AH, [BX]

- încarcă in AX valoarea 0 iar in AH cu valoarea din memorie aflata la adresa 23.

- se foloseste DS pt. ca BX atunci cand contine un pointer de memorie face o adresare relativa la adrea continuta in DS



Registrul CX (ECX) – contor (Counter register)

- pentru operatii de numarare/contorizare

- pentru operaţii aritmetico-logice

```
        MOV CX, 5
start:  …
        SUB CX, 1
        JNZ start
```

SAU

```
        MOV CX, 5
start:  …
        LOOP start
```

sau se poate folosi instructiunea LOOP care decrementeaza automat contorul si face salt la eticheta start, pana cand CX va contine valoarea zero

Care va fi efectul:

```
        MOV CX, 5
start:
        INC AL
        SUB CX, 1
        JNZ start
```

Registrul DX (EDX) – registru de date (Data register),

- pentru transferuri de date Intrare/Ieşire
- pentru operaţii de înmulţire/ împărţire. Impreuna cu registrul AX se foloseste in calculele ale caror rezultate depasesc dimensiunea unui cuvant.

- Instrucţiunea **IN AL, DX** copiază o valoare de tip Byte dintr-un port de intrare, a cărui adresă se află în registrul DX. Următoarele instrucţiuni determină scrierea valorii 123 în portul I/O aflat la adresa 1002:

```
MOV AL, 123
MOV DX, 1002
OUT DX, AL
```

- La înmulţire, atunci când se înmulţesc două numere pe 16 biţi, cei mai semnificativi 16 biţi ai produsului vor fi stocaţi în DX iar cei mai puţin semnificativi 16 biţi în registrul AX.

Registrul SI (Source Index)

- folosit, ca şi BX, pentru a referi adrese de memorie.

Exemplu 1:

```
MOV AX, 0
MOV DS, AX
MOV SI, 40
MOV AL, [ SI ]
```

incarcă valoarea (pe 8 biţi) din memorie de la adresa 40 în registrul AL.

Exemplu 2 (utilizare pt. string-uri):

```
CLD                 //se incrementeaza automat toti registrii legati de sir dupa operatie
MOV AX, 0
MOV DS, AX
MOV SI, 40
LODSB               //L incarca byte-ul de la adresa  DS:(E)SI in registrul  AL
```

-încarcă registrul AX cu valoarea de la adresa de memorie referită de registrul SI,

-adună valoarea 1 la SI.

!Eficienta atunci când se accesează secvenţial o serie contigua de locaţii de memorie (şiruri de caractere).

Registrul DI (Destination Index)

- Utilizare asemănătore registrului SI.

Exemplu:

MOV AX, 0

MOV DS, AX

MOV DI, 1000

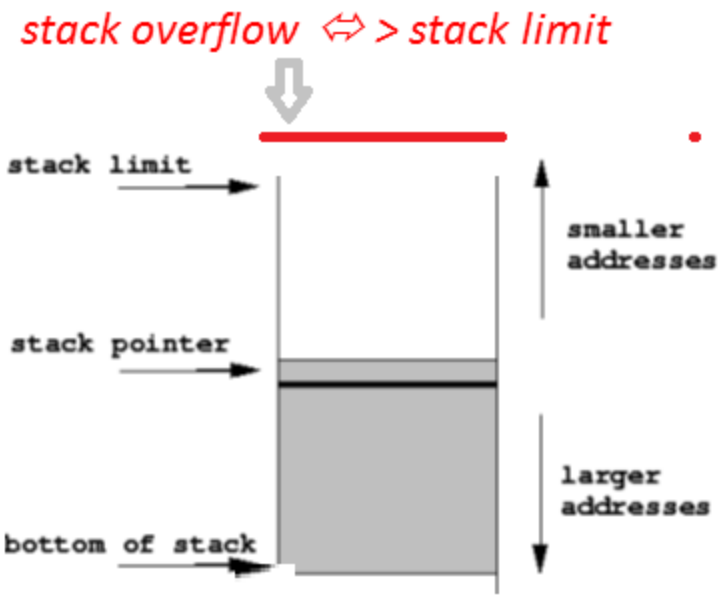
ADD BL, [DI]

- se adună la registrul BL valoarea pe 8 biţi stocată la adresa 1000.

! SI este întotdeauna pe post de pointer sursă de memorie, DI serveşte drept pointer destinaţie de memorie.

Stiva (stack) de memorie

- Stiva este o zona a memoriei unde valorile pot fi stocate şi accesate pe principul LIFO (Last In – First Out.
- este utilizată la apelul unei proceduri sau la întoarcerea dintr-un apel de procedură (principalele instrucţiuni folosite sunt CALL şi RET).
- reprezintă o zona specială de locaţii adiacente din memorie.
- conţinută în cadrul unui segment de memorie şi identificată de un selector de segment memorat în registrul SS (cu excepţia cazului în care se foloseşte modelul nesegmentat de memorie în care stiva poate fi localizată oriunde în spaţiul de adrese liniare al programului).



La fel ca si structura de date cu acelasi nume, operatiile pe stiva sunt push si pop:
push se pot elimina unul sau mai multi registrii prin setarea pointerului de stiva la o valoare mai mica
pop ….
pushing este o modalitate de a salva continutul registilor in memorie iar popping este o modalitate de a restaura continutul registrilor din memorie

Operatiile se fac prin incrementari/decrementari ale pointerului de stiva cu valoarea 4

Registrul pointer de bază, BP (Base Pointer)

- poate fi utilizat ca pointer de memorie precum regiştrii BX, SI şi DI.

- registrul BP asigurâ adresarea în segmentul de stivă.

- registrul BP ofera suport pentru accesul la parametri, variabile locale şi alte necesităţi legate de accesul la porţiunea de stivă din memorie.

Diferenţa: BX, SI şi DI sunt utilizaţi în mod normal ca pointeri de memorie relativ la segmentul DS, iar registrul BP face referire relativ la segmentul de stivă SS.

Registrul SP (Stack Pointer) - pointerul de stivă

- reţine adresa de deplasament a următorului element disponibil în cadrul segmentului de stivă.

- Este cel mai puţin „general” dintre regiştrii de uz general, deoarece este dedicat mai tot timpul administrării stivei.

- registrul BP face în fiecare clipă referire la vârful stivei, care reprezintă adresa locaţiei de memorie în care va fi introdus următorul element în stivă.
-
- introducerea unui nou element în stivă se numeşte „împingere” (în engleză *push*); operatia PUSH.

- operaţia de scoatere a unui element din vârful stivei poartă, în engleză, numele de *pop;* operatia POP.

Atentie:

- e permisă stocarea valorilor în registrul SP precum şi modificarea valorii sale prin adunare sau scădere la fel ca şi în cazul celorlalţi regiştri de uz general;

DAR

- nu este recomandat dacă nu suntem foarte siguri de ceea ce facem. Prin modificarea SP, se modifica adresa de memorie a vârfului stivei, ceea ce poate avea efecte neprevăzute (deoarece PUSH şi POP nu reprezintă unicele modalităţi de utilizare a stivei (unele resurse de sistem, precum tastatura sau ceasul de sistem, pot folosi stiva în momentul trimiterii unei întreruperi la microprocesor, adica stiva este folosită continuu, deci dacă se modifică registrul SP (adică adresa stivei), datele din noile locaţii de memorie nu vor mai fi cele corecte)

- În concluzie, registrul SP nu trebuie modificat în mod direct; el este modificat automat în urma instrucţiunilor POP, PUSH, CALL, RET. Oricare dintre ceilalţi regiştri de uz general pot fi modificaţi în mod direct în orice moment.

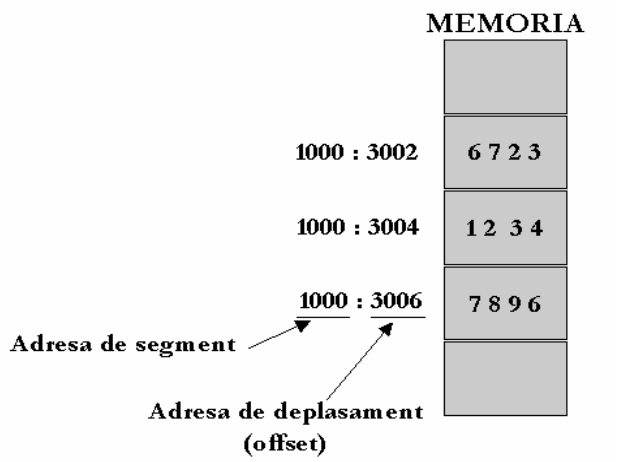
Registrul pointer de instrucţiuni (IP – Instruction Pointer)

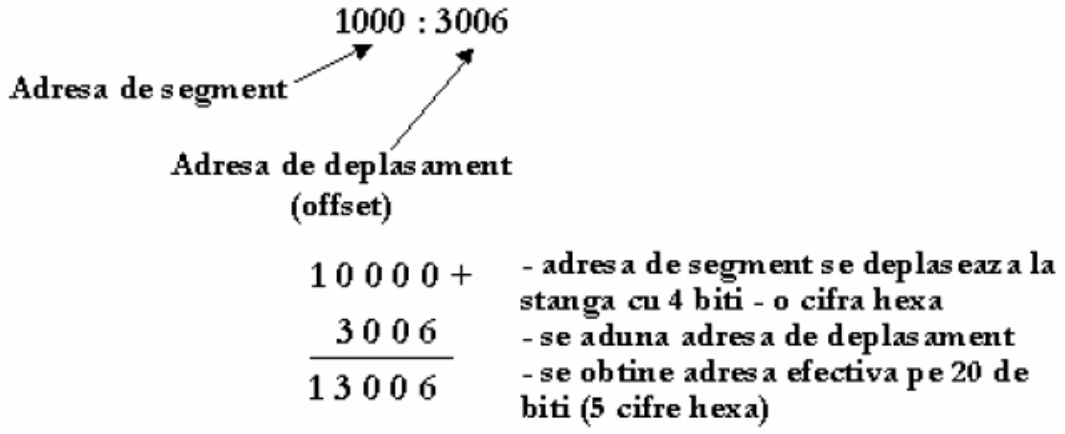
- folosit, întotdeauna, pentru a stoca adresa următoarei instrucţiuni ce va fi executată de către microprocesor. Pe măsură ce o instrucţiune este executată, pointerul de instrucţiune este incrementat şi se va referi la următoarea adresă de memorie (unde este stocată următoarea instrucţiune ce va fi executată). De regulă, instrucţiunea ce urmează a fi executată se află la adresa imediat următoare instrucţiunii ce a fost executată, dar există şi cazuri speciale (rezultate fie din apelul unei subrutine prin instrucţiunea CALL, fie prin întoarcerea dintr-o subrutină, prin instrucţiunea RET).

- Pointerul de instrucţiuni nu poate fi modificat sau citit în mod direct; doar instrucţiuni speciale pot încărca acest registru cu o nouă valoare. Registrul pointer de instrucţiune nu specifică pe de-a întregul adresa din memorie a următoarei instrucţiuni ce va fi executată, din aceeaşi cauză a segmentării memoriei. Pentru a aduce o instrucţiune din memorie, registrul CS oferă o adresă de bază iar registrul pointer de instrucţiune indică adresa de deplasament plecând de la această adresă de bază.

Regiştrii de segment

- sunt în strânsă legătură cu noţiunea de segmentare a memoriei. Premiza de la care se pleacă este următoarea: 8086 este capabil să adreseze 1MB de memorie, astfel că sunt necesare adrese pe 20 de biţi pentru a cuprinde toate locaţiile din spaţiul de 1 MB de memorie. Totuşi, registrele utilizate sunt registre pe 16 biţi, deci a trebuit să se găsească o soluţie pentru această problemă. Soluţia găsită se numeşte *segmentarea memoriei*; în acest caz memoria de 1MB este împărţită în 16 segmente de câte 64 KB (16*64 KB = 1024 KB = 1 MB).
- Segmentarea memoriei presupune utilizarea unor adrese de memorie formate din două părţi: adresa segmentului si adresa de deplasament, sau offset-ul





Regiştrii de segment

- Registrul CS – acest registru face referire la începutul blocului de 64 KB de memorie în care se află codul programului (segmentul de cod). Microprocesorul 8086 nu poate aduce altă instrucţiune pentru execuţie decât cea definită de CS. Registrul CS poate fi modificat de un număr de instrucţiuni, precum instrucţiuni de salt, apel sau de întoarcere. El nu poate fi încărcat în mod direct cu o valoare, ci doar prin intermediul unui alt registru general.
- Registrul DS – face referire către începutul segmentului de date, unde se află mulţimea de date cu care lucrează programul aflat în execuţie.
- Registrul ES – face referire la începutul blocului de 64KB cunoscut sun denumirea de extra-segment. Acesta nu este dedicat nici unui scop anume, fiind disponibil pentru diverse acţiuni. Uneori acesta poate fi folosit pentru creearea unui bloc de memorie de 64 KB adiţional pentru date. Acest extra-segment lucrează foarte bine în cazul instrucţiunilor de tip STRING. Toate instrucţiunile de tip STRING ce scriu în memorie folosesc adresarea ES : DI ca adresă de memorie.
- Registrul SS – face referire la începutul segmentului de stivă, care este blocul de 64 KB unde se află stiva. Toate instrucţiunile ce folosesc implicit registrul SP (instrucţiunile POP, PUSH, CALL, RET) lucrează în segmentul de stivă deoarece registrul SP este capabil să adreseze memoria doar în segmentul de stivă.

Formatul unei instrucţiuni în limbaj de asamblare

<nume> <instructiune/directiva> <operanzi> <;comentariu>

unde:

<nume> - reprezintă un nume simbolic opţional;

<instructiune/directiva> - reprezintă mnemonica (numele) unei instrucţiuni sau a unei directive;

<operanzi> - reprezintă o combinaţie de unul, doi sau mai mulţi operanzi (sau chiar nici unul), care pot fi constante, referinţe de memorie, referinţe de regiştri, şiruri de caractere, în funcţie de structura particulară a instrucţiunii;

<;comentariu> - reprezintă un comentariu opţional ce poate fi plasat după caracterul „;” până la sfârşitul liniei respective de cod.

Observaţie: Limbajul de asamblare **nu** este *case sensitive*.