1. Introducere

1.1 Definiții

Un **microprocesor,** sau poate mai corect **un microprocesor de uz general**, este o unitate centrală de prelucrare (CPU - Central Processing Unit), unitate prezentă în orice arhitectură de calcul, implementată însă la nivelul unui singur circuit integrat monolitic (un singur chip de siliciu, microcircuit). Cu alte cuvinte, **un microprocesor este un chip de siliciu care conține cel puțin un CPU**.

Există cel puțin 5 componente cheie prezente la un microprocesor:

- **unitatea aritmetică şi logică** (ALU-Arithmetic Logic Unit) responsabilă cu tot ce înseamnă operații de calcul aritmetic şi/sau logic
- **registrele** (registers) utilizate pentru memorarea temporară a informației cu care lucrează ALU
- **unitatea de control** (Control Unit) responsabilă cu controlul operațiilor efectuate de CPU pe baza decodificării instrucțiunilor
- un sistem **magistrale (buses)** interne şi externe, care interconectează componentele interne şi realizează conectarea cu exteriorul
- un **ceas (clock)** la baza căruia stă ceasul sistem (system clock) cel care realizează periodizarea tuturor operațiilor CPU

Alte resurse tipice prezente de obicei la un microprocesor sunt: **un sistem de întreruperi** si un **sistem de acces direct la memorie** (DMA-Direct Memory Acces)

Microprocesoarele se pot diferenția sau clasifica prin intermediul :

- setului specific de instrucțiuni (ISA- Instruction Set Architecture)
- dimensiunii cuvântului de date sau, cu alte cuvinte, numărului de biți care pot fi prelucrați într-o singură instrucțiune: 8,16, 32, 64, etc.
- caracteristicilor arhitecturale: RISC sau CISC, Von Neumann sau Harvard
- vitezei de lucru sau frecvenței maxime a ceasului sistem

Actualmente, unități funcționale altădată separate cum ar fi unitatea de prelucrare în virgulă mobilă (FPU- Floating Point Unit), unitatea de gestiune a memoriei (MMU-Memory Management Unit) sau memoria cache, sunt şi ele integrate la nivelul aceluiaşi circuit. Uneori termenii de microprocesor şi CPU sunt folosiți alternativ pentru a desemna aceiaşi entitate.

O caracteristică esențială a microprocesorului (ca de altfel a orice CPU) este **programabilitatea**. Programarea unei aplicații folosind un microprocesor se poate face, mai rar, la nivelul producătorului microcircuitului sau, mult mai des, la nivelul utilizatorului final al sistemului de calcul.

Orice sistem de calcul va avea nevoie, în afară de microprocesor (CPU), de memorie şi de modalități de conectare la dispozitivele de intrare-ieşire (I/O devices) prin intermediul cărora se face legătura cu lumea exterioară.

Plecând de la cerințele generale pentru alcătuirea unui sistem de calcul, precum şi de la o **orientare pe anumite categorii de aplicații** (calculatoare sau stații de lucru personale, stații de lucru profesionale, controlere industriale de proces, sisteme de prelucrare numerică a semnalelor) a apărut în decursul timpului şi o anumită **specializare**

a microprocesoarelor rezultând două categorii noi: microcontrolerele (**microcontrollers**) şi procesoarele numerice de semnal (**DSP**- Digital Signal Processors).

Un **microcontroler(MC, MCU)** este alcătuit dintr-o unitate centrală (CPU) căreia i s-a adăugat, **pe acelaşi microcircuit (on-chip), memorie şi dispozitive periferice** (specializate pe interfața cu mediul exterior). El poate constitui astfel un sistem de calcul de sine stătător, realizat la un raport preț/performanță optim, orientat pe controlul interacțiunii cu lumea exterioară (controler industrial). Un microcontroler este o arhitectură de calcul orientată pe control: el trebuie să "simtă" evenimentele externe şi să le "răspundă" (în sensul de a controla) cât mai eficient, din punct de vedere al timpului de calcul şi al resurselor folosite. Această interacțiune este, de cele mai multe ori, bazată pe utilizarea **întreruperilor**. Din acest motiv, toate microcontrolerele înglobează şi **un sistem de întreruperi** performant si simplu de utilizat.

Un **procesor numeric de semnal (DSP)** este caracterizat de existența unei unități centrale (CPU) **specializată pe operații de calcul intensiv şi foarte eficient, în virgulă fixă sau virgulă mobilă**. Arhitectura CPU este optimizată in acest caz pentru executarea într-un timp minim a operațiilor de calcul necesare implementării unor algoritmi de prelucrare numerică a semnalului.

Majoritatea procesoarelor numerice de semnal au o arhitectură de sistem de tip Harvard (cu spații de memorie separate pentru program şi date). Se întâlnesc şi aici, din ce în ce mai des, caracteristici specifice microcontrolerelor: prezența memoriei on-chip şi a unor dispozitive periferice specializate. La ora actuală, **granița dintre anumite categorii de microcontrolere, microprocesoare şi multe din procesoarele numerice de semnal este din ce în ce mai puțin distinctă**.

1.2 Aplicații şi utilizări – sisteme integrate (embedded systems)

Din momentul apariției primelor calculatoare zise "personale" şi până în prezent, aplicațiile microprocesoarelor (în primul rând) se pot grupa în două mari categorii: **sisteme de calcul de uz general -SCUG** (General Purpose Computing System), în sensul de calculator mai mult sau mai puțin personal) şi **sisteme integrate** (**embedded systems**).

OBSERVAȚIE Traducerea în limba română a conceptului "embedded system" nu este încă unanimă: se mai întâlnesc cel puțin şi variantele de "sistem încapsulat" sau "sistem înglobat" (provenite din alte limbi de origină latină).

Un **sistem integrat (SI)** este un **sistem de calcul programat să realizeze numai o anumită sarcină sau categorie îngustă de sarcini**. Nu este relevant cât de complexă este această sarcină. Spre deosebire de un calculator personal care şi el execută sarcini , un sistem integrat execută mereu această sarcină, din momentul pornirii până în momentul opririi. Programarea sa nu poate fi modificată decât în sensul în care programatorul aplicației l-a prevăzut şi permis.

Un sistem integrat nu este altceva decât **o componentă a unui sistem mai mare**, la fel ca o poartă logică sau o altă componentă electronică, care nu are de îndeplinit decât o singură sarcină. El este un sistem de calcul programat să execute această sarcină, foarte complexă sau foarte simplă. Deoarece el nu este decât o componentă, la rândul său poate fi utilizat drept componentă a altui sistem integrat.

Gradul de complexitate al sistemului integrat în sine, puterea sa de calcul şi eventual dimensiunile fizice sunt legate firesc de sarcina pe care trebuie să o îndeplinească şi pot varia de la complexitatea hardware a unui circuit cu 8 pini care îndeplineşte funcţiile câtorva porţi logice până la complexitatea hardware echivalentă a unui calculator personal foarte performant, din ultima generaţie.

In practica de toate zilele există nenumărate exemple de sisteme (sau aplicaţii) care incorporează şi care sunt controlate prin intermediul unuia sau mai multor astfel de sisteme integrate:

- periferice pentru calculatoare (imprimante, scanere, etc.)
- electronică de consum (aparatura audio, video, camere, etc.)
- aparatură electrocasnică (frigidere, maşini de spălat, cuptoare cu microunde, etc.)
- aparatură de comunicaţii (telefonie mobila, routere, servere back-up, etc)
- industria automobilului (control motor, control frânare, etc.)
- roboţi industriali şi roboţi mobili
- aparatură medicală
- aparatură pentru automatizarea şi climatizarea spaţiilor de locuit sau de lucru
- aparatură pentru măsurare şi control în industrie

Toate aceste aplicaţii denotă drept caracteristici comune:

- realizarea unei sarcini bine definite pentru produsul, echipamentul sau sistemul în care sunt integrate

- ele nu permit interacţiunea cu utilizatorul decât în măsura în care această interacţiune face parte din sarcina pe care o are de îndeplinit

- ele sunt considerate doar o parte a unui produs, echipament sau sistem

Complexitatea unui sistem integrat este dată de atât de componenta hardware, deja menţionată, cat si de cea software. Prin intermediul software-ului este determinată funcţionalitatea sistemului integrat. Complexitatea software poate de asemenea varia de la echivalentul unui cod, de sine stătător, cu dimensiunea de câteva zeci de octeţi (Bytes) până la complexitatea unei aplicaţii de pană la zeci de megaocteţi (MBytes), care la rândul său rulează sub un sistem de operare, eventual unul de timp real.

Vom mai puncta în final doar câteva din diferenţele cele mai importante dintre un SI şi un SCUG.

1. Un SI este proiectat astfel încât să poată rula o categorie îngustă, foarte specializată de aplicaţii spre deosebire de un SCUG destinat să ruleze o gamă cât mai largă de aplicaţii.

2. SI, în general, nu sunt programabile la nivelul utilizatorului final, spre deosebire de SCUG la care utilizatorul final poate programa în întregime aplicaţia.

3. Performanţele în materie de viteză de calcul ale unui SI sunt fixate şi prevăzute din faza proiectare, pe când la un SCUG se doreşte mereu creşterea acestora.

4. Pentru majoritatea SI se cere să aibă un consum de energie cât mai mic, pe când la un SCUG această cerinţă nu este critică (poate doar din considerente ecologice).

5. Un SI trebuie să aibă un preţ de cost cât mai mic posibil, pe când un SCUG are de regulă un cost ridicat.

OBSERVAŢIE Din acest punct de vedere, am putea acum redefini varietatea de microprocesor pe care am numit-o **microcontroler** ca fiind **un circuit destinat realizării unui sistem integrat cu ajutorul unui număr minim de circuite (ideal numai unul!)**.

1.3 Scurtă cronologie

1947 - apariția tranzistorului bipolar (Bell)
1959 - primul circuit integrat bipolar (Texas Instruments)
1969 - primul circuit integrat LSI, un circuit de memorie de 1Kb (Intel)
1971
- primul microprocesor de 4 biți Intel 4004
- primul microprocesor de 8 biți 8008 proiectat de Texas Instruments, dar implementat de Intel; capacitate de adresare de 16kB, 45 de instrucțiuni
1973 - apare primul minicalculator bazat pe 8008
1974-1975
- apare microprocesorul de 8 biți Intel 8080, capacitate de adresare de 64kB, 75 de instrucțiuni
- apare primul calculator "personal" Altair 800
- apare floppy-diskul
- primul sistem de operare pentru un calculator personal, CPM-80 (Control Program for Microcomputer)
- Motorola (devenit ulterior in anii 2000 Freescale) lansează microprocesorul de 8 biți M6800, pentru prima oară împreună cu o familie de circuite periferice care implementează funcții bine precizate (interfață paralelă, interfață serială, ceas-temporizator)
- MOS Technologies lansează 6501 şi apoi 6502, compatibile cu 6800, dar la un preț de cost mai scăzut; pe baza lui 6502 se vor realiza calculatoarele personale Apple1 şi Apple2
- doi proiectanți (Fagin şi Shima) plecați de la Intel realizează şi lansează pentru firma Zilog microprocesorul de 8 biți Z80, unul din cele mai utilizate, care mai există şi azi pe piață
1976 - Intel propune microprocesorul de 16 biți 8086, microprocesor care va fi realizat 3 ani mai târziu

1979
- Intel lansează pe piață 8086 şi apoi 8088 (cu interfață externă de 8 biți)
- Motorola (Freescale) lansează microprocesorul de 16 biți M68000 (se numeşte aşa deoarece avea 68000 de tranzistori !); spre deosebire de 8086 avea un set de instrucțiuni ortogonal: se elimină instrucțiunile speciale, toate registrele, tipurile de date şi modurile de adresare sunt disponibile pentru orice instrucțiune (sau aproape!); 68008 este varianta cu interfață externă de 8 biți
- Zilog lansează pe piață propriul microprocesor de 16 biți, Z8000, incomplet testat şi validat; nu va fi utilizat aproape niciodată în aplicații semnificative
1980...
- din acest moment atât Intel cât şi Motorola (Freescale) pun bazele unor familii de procesoare denumite generic 80x86 şi respectiv 68K; deşi îşi au aparent originea în microprocesoarele de 8 biți, nu există nici o compatibilitate la nivel de cod cu acestea; această compatibilitate există însă începând cu primul microprocesor de 16 biți din familie (cu cele apărute ulterior)
80x86:
8080 **8086 80286 80386 80486 Pentium Pentium II Pentium III Pentium IV**...

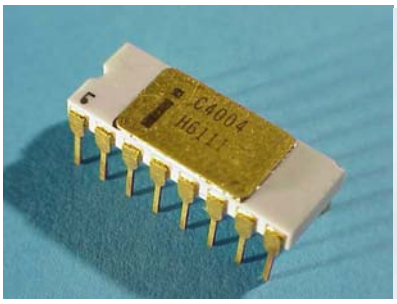
68K:
6800 **68000 68020 68030 68040 …Cold Fire**
.... **PowerPC..**

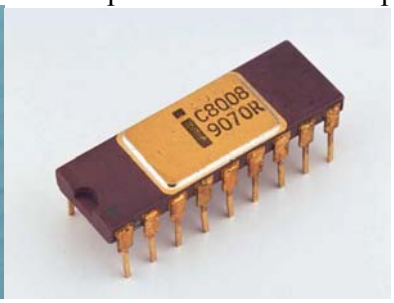
1981 - IBM alege Intel 8088 pentru seria de calculatoare personale IBM PC şi IBM XT

1986-1987 - apariția primelor microprocesoare RISC (Reduced Instruction Set) comerciale: AMD2900 (AMD), ARM (Acorn), MIPS R2000 (Stanford), SPARC (Sun), PA-RISC (Hewlett Packard); la mijlocul anilor '90 ele (variante noi ale acestora) vor deveni disponibile pentru stații de lucru (workstation) desktop

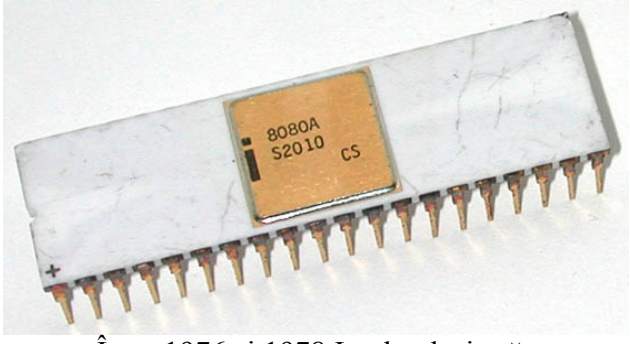
2. O perspectivă istorică a familiei de microprocesoare 80x86

Firma Intel a dezvoltat şi introdus pe piaţă **primele microprocesoare de 4 biţi I4004 şi I4040** la începutul anilor 70. Le-au urmat rapid variantele de 8 biţi I8008 şi apoi I8080. Ultimul a stat la baza unuia din primele calculatoare "personale", Altair 800.

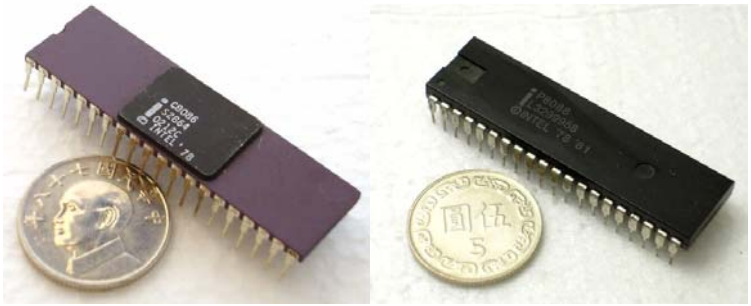




Pe piaţa microprocesoarelor de 8 biţi Intel a fost repede şi puternic concurat de competitori cum ar fi Motorola (6800), MOS Technologies (6502) sau Zilog (Z80). Microprocesoarele acestora erau fie mai uşor de programat (6800), fie uşor de programat şi mai ieftine (6502) fie, şi mai rău, cod compatibile cu 8080 dar cu o arhitectură mult îmbunătăţită (Z80). La nivelul anului 1978 toate calculatoarele personale utilizau fie 6502, fie Z80. Chiar şi o variantă îmbunătăţită a lui 8080, anume 8085 (care se mai găseşte şi astăzi pe piaţă ca un 80C85 !), nu aducea multe lucruri noi.



Între 1976 şi 1978 Intel a decis că este momentul unui salt calitativ şi cantitativ în această competiţie şi a produs un microprocesor de 16 biţi care să ofere substanţial mai multă putere de calcul decât competitorii de pe piaţa microprocesoarelor de 8 biţi. Acesta iniţiativă a dus la apariţia microprocesorului 8086, la acel moment cel mai performant microprocesor de 16 biţi single chip (dintr-un sigur circuit).



La momentul apariţiei lui 8086 memoria (circuitele de memorie semiconductoare) era o componentă scumpă şi foarte scumpă a sistemului. Una din probleme era că un program pentru un microprocesor de 16 biţi tinde să consume mai multă memorie decât

echivalentul său de 8 biți. Intel, conştient de riscurile pe care le prezenta un cost final prea mare al sistemului (datorat memoriei în acest caz) a făcut un **efort special la proiectare pentru a obține o densitate cât mai mare a instrucțiunilor**. Astfel este posibil să încapă cât mai multe instrucțiuni în cât mai puțină memorie. Intel şi-a realizat atunci obiectivul, dimensiunea codului fiind comparabilă cu cea a procesoarelor de 8 biți, dar urmările neplăcute vom vedea că se mai resimt şi astăzi. Din acest motiv spațiul maxim de memorie adresabil de 1Mbyte (1024K) pentru un 8086 nu constituia un dezavantaj vizibil.

În momentul proiectării lui 8086 durata medie de viață a unui procesor era de câțiva ani. Experiența Intel cu microprocesoarele precedente i-a învățat că proiectanții de sisteme vor renunța rapid la vechea tehnologie dacă cea nouă este radical mai bună. Astfel ei au proiectat 8086 plecând de la ideea, justificată de experiența lor, că orice compromis făcut pentru obținerea densității mari a instrucțiunilor va fi remediat la variantele ulterioare

Competitorii Intel au lansat şi ei propriile variante de 16 biți : M68000 (Motorola, actualmente Freescale), Z8000 (Zilog) sau NS16032/16 (National Semiconductor). Toate aceste microprocesoare au avut ca obiectiv de proiectare uşurința în programare şi nu densitatea instrucțiunilor. Toate erau din punct de vedere al dezvoltării de programe mult mai bune decât 8086.

Intel a introdus apoi pe piață circuitul **8088**, un 8086 cu interfața externă de 8 biți (magistrala de date) şi cu acelaşi spațiu adresabil de memorie, din dorința de a reduce costurile sistemului în ansamblu.

La începutul anilor '80 Intel a început de asemenea să lucreze la presupusul succesor al lui 8086, numit iAPX432, o arhitectură revoluționară (la vremea aceea) de 32 de biți.

Evenimentul care a schimbat însă istoria microprocesoarelor a fost, în 1980, decizia IBM de a intra pe piața calculatoarelor personale, pentru a realiza ceva similar calculatoarelor Apple II (6502) sau TRS-80 (Z80), cele mai populare la vremea respectivă.

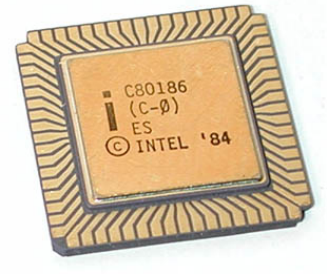
Ei au ales ca microprocesor un I8088, probabil datorită faptului că puteau obține un sistem minimal cu doar 16Kocteți de RAM şi cu un set de circuite periferice ieftine (acesta era de fapt şi scopul pentru care Intel a produs 8088). De aici a rezultat ceea ce se va numi **IBM PC,** un calculator care s-a vândut mai bine decât se aştepta oricine.

Pe măsură ce popularitatea şi vânzările IBM PC creşteau (odată şi cu apariția "clonelor" posibilă datorită arhitecturii deschise a IBM PC), o mulțime de firme au început să scrie software de aplicație sau sistem de operare pentru 8086 (8088), marea majoritate în limbaj de asamblare.

Din păcate pentru Intel, nimeni sau aproape nimeni nu era interesat de iAPX432 şi de limbajul asociat de programare, de nivel înalt, numit Ada (presupus succesorul pentru Pascal concurent). Mai mult softul scris pentru PC nu putea fi rulat pe 432, care s-a dovedit fiind şi foarte lent. iAPX432 a devenit în final cel mai mare eşec comercial din istoria firmei.

La începutul anilor 80 Intel a dezvoltat noile procesoare **80186** şi **80188**, variante îmbunătățite ale lui 8086 şi 8088 care, foarte important, erau compatibile la nivel de cod; acesta era un lucru nou pentru firmă deoarece până acum nu exista compatibilitate de cod între generații succesive de microprocesoare (de la 4040 la 8080 sau de la 8080 la

8086). Intel a recunoscut astfel efortul depus în dezvoltarea de soft pentru 8086, decizând să creeze o variantă superioară atât hardware cât şi software, total cod compatibilă. Deşi 80186 nu a fost utilizat prea mult în PC-uri, el este cunoscut pentru aplicațiile sale în control, fiind încă întâlnit destul de des şi produs în continuare.



Popularitatea neaşteptată a IBM PC a creat o adevărată problemă pentru Intel. Nu mai era adevărat că un proiectant de sistem va alege un procesor mai bun când acesta va apărea pe piață, chiar de ar fi să rescrie tot softul. IBM şi zecile de mii de programatori doreau ceva mai bun decât 8086, dar în acelaşi timp să păstreze şi softul deja scris. Dacă ar fi fost să se treacă la un nou procesor, ofertele Motorola, Zilog sau National erau mult mai atractive.

Astfel Intel a început să facă ceva care probabil a salvat-o ca firmă dar care nu este o "reuşită" de lăudat în materie de arhitecturi de calcul: a început să creeze variante superioare de procesoare (prin noile caracteristici introduse) care să fie capabile să execute programe scrise pentru precedenții membrii ai familiei.

Scăderea prețurilor la circuitele de memorie precum şi dorința programatorilor de a realiza programe din ce în ce mai sofisticate a făcut ca bariera de 1Mbyte pentru memoria adresabilă să devină un inconvenient major. În acelaşi timp M68000 (Motorola) a oferit încă de la început un spațiu de adresare linear (nesegmentat) de 16M .

Pentru a nu pierde piața PC-urilor Intel a introdus un nou procesor, **80286**, care aducea o serie de îmbunătățiri consistente: noi instrucțiuni care uşurau programarea şi, în primul rând, un nou mod de lucru, modul zis "protejat", mod care permitea adresarea a 16MB de memorie. Unele modificări de arhitectură precum şi creşterea frecvenței de ceas a făcut ca 80286 să fie de circa 10 ori mai rapid decât un 8088 într-un sistem IBM PC.
La rândul său IBM a utilizat 80286 în noul **IBM PC AT**, un imens succes comercial atât al produsului IBM cât şi al nenumăratelor clone apărute pe piață.

Realizând că 80x86 (deocamdată cu x=1 sau 2) reprezintă o sursă sigură de venituri, Intel şi-a concentrat eforturile în proiectarea de noi procesoare care să fie capabile să execute aceeaşi cod cu precedentele, dar să ofere performanțe îmbunătățite, devenind primul producător din lume ca număr de procesoare vândute.

O diferență majoră între procesoarele Intel şi cele ale competitorilor (Motorola şi NS) era, la acel moment, că procesoarele acestora aveau o arhitectură internă de 32 de biți pe când Intel rămăsese la una de 16 biți. Astfel că Intel a lansat noul **80386** şi odată cu el cea ce se va numi **IA32** (Intel Architecture - Arhitectura Intel pe 32 de biți) .

80386 a fost un procesor remarcabil la momentul apariției: în afară de arhitectura de 32 de biți, capacitatea de adresare a crescut la 4GB, a rezolvat unele probleme legate de natura "segmentată" a memoriei, a introdus un mecanism de paginare ("paging") al memoriei, aproape s-a dublat numărul instrucțiunilor şi au fost adăugate noi facilități de gestiune a memoriei ("memory management") şi depanare ("debugging"). 80386 a fost astfel una din cele radicale modificări pe care le-a suferit familia 80x86.

80386 rula programele existente pentru 8088 şi 80286 mai rapid, chiar şi la aceiaşi frecvență de ceas, astfel că el a fost adoptat de proiectanți pentru performanțele lui superioare. Din păcate, cel puțin pentru început, nu s-a scris şi un nou soft (aplicații de 32 de biți!) care să-i valorifice noile caracteristici.

Momentul apariției lui 80386 este şi momentul în care au apărut pe piață o nouă familie de procesoare numite **RISC** ("Reduced Instruction Set Computer" - procesoare cu set redus de instrucțiuni). Exemple reprezentative de astfel de procesoare (de fapt familii) sunt, nu neapărat în ordinea apariției: SPARC, MIPS, PowerPC, Alpha. Pe lângă alte caracteristici pozitive unul din avantajele acestora era că executau o instrucțiune maşină (toate instrucțiunile RISC sunt simple, şi relativ puține ca număr) într-un singur ciclu de ceas, deci foarte eficient. Comparativ, procesorul 80x86, până în acest moment, intră în categoria **CISC** ("Complex Instruction Set Computer" - procesoare cu set complex de instrucțiuni) la fel ca şi competitorii săi 68000, 32016, Z8000.

Instrucțiunile lui 80386 se executau într-un număr de cicluri de ceas extrem de variat: de la câteva cicluri la peste o sută! Deşi o comparație 80386 - RISC nu este total relevantă, pentru că o instrucțiune 80386 (sau CISC în general) realiza ceea ce realizau 2 sau mai multe instrucțiuni RISC, percepția proiectanților de sisteme a fost că 80386 era mai lent decât acestea, la aceiaşi frecvență de ceas.

Noul procesor **80486** va aduce şi el două îmbunătățiri majore. 80486 integrează pentru prima oară o unitate prelucrare în virgulă mobilă (FPU- Floating Point Unit). Aceste zise **co-procesoare în (de) virgulă mobilă** existau deja ca circuite de sine stătătoare: **8087, 80287** şi **80387**. Prin integrarea pe acelaşi circuit a lui 80387 s-a reuşit accelerarea operațiilor în virgulă mobilă şi scăderea costului acestora.

A doua îmbunătățire majoră a fost introducerea unui **pipeline pentru instrucțiuni** care a permis suprapunerea execuției a două sau mai multe instrucțiuni, efectul global fiind de reducere a numărului de cicluri per instrucțiune. Astfel pentru instrucțiuni mai degrabă simple se obținea pe ansamblu, în condiții ideale, randamentul dorit de o instrucțiune per ciclu.

Nu în ultimul rând la acest procesor apare pentru prima oară integrată (Level 1) o **memorie cache**- memorie imediată sau asociativă, o memorie de mare viteză care permite, în anumite condiții, creşterea vitezei de prelucrare prin scurtarea timpului de acces la memorie.

În timp ce Intel introducea pipeline-ul la familia 80x86 producătorii de procesoare RISC au introdus pe piață primele arhitecturi de procesoare zise **superscalare** care execută mai mult de o instrucțiune per ciclu maşină. Altă problemă era că unitatea de virgulă mobilă integrată la 80486 era mult mai lentă decât echivalentele ei prezente la procesoarele RISC. Astfel producătorii de stații de lucru ("workstation") utilizate în inginerie (proiectare asistată, CAD, CAM) s-au orientat către procesoarele RISC, mai rapide decât ceea ce putea oferi Intel la acel moment.

Din punct de vedere al programatorului era o diferență foarte mică între un 80486 şi o pereche 80386-80387, astfel că pentru inginerul soft 486 era doar o combinație 80386-80387 mai rapidă. Noile instrucțiuni adăugate nu erau foarte utile la nivel de aplicație, ci doar la nivel de sistem de operare.

Intel a început să lucreze la un nou procesor cu o arhitectură superscalară şi care să ofere performanțe de virgulă mobilă mult îmbunătățite, apropiindu-se astfel de performanțele procesoarelor RISC. Noul proiect avea doar un număr mic de noi instrucțiuni, similar lui 486, utile mai ales la nivel de sistem de operare.

Noul procesor nu s-a mai numit 80586 ci **Pentium™.** Motivul a fost confuzia de pe piață legată de identificarea procesoarelor. Intel deja nu mai era singura companie care producea procesoare 80x86 fiind concurată de AMD, Cyrix ş.a.. Până la 80486 arhitectura internă era destul de simplă astfel că şi companii mici au reuşit să reproducă funcționalitatea unui procesor Intel. Cu 80486 lucrurile stăteau cu totul altfel: era un circuit destul de complex care depăşea posibilitățile micilor companii. Unele companii cum ar fi AMD au cumpărat licența pentru 486 şi astfel au produs circuite care erau total compatibile cu cele de la Intel. Alte companii au încercat să creeze propria versiune de 486 dar nu au reuşit decât parțial (nu au reuşit să integreze FPU sau pipeline-ul, etc.); deşi circuitele erau marcate ca fiind 80486 ele erau un fel de 80386 cu mici îmbunătățiri. Această situație a dus la o mare confuzie pe piață.

În momentul în care se cumpăra un PC zis 486 nu se mai ştia dacă ceea ce s-a cumpărat avea un procesor 486 sau un 80386 remarcat. Acesta este şi motivul pentru care Intel a promovat campania de marcare "Intel Inside" care permitea să se cunoască dacă se utiliza un procesor de la Intel sau de la un alt producător (deşi, de exemplu, procesoarele 486 de la AMD erau total compatibile, fiind de fapt procesoare Intel !).

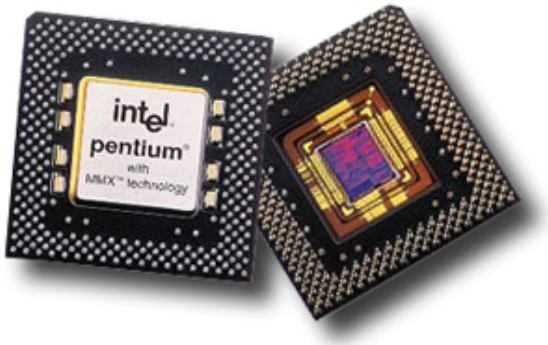
Pentru a nu se repeta această situație cu noul procesor, Intel a creat marca înregistrată "Pentium". Alți producători şi-au creat propriile variante de 80586 (unele

numite chiar aşa) dar nu au mai putut să folosească marcarea "Pentium". Aceasta a fost un lucru pozitiv pentru că, la acel moment, aceste procesoare zise 586 nu atingeau performanţele unui Pentium.

Pentium a consolidat poziţia Intel pe piaţa calculatoarelor personale, cu performanţe apropiate de RISC şi cu foarte mult soft compatibil. Doar Apple Macintosh, staţiile de lucru UNIX de mare performanţă şi serverele mai foloseau procesoare RISC, ele reprezentând mai puţin de 10% (ca număr) din piaţa calculatoarelor desktop.

Din dorinţa de a concura şi pe piaţa serverelor de performanţă Intel a dezvoltat procesorul **Pentium Pro** bazat pe o nouă micro arhitectură numită **P6** şi pe un nou proces tehnologic BiCMOS 0.6um. Acesta avea câteva caracteristici care-l făceau ideal pentru aplicaţii de tip server: capacitate de calcul pe 32 de biţi mai bună (în detrimentul celei de 16 biţi), suport mai bun pentru multiprocesare (serverele de performanţă au două sau mai multe procesoare), perfecţionarea pipeline-ului. Din păcate marea majoritate a softului de aplicaţie, la momentul lansării lui Pentium Pro, era un soft de 16 biţi care se executa mai lent pe un Pentium Pro decât pe un Pentium, la aceiaşi frecvenţă de ceas. Astfel, deşi a fost utilizat în unele servere, el nu a fost niciodată foarte răspândit comparativ cu celelalte procesoare Pentium.

Un alt motiv pentru semieşecul comercial al lui Pentium Pro a fost apariţia, la scurt timp, a lui **Pentium MMX** (MultiMedia eXtension). Acesta reprezenta o dezvoltare a unui Pentium standard căruia i s-a adăugat un nou set de instrucţiuni zise MMX (cca. 60), instrucţiuni în virgulă fixă, destinate să ofere putere mai mare de prelucrare în aplicaţiile audio şi video (multimedia). Acest procesor a devenit repede extrem de utilizat, eliminând practic Pentium Pro de pe piaţa calculatoarelor personale.



Rezultatul corectării deficienţelor de prelucrare de 16 biţi de la Pentium Pro şi adăugării setului de instrucţiuni MMX a fost materializat prin noul procesor **Pentium II** .

Apariția Pentium II a mai fost însoțită şi de descoperirea unei noi realități. Până la introducerea acestuia, Intel, ca de altfel şi restul de producători de procesoare, au presupus că un utilizator va dori întotdeauna mai multă putere de calcul în sistemul său. Chiar dacă ei nu au nevoie explicită de calculatoare mai rapide, producătorii de software vor dezvolta noi sisteme de operare şi aplicații din ce în ce mai complexe (şi mai lente ca execuție) necesitând din ce în ce mai multă putere de calcul. Pentium II a dovedit că această idee este greşită. Un utilizator mediu (şi aceştia sunt cei mai mulți) are nevoie de editare de texte, e-mail, acces la Internet, suport multimedia, calcul tabelar (spreadsheet), facilități de editare grafică relativ simple, etc. Majoritatea acestor aplicații erau suficient de rapide cu procesoarele existente, iar cele care erau mai lente, erau din alte cauze (cum ar fi viteza de comunicație în accesul la Internet limitată de modem).

Intel a descoperit astfel că utilizatorii preferau costisitorului Pentium II procesoarele concurenței, compatibile x86, mult mai ieftine, şi care le erau suficiente ca putere de calcul. Dându-şi seama de această realitate Intel a dezvoltat o variantă cu cost redus (şi performanțe reduse) a lui Pentium II pe care a numit-o **Celeron**. Celeron era un Pentium II la care fusese eliminată memoria cache integrată pe chip (Level 1- L1 cache). Fără acest cache Celeron avea o viteză puțin mai mare de jumătate din cea a lui Pentium II, dar performanța sa era comparabilă cu cea a competitorilor. Ulterior, astfel de variante Celeron au apărut au apărut şi pentru succesorii lui Pentium II, toate având resurse cache L1 diminuate.

Pentru piața stațiilor de lucru de mare performanță s-a creat o a treia variantă a lui Pentium II şi anume **Xeon**. Xeon a corectat faptul că Pentium II suportă nativ multiprocesare doar pentru 2 procesoare, fiind foarte dificil de utilizat mai mult de două procesoare. În plus s-a mărit memoria cache integrată. Astfel Xeon a devenit (împreună

cu noile versiuni multiprocesor ale sistemelor de operare UNIX şi Windows NT) o alternativă credibilă a procesoarelor RISC.

Pentium III introduce pe lângă multe alte îmbunătăţiri (noi instrucţiuni în virgulă fixă, cache) şi o extensie a setului de instrucţiuni numită SIMD (Single Instruction Multiple Data). Acest set de instrucţiuni asigură o performanţă de virgulă mobilă sporită, permiţându-i să concureze din acest punct de vedere cu cele mai bune procesoare RISC. Din acest moment, şi mai cu seamă odată cu apariţia lui **Pentium IV**, având la lansare pe piaţă frecvenţe de ceas de peste 1GHz, se pare că Intel a reuşit să tranşeze în favoarea sa competiţia cu procesoarele RISC (cel puţin pentru moment).

Performanţele au fost posibile şi datorită faptului că Intel, în măsura posibilului, a împrumutat unele lucruri din tehnologia RISC. Astfel setul de instrucţiuni (la nivel de microcod, inaccesibil utilizatorului) a fost împărţit în două părţi: un set de instrucţiuni simple numit "miez RISC" şi un set de instrucţiuni complexe.

Intel admite faptul că a dus deja arhitectura 80x86 cât de departe a fost posibil. Limitările acestei arhitecturi provin din deciziile anilor '76-'78 legate de densitatea instrucţiunilor şi din dorinţa de a menţine compatibilitatea codului.

Este mult mai uşor să creezi o arhitectură performantă de procesor, pornind de la zero şi mai ales fără a trebui moştenite aplicaţii care să ruleze pe noua platformă de calcul. Este remarcabil că Intel a putut să ducă atât de departe o arhitectură care plăteşte şi azi preţul unor decizii strategice luate cu un sfert de secol în urmă.

Cea mai mare problemă actuală pentru Intel este că producătorii RISC şi-au extins arhitecturile la 64 de biţi. Aceasta duce la o viteză mai mare de calcul pentru operaţiile interne şi indirect la capacitatea de a adresa mai mult de 4 GB de memorie, acest din urmă factor fiind cel mai important pentru staţii şi servere. Există la ora actuală servere de mare performanţă care au mai mult de 4GB de memorie instalată! Acesta este motivul pentru care Intel avea nevoie de un procesor de 64 de biţi care să concureze corespondentele RISC.

La mijlocul anilor '90 Intel a realizat un parteneriat strategic cu Hewlett-Packard cu scopul de a crea un nou procesor de 64 de biţi , bazat o arhitectură RISC proprietate a

HP numită **PA** (Precision Architecture). Noul procesor va executa cod 80x86 într-un mod special de lucru zis de "emulare" şi va rula cod nativ de 64 de biţi cu un nou set de instrucţiuni. Se utilizează de asemenea un nou concept tehnologic şi anume **VLIW** ("Very Long Instruction Word"- cuvânt -cod- instrucţiune foarte lung). Ideea este de a utiliza un cod instrucţiune foarte lung care să gestioneze mai multe operaţii în acelaşi timp. Mai mult, deoarece sarcinile îndeplinite sunt independente (şi la CISC sarcinile îndeplinite de o instrucţiune pot fi complexe raportat la RISC), aceasta va permite procesorului să execute mai multe instrucţiuni în paralel.

La începutul anului 2001 primele astfel de procesoare rulau cod x86 de 32 de biţi mult mai lent decât Pentium III sau IV. Nu numai emularea încetinea lucrurile dar şi faptul că, pentru primele astfel de procesoare, frecvenţa de ceas era la jumătate din cea a lui Pentium IV. Situaţia este oarecum similară cu Pentium Pro care trebuia să ruleze cod de 16 biţi. Mai mult noul procesor (familia **IA64** - Intel Architecture 64 bits) se bazează masiv pe o tehnologie a compilatoarelor care încă nu există ca atare şi foloseşte o arhitectură încă netestată comercial (VLIW). Aici lucrurile seamănă puţin cu istoria proiectului iAPX432 şi a compilatorului Ada.

Intel speră că utilizatorii vor trece la IA64 atunci când vor avea nevoie de putere de prelucrare pe 64 de biţi, iar noi rămâne să sperăm ca mai există şi o variantă de rezervă. Pe de altă parte AMD crede că utilizatorii doresc un procesor de 64 de biţi bazat pe 80x86, trecând la noua arhitectură aşa cum s-a trecut de la 8086 şi 80286 la 80386.
Doar timpul poate spune care din cele două firme va avea succes cu noul lor procesor cu arhitectură de 64 de biţi.

Familia 80x86

Procesor/ an apariţie	**Tranzistoare pe cip**	**Frecvenţa de ceas la apariţie**	**Viteză de calcul (MIPS*)**	**Memorie cache**	**Memorie adresabilă**
8086 / 1978	29K	8 MHz	0.8	-	1M
8088 / 1979	29K	5 MHz	0.33	-	1M
80286 / 1982	134k	12.5 MHz	2.7	-	16MB
80386 / 1985	275k	20 MHz	6	-	4GB
80486 / 1989	1.2M	25 MHz	20	8K Level 1	4GB
Pentium/ 1993	3.1M	60 MHZ	100	16K L1	4GB
PentiumPro /1995	5.5M	200MHz	440	16K L1 256/512K L2	64GB
PentiumII/ 1997	7.5M	266MHz	466	32K L1 256/512K L2	64GB
Pentium III/ 1999	8.2M	500 MHz	1,000	32K L1 512K L2	64GB
Pentium IV/ 2000	42M	>1GHz	1,700	32K L1 512K L2	64GB

* MIPS - Millions Instructions Per Second; o măsură brută a puterii (vitezei) de calcul exprimată în milioane de instrucţiuni în virgulă fixă executate pe secundă;
MFLOPS - Millions FLOating Point instructions per Second: milioane de instrucţiuni în virgulă mobilă executate pe secundă

3. INTRODUCERE ÎN ARHITECTURA INTEL 80X86

Procesorul (CPU - unitatea centrală) a familiei Intel 80x86 este clasificată ca având o arhitectură de **Maşină Von Neumann** (MVN). Un calculator (o maşină) Von Neumann are trei componente principale: **unitatea centrală de prelucrare** (CPU-Central Processing Unit), **memoria** şi **dispozitivele de intrare/ieşire** (Input /Output Devices - I/O ports). Cele trei componente sunt interconectate prin intermediul magistralei de sistem (system bus).

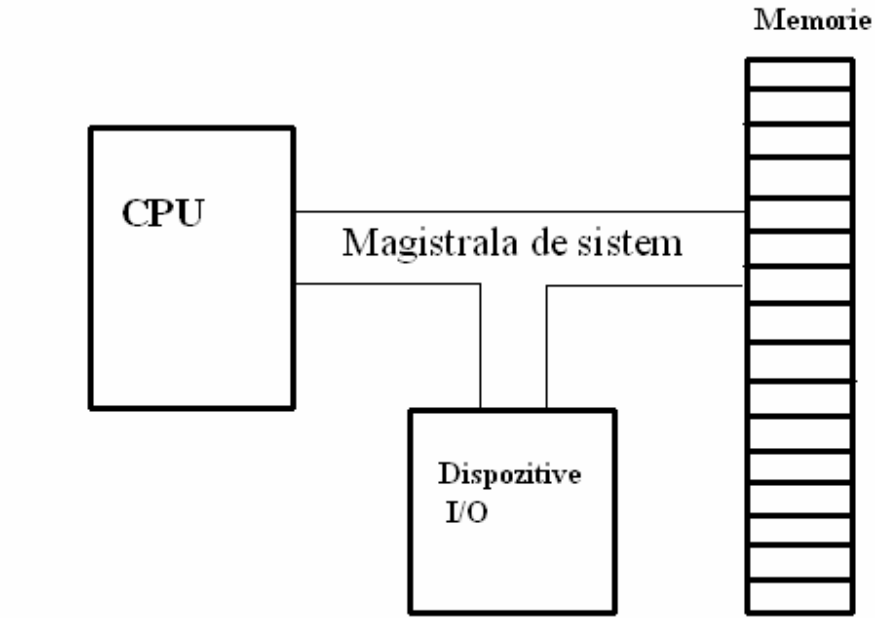


Figura 1

În cazul MVN toate acțiunile de prelucrare au loc la nivelul unității centrale (CPU). Toate calculele se efectuează aici, datele şi instrucțiunile maşină stau în memorie până când unitatea centrală are nevoie de ele. Pentru unitatea centrală dispozitivele I/O sunt similare memoriei pentru că ea poate memora (scrie) date pe un dispozitiv de ieşire sau poate citi date de pe un dispozitiv de intrare. Singura diferență aparentă este asocierea acestora cu ceea ce există în exteriorul sistemului (interfața cu lumea exterioară).

Magistrala de sistem (System Bus)

Magistrala de sistem conectează între ele componentele MVN. Un procesor 80x86 (la fel ca multe alte procesoare) are drept componente trei magistrale majore : **adrese, date şi control**. Aceste magistrale sunt în general diferite de la un procesor la altul, ele fiind clasificate după natura informației vehiculate. De exemplu implementarea magistralelor de date este diferită la 8088 comparativ cu 80386, dar în ambele cazuri se vehiculează date între unitatea centrala, memorie şi dispozitivele I/O.

Din punct de vedere electric, **nivelele logice** implicate pentru **componentele tipice conectate la aceste magistrale în cazul lui 80x86 (de la 8086/88 până la Pentium şi Pentium MMX !!)**, **sunt compatibile TTL (chiar dacă se utilizează şi tensiuni de alimentare de 3.3V)**.

La procesoarele noi sau relativ noi (de la Pentium II la Pentium IV) pentru magistrala de sistem se utilizează o nouă interfață electrică, care nu este compatibilă TTL, în variantele: GTL (Gunning Transceiver Logic), GTL+ sau AGTL (Assisted Gunning Transceiver Logic).

Magistrala de date (Data Bus)

Dimensiunea (numărul de linii de semnal) al acestei magistrale este diferită în cadrul familiei 80x86, ea definind într-un fel şi "mărimea" procesorului.

Într-un sistem 80x86 tipic magistrala de date poate avea 8,16,32 sau 64 de biți (de linii de date). Procesoarele 8088 şi 80188 au un bus de date de 8 biți. 8086, 80186, 80286 şi 80386SX au o magistrală de 16 biți. 80386DX, 80486 şi Pentium Overdrive au o magistrală de date de 32 biți.

Procesoarele Pentium, Pentium MMX, Pentium Pro, Pentium II, Pentium III şi Pentium IV au o magistrală de date de 64 biți. Variantele ulterioare pot avea dimensiuni ale magistrale mai mari (ex. AMD 86-64).

Existența unei magistrale de 8 biți nu înseamnă că un procesor este limitat doar la un singur tip de date de 8 biți (la un octet sau byte). Înseamnă de fapt că procesorul poate accesa doar un octet într-un ciclu de memorie (în cazul MVN). Astfel cei 8 biți ai magistralei de date la un 8088 pot transmite doar jumătate din informație în unitatea de timp (un ciclu de memorie) comparativ cu magistrala de date de 16 biți de la 8086.
Acesta este motivul pentru care procesoarele cu o magistrală de date de 16 biți sunt în mod natural mai rapide decât cele cu o magistrală de 8 biți, ş.a.m.d.. Dimensiunea magistralei de date afectează performanțele sistemului mai mult decât dimensiunea oricărei alte magistrale.

Legat tot de "mărimea" procesorului (8,16,32 sau 64 de biți) una din modalitățile de definire consideră că ea este dată de dimensiunea celui mai mare registru întreg de uz general al unității centrale. Astfel, 8088 deşi are o magistrală de date de 8 biți este de fapt un procesor de 16 biți, iar un Pentium cu o magistrală de date de 64 de biți este un procesor de 32 de biți.
OBSERVATIE Distincția referitoare la registru întreg (virgulă fixă) este făcută pentru a nu lua în considerare registrele unității de prelucrare în virgula mobilă (FPU), atunci când ea există, registre care pot avea o dimensiune mai mare.

Deşi este evident posibil să prelucrăm şi date pe 12 biți cu un 80x86, formatul preferat este cel de 16 biți deoarece oricum procesorul preia şi manipulează 16 biți de date, preluarea făcându-se întotdeauna în grupe de 8 biți . Astfel manipularea datelor în formate **multiplu de 8 biți** este întotdeauna cea mai eficientă în acest caz.

Procesoarele familiei 80x86 cu magistrale de date de 16,32 sau 64 de biți pot prelucra date cu dimensiuni până la acelea ale magistralei, dar ele pot accesa şi unități mai mici, de 8,16 sau 32. Orice ce se poate face cu o magistrală mai mică se poate face şi cu una mai mare diferind doar timpul in care se accesează datele respective.

Magistrala de adrese (Address Bus)

Magistrala de adrese este cea prin intermediul căreia se diferenţiază locaţiile de memorie sau dispozitivele I/O în cazul unui transfer de date, asociindu-le acestora o adresă unică (în mod tipic). Când doreşte să acceseze o locaţie particulară de memorie sau un anumit dispozitiv I/O, procesorul plasează pe magistrala de adrese o valoare (o adresă); circuitele asociate memoriei sau dispozitivelor I/O recunosc această adresă şi fac ca acestea să citească informaţia de pe magistrala de date sau să plaseze informaţia pe aceiaşi magistrală. Folosind **n** linii de adresă (magistrala de adresă de n biţi) procesorul poate furniza 2^n valori distincte ale adresei.

Primii membrii ai familiei 80x86, 8086 şi 8088 au magistrale de adrese cu n = 20 biţi, astfel încât spaţiul de adresă are 2^{20} = 1,048,576 locaţii de memorie distincte, un spaţiu de adrese foarte limitat conform cerinţelor actuale. Acest dezavantaj a fost corectat la variantele ulterioare (vezi tabel).

Viitoarele procesoare din familie vor avea probabil magistrale de adrese de 40, 48 sau 64 de biţi.

Procesor	**Linii Adrese**	**Memorie maxim adresabilă**	
8088	20	1,048,576	1M
8086	20	1,048,576	1M
80188	20	1,048,576	1M
80186	20	1,048,576	1M
80286	24	16,777,216	16M
80386SX	24	16,777,216	16M
80386DX	32	4,294,976,296	4 G
80486	32	4,294,976,296	4G
Pentium	32	4,294,976,296	4G
Pentium Pro	36	68,719,476,736	64G
Pentium II	36	68,719,476,736	64G
Pentium III	36	68,719,476,736	64G
Pentium IV	36	68,719,476,736	64G

Magistrala de control (Control Bus)

Magistrala de control reprezintă un set variat de semnale prin care procesorul comunică cu restul sistemului.

În primul rând trebuie să existe semnale care să stabilească sensul transferului de date **de la procesor (scriere- Write, WR)** sau **către procesor (citire -Read, RD)**. Astfel există două linii de semnal Read şi Write care stabilesc sensul transferului de date. În afară de acestea mai există ceasuri de sistem, linii de întrerupere, de stare ş.a.m.d. Componenţa exactă a acestei magistrale este diferită în cadrul familiei, dar anumite linii de control sunt comune şi vor fi menţionate în continuare.

Semnalele **Read** şi **Write** mai sus menţionate controlează deci sensul accesului pe magistrala de date, ele fiind active în "0" (/RD, /WR) şi **mutual exclusive**.

Semnalele de tip **Byte Enable** (activare byte/octet) permit procesoarelor cu magistrale de date de 16,32 sau 64 biţi să acceseze date cu dimensiuni mai mici decât magistrala (8,16,32 biţi).

Familia 80x86 are **două spații distincte de adresare : unul pentru memorie şi altul pentru dispozitivele I/O.** În timp ce magistrala de adrese pentru memorie are dimensiuni diferite, cea pentru I/O are întotdeauna 16 biți, permițând accesarea a 65,536 locații (sau porturi I/O) diferite, un număr mai mult decât suficient pentru orice aplicație.
OBSERVAȚIE Designul original IBM PC permitea accesarea numai a 1024 din acestea (existau doar 10 linii de adresă disponibile pe magistrala externă ISA).

Deşi familia 80x86 suportă două spații de adresare diferite, **nu există două magistrale de adrese separate** (memorie şi I/O). Se utilizează aceiaşi magistrală de adrese, dar se folosesc linii de control diferite pentru Read şi Write din spațiul de memorie al dispozitivelor I/O (active tot în "0" şi **mutual exclusive**). Când aceste semnale sunt active, sunt luați în considerare **numai cei mai puțin semnificativi 16 biți ai magistralei de adrese**. Aceste semnale sunt la rândul lor mutual exclusive cu cele pentru accesarea memoriei (cu excepția unor cicluri speciale).

4. O prezentare sumară a microprocesoarelor Intel 80x86

Pentru aproape orice fel de microprocesor există cel puțin trei componente care alcătuiesc ceea ce s-ar numi **un model de programare** sau un **model pentru programator**: **registrele**, **un model al memoriei** precum şi **setul de instrucțiuni** cu **modurile de adresare** ale operanzilor.

4.1 Registrele de uz general

Din punct de vedere al utilizatorului (şi mai ales al programatorului) cele mai importante componente ale unității centrale (CPU) sunt **registrele.** Se pot pune în evidență 4 categorii de registre: **de uz general, de uz special accesibili aplicației, de segment şi de uz special în mod kernel (nucleu).**

Vom aborda numai primele două categorii de registre. Registrele de segment sunt esențiale pentru înțelegerea mecanismului de adresare al memoriei atunci când programăm în limbaj de asamblare, dar în programarea modernă (de 32 biți) de nivel înalt (Windows 9x, Windows NT, 2000 sau XP, Linux, BeOS, etc.) ele sunt, ca prezență, aproape transparente pentru utilizator.

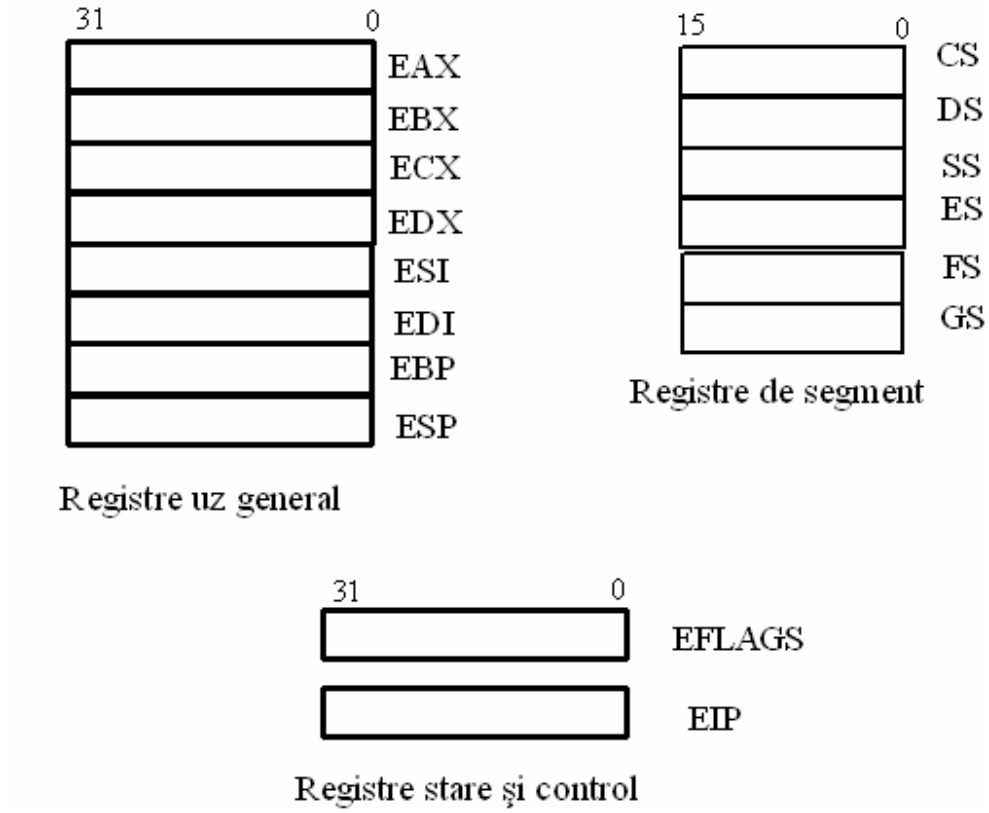


Figura 2

Registrele de uz special asociate modului kernel (nucleu) sunt utilizate atunci când se scriu programe de nivel sistemic cum ar fi sistemele de operare, executive de timp real (un tip particular de sistem de operare) şi programele de depanare (debuggers).

Există **8 registre** de 32 de biți de "uz general" care pot fi utilizate de orice aplicație: **EAX, EBX, ECX, EDX, ESI, EDI, EBP** si **ESP**.

Prefixul E provine "Extended - extins" şi este folosit pentru a diferenția registrele de 32 de biți (atunci când ele există, la IA-32!) de cele de 16 biți: **AX, BX, CX, DX, SI, DI, BP** şi **SP** .

În final mai există şi 8 registre de 8 biți : **AL, AH, BL, BH, CL, CH, DL** şi **DH**.

Din păcate acestea **nu sunt registre separate, notațiile utilizate fiind mai degrabă alternative**. **Nu există 24 de registre independente**: registrele de 32 de biți sunt suprapuse peste cele de 16 biți care la rândul lor sunt suprapuse peste cele de 8 biți (vezi figura). Astfel este important de ştiut că ele **nu sunt independente**.

Modificarea unuia dintre ei înseamnă modificarea a cel puțin unui alt registru şi cel mult a altor trei registre. De exemplu modificare lui EAX poate afecta AL, AH şi AX.

O greşeală adesea întâlnită în programarea în limbaj de asamblare este modificarea nedorită a valorilor unor registre, provenită din neînțelegerea acestei suprapuneri.

Registrele de 16 biți sunt mapate direct pentru 8086/88 şi 80286 (există doar ele împreună cu corespondentele de 8 biți), iar cele de 32 de biți există doar la procesoarele cu arhitectură IA 32(de 32 de biți).

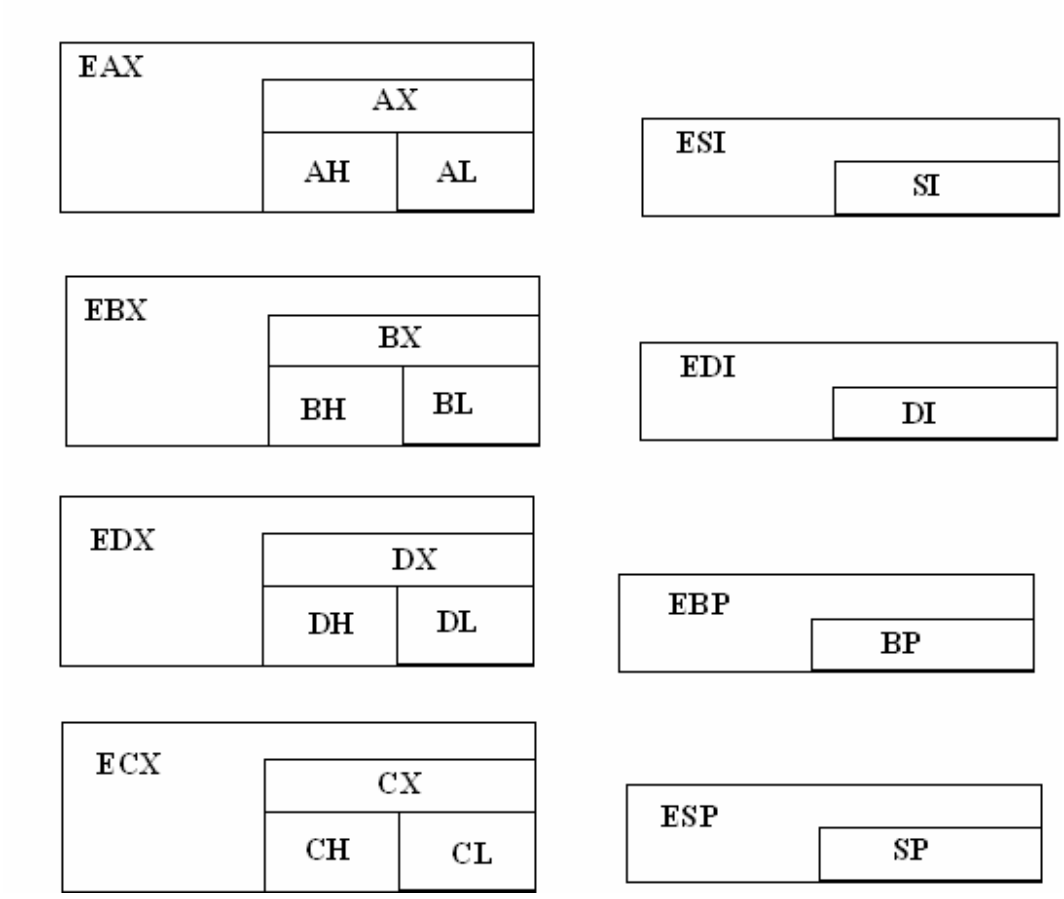


Figura 3

O descriere sumară a modului lor curent de utilizare este dată în continuare:
EAX - Acumulator pentru operanzi şi rezultat
EBX - Pointer (indicator) pentru date în segmentul DS
ECX - Contor pentru operațiile cu şiruri şi buclări
EDX - Pointer(indicator) pentru operațiile de I/O.

Următoarele registre sunt destinate implementării unui mecanism de adresare, ele conținând de regulă informație care reprezintă **o adresă sau un index** folosit la calcularea unei adrese, cu alte cuvinte, sunt folosite ca pointeri (indicatori) în sensul limbajului C.
ESI - Pointer pentru date în segmentul indicat de registrul DS; pointer sursă pentru operațiile cu şiruri
EDI - Pointer pentru date (sau destinație) în segmentul indicat de registrul ES; pointer destinație pentru operațiile cu şiruri
ESP - Stack Pointer - Indicator stivă (în segmentul SS). **Stiva** (Stack), prezentă la orice arhitectura de procesor, reprezintă o resursă pentru memorarea temporară şi automată a informației (adrese şi-sau date). Ea poate fi folosită şi ca o resursă de memorare de uz general. Utilizarea ei se face prin intermediul unui indicator de stivă.
EBP - Pointer de bază (Base) pentru date în segmentul stivă (în segmentul SS).

Aceste registre există fie ca registre 32 de biți la IA-32 (de exemplu ESI), fie ca registre de 16 biți la 8086 (de exemplu SI)) şi nu pot fi utilizate decât ca un tot unitar (de exemplu, nu poate fi accesat separat un octet din componența lor, iar SI nu este un sub registru al lui ESI, etc.).

Un alt registru important şi oarecum similar ca funcționalitate registrelor anterioare este **EIP** (Instruction Pointer) - indicator instrucțiune. Rolul său este similar unui registru numit şi **PC** - Program Counter - contor program, denumire întâlnită la majoritatea arhitecturilor de calcul. Acest registru de 32 de biți (la IA32) conține **adresa din memorie unde se află următoarea instrucțiune maşină care va fi executată**.
OBSERVAȚIE Mai corect, aşa cum vom vedea, el conține offsetul în segmentul de cod curent pentru următoarea instrucțiune ce va fi executată.

Deşi **EIP nu poate fi alterat direct sau explicit într-un program utilizator** (nu poate fi accesat prin software), instrucțiunile care îi modifică valoarea tratează acest registru ca un operand implicit, el fiind astfel ascuns utilizatorului. Este cazul instrucțiunilor care controlează fluxul program: salturi (JMP), apeluri de subrutine (CALL) şi revenirea din subrutine (RET), întreruperi şi excepții.

Toate procesoarele IA folosesc o preluare anticipată a codului instrucțiune ("**prefetch**"). Din acest motiv **nu există corespondență între valoarea magistralei de adresă pe durata accesării (citirii) opcodului unei instrucțiuni şi valoarea EIP**. **Chiar dacă generațiile diferite de procesoare IA folosesc mecanisme diferite de prefetch, funcția registrului EIP de a controla direct fluxul program este aceiaşi la toate**.

Un registru important este registrul **EFLAGS**, un registru de 32 de biți (la IA32) care cuprinde mai multe variabile boolene (de tip bit).

Cele mai multe din aceste variabile sunt fie rezervate pentru modul kernel (sistem de operare) fie nu prezintă interes în programarea unei aplicații.

Doar 8 din aceşti biți (sau **flags**- fanioane) prezintă interes pentru programarea unei aplicații în limbaj de asamblare. Aceştia sunt: depăşire, direcție, activare întrerupere, semn, zero, transport auxiliar, paritate şi transport. În figură sunt prezentați **primii 16 biți** ai lui EFLAGS **care coincid cu registrul de 16 biți de la 8086/88 numit acolo PSW - Program Status Word - Cuvânt de stare program**.

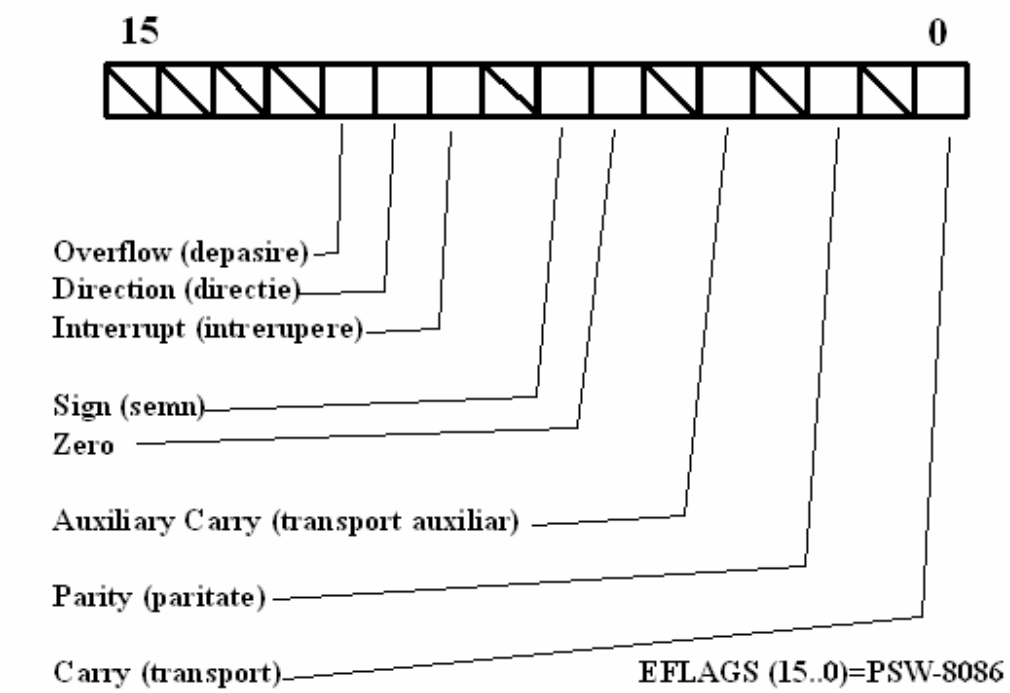


Figura 4

Din aceştia patru sunt în particular deosebit de importanţi: depăşire (**O**verflow), transport (**C**arry), semn (**S**ign) şi **Z**ero. Împreună ei (**OF, CF, SF, ZF**) formează ceea ce s-ar numi **indicatori de condiţie**. Starea acestor variabile boolene (de tip bit) **poate fi testată,** ca un rezultat al calculelor anterioare, şi permite **luarea unor decizii**. De exemplu, după compararea a doi operanzi, indicatorii de condiţie pot da informaţii despre valoarea relativă a celor doi operanzi (egal, mai mic, mai mare). Există instrucţiuni speciale pentru testarea acestor flaguri, individuală sau în anumite combinaţii.

Flagul transport auxiliar (**AF**) este folosit în operaţiile aritmetice cu date având formatul de reprezentare BCD.

Starea flagului Direcţie (**DF**) stabileşte sensul transferului de informaţie in cazul operaţiilor cu blocuri de date(şiruri).

Flagul întrerupere (**IF**) permite activarea sau dezactivarea sistemului de întreruperi.

Un aspect important legat de unitatea centrală 80x86 este faptul că toate **operaţiile de calcul propriu-zis trebuie să implice un registru**. Pentru a aduna două locaţii de memorie (conţinutul lor) cu memorarea sumei într-o a treia, unul din operanzi trebuie transferat într-un registru, al doilea operand adunat la acest registru, iar rezultatul din registru transferat în locaţia de memorie pentru rezultat. **Registrele trebuie astfel utilizate ca intermediari în aproape orice fel de calcul**. Acest este şi motivul pentru care **arhitectura de calcul a lui 80x86** se numeşte **orientată pe registru.**

Un alt aspect important este legat de denumirea de "uz general". **Uz general nu înseamnă neapărat ca utilizatorul îl poate modifica după dorinţă şi fără nici un fel de precauţii.**

Registrul **SP/ESP este indicatorul de stivă (Stack Pointer)** esențial în implementarea mecanismului de apel al subrutinelor (procedurilor), lucru care îl face de neutilizat ca registru de uz general.

Similar stau lucrurile cu BP/EBP a cărui utilizare ca registru de uz general e limitată. Până la urmă toate registrele au un anumit scop special, lucru care le limitează utilizarea într-un anumit context.

4.2 Registrele de segment

Existența acestei categorii de registre este specifică arhitecturii Intel (IA), fiind legată de ceea ce se numeşte **modelul segmentat al memoriei** utilizat la primele procesoare Intel 8086, model considerat primitiv la ora actuală, dar care regăseşte într-o formă evoluată şi la ultimele procesoare 80x86.

Conceptele de bază asociate acestui model sunt:

- există trei tipuri diferite de informație care se doreşte memorată: **program**, **date** si **stivă (stive)**
- există două tipuri de adrese: **adrese logice** (pentru programator) şi **adrese fizice** sau liniare (corespunzătoare liniilor fizice de adresă existente); adresa fizică este construită automat de CPU pe baza adresei logice
- spațiul de memorie este împărțit în **blocuri** (care pot fi şi suprapuse) având o dimensiune maximă fixată, blocuri numite **segmente**; există o adresă liniară în cadrul segmentului numită **offset-deplasare** (față de adresa de început a segmentului)
- adresa logică are forma generală **selector_segment : offset**, cele două componente fiind memorate în registre separate, unul de 16 biți - registrul de segment si unul de 16, 32 sau 36 de biți-(E)IP

Fiecare din registrele de segment prezentate este asociat nativ unui anumit tip de informație memorată: **program** (cod), **date** (variabile) şi **stivă**.

Registrul **CS (Code Segment)** conține selectorul de segment pentru **segmentul de cod,** acolo unde sunt memorate instrucțiunile care se execută. Procesorul preia instrucțiunile din segmentul de cod, folosind o **adresă logică** formată din selectorul de segment din CS şi conținutul lui EIP. EIP conține adresa liniară **în interiorul segmentului** de cod pentru următoarea instrucțiune ce trebuie executată. CS nu poate fi încărcat în mod explicit de programul aplicație. El este încărcat implicit de instrucțiuni sau de operații interne ale procesorului care modifică fluxul program (salturi, apeluri de subrutine, întreruperi, comutarea task-urilor).

Registrele **DS (Data Segment), ES (Extra Segment), FS (extra Segment)** şi **GS (extra Segment)** conțin selectori de segment pentru **4 segmente de date**. Disponibilitatea simultană a 4 segmente de date permite un acces eficient şi sigur la diferite tipuri de structuri de date. De exemplu, se pot crea şi utiliza patru segmente separate de date: unul pentru structurile de date din modulul curent, altul pentru datele exportate dintr-un modul de nivel superior, un al treilea pentru structurile de date create dinamic şi un al patrulea pentru date comune cu un alt program. Pentru a accesa segmente de date adiționale, aplicația trebuie să încarce selectorii de segment corespunzători în aceste registre.

Registrul **SS (Stack Segment)** conține selectorul pentru **segmentul de stivă**. Toate operațiile cu stiva folosesc registrul SS pentru identificarea segmentului care va

conține stiva. Spre deosebire de registrul CS, **registrul SS poate fi încărcat explicit**, cea ce permite aplicațiilor să creeze stive multiple şi să comute între ele.

Cele patru registre de segment **CS, DS, SS şi ES sunt aceleaşi cu registrele de segment de la 8086/88 şi 80286**, pe când **FS şi GS au apărut odată cu IA32 (odată cu 80386),** fiind registre suplimentare de segment, de uz general, similare lui ES.

Registrele de segment (CS, DS, SS, ES, FS, şi GS) conțin o valoare pe 16 biți utilizată ca un selector de segment. Un **selector de segment** este de fapt un pointer (indicator) special care identifică un segment în memorie. Pentru a accesa un segment particular din memorie, selectorul de segment pentru acel segment trebuie să fie prezent în registrul de segment corespunzător.

Atunci când scriem codul pentru o aplicație în limbaj de asamblare, selectorii de segment se creează folosind directive ale asamblorului şi simboluri corespunzătoare. Asamblorul şi link-editorul (în mod tipic) creează apoi valorile actuale pentru selectorii de segment pe baza acestora. Atunci când codul scris este pentru un sistem de operare poate fi nevoie să controlăm direct selectorii de segment.

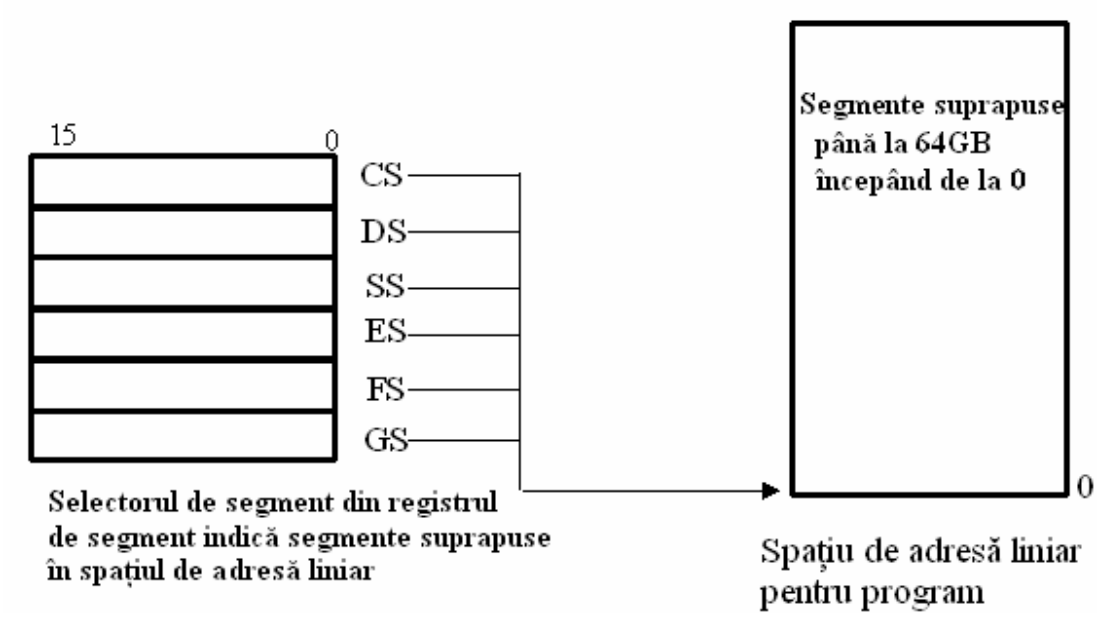


Figura 5

Cum anume sunt folosite registrele de segment depinde de tipul modelului de gestiune al memoriei (memory management) utilizat.

Model de memorie plat ("flat") sau **liniar** a apărut odată cu arhitectura IA-32 (odată cu 80386). Deşi el este numit de multe ori şi **nesegmentat**, în realitate el se bazează tot pe segmente. Ideea de bază este ca **registrele care conțin selectorii de segment să fie încărcate doar o singură dată, la inițializare-început, astfel încât să indice întreg spațiul de adresare de 4 sau 64GB.** Bazându-ne pe aceste valori implicite, registrele de segment nu vor mai trebui încărcate de fiecare dată. Deoarece se utilizează aceste valori implicite, adresele pot fi exprimate într-o formă pură ca adrese de 32 sau 36 de biți.

Când utilizăm un **astfel de model plat**, registrele de segment sunt **încărcate inițial** cu selectori de segment care indică segmente suprapuse, fiecare din ele începând

de la adresa 0 a spațiului de adresă liniar. Aceste **segmente suprapuse** includ spațiul de adresă liniar pentru program. În mod tipic sunt definite două segmente suprapuse: unul pentru program (cod) şi unul pentru date (variabile) şi stive. Registrul de segment CS indică un segment de cod (program) şi toate celelalte registre de segment indică segmente de date sau stivă.

Nu este obligatoriu ca segmentele să aibă dimensiunea maximă de 4 sau 64GB, este suficient ca ele să aibă lungimea necesară aplicației. Astfel este posibilă şi depistarea situației în care se încearcă accesarea unei zone de memorie care nu aparține aplicației.

Avantajele majore al unei aplicații scrise folosind modelul plat sunt că ea se execută mai rapid (cu 15-20%) şi este mai compactă decât aceiaşi aplicație scrisă folosind un model segmentat propriu-zis.

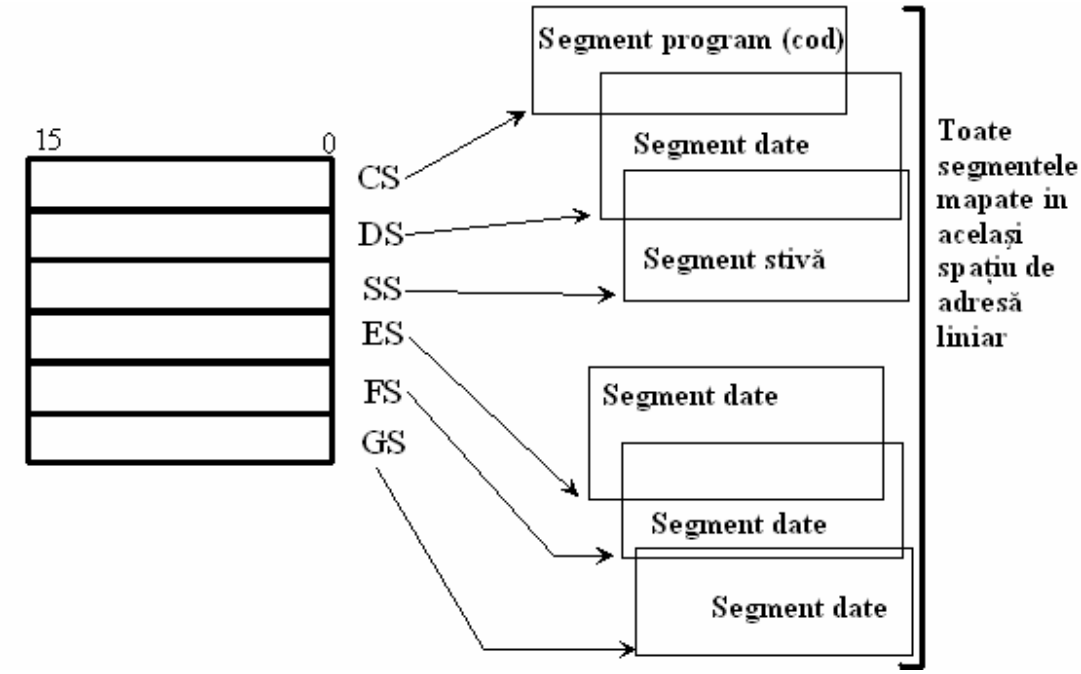


Figura 6

Când utilizăm **un model segmentat de memorie**, fiecare registru de segment este, în mod obişnuit, **încărcat cu un selector de segment diferit**, astfel că fiecare registru de segment indică un **segment diferit** în spațiul de adresă liniar. Astfel, oricând, un program poate accesa până la 6 segmente în spațiul liniar. Pentru a accesa un segment care nu este indicat de unul din cele 6 registre, programul trebuie mai întâi **să încarce selectorul de segment** pentru segmentul ce se doreşte accesat **în registrul de segment** corespunzător tipului de memorie.

4.3 Organizarea memoriei

Memoria adresată de unitatea centrală prin intermediul magistralei de adrese se numeşte **memorie fizică**. Memoria fizică este organizată ca o secvență de octeți (8 biți,

byte). Fiecare octet are asignată **o adresă unică** numită **adresă fizică**. Daca la primele procesoare 8086 spațiul de adrese fizice era de maxim 1MB, la ultimele procesoare spațiul de adrese fizice se întinde de la 0 la maxim 2^{36} -1 (64GB).

Practic, la ora actuală, orice sistem de operare proiectat să lucreze cu un procesor IA32 va folosi facilitățile oferite de **sistemul de gestiune al memoriei** (**Memory Management**). Printre caracteristicile acestuia se numără **segmentarea** şi **paginarea,** care permit ca memoria să fie gestionată eficient şi fiabil. În continuare vom prezenta metodele de bază pentru adresarea memoriei folosite de sistemul de gestiune al memoriei.

Când un program utilizează sistemul de gestiune al memoriei, **el nu va adresa direct memoria fizică**. Accesul la aceasta se va face folosind unul din cele trei modele de memorie posibile: "plat", segmentat sau cu adresă reală.

În cazul **modelului plat** memoria apare programatorului ca un singur spațiu de adrese continuu numit **spațiu de adrese liniar**. Codul (instrucțiunile), datele (variabilele) şi stivele pentru proceduri (subrutine) sunt toate conținute în acest spațiu. Spațiul liniar de adrese este adresabil la nivel de octet (byte), cu adrese succesive de la 0 la maxim 2^{n} -1. O adresă de octet în spațiul liniar este numită **adresă liniară**.

Pentru **modelul de memorie segmentat** memoria apare programului ca un grup de spații de adresă independente numite **segmente**. În acest model codul, datele şi stivele sunt, tipic, conținute în segmente separate. Pentru a adresa un octet dintr-un segment, programul utilizează o **adresă logică**, care constă dintr-un **selector de segment** şi un **offset (un decalaj-deplasament)**, descrierea fiind de forma **selector segment : offset**. Adresa logică mai este adesea asimilată unui **pointer** de tip **FAR** (îndepărtat). **Selectorul de segment** identifică segmentul care va fi accesat, iar **offsetul** identifică octetul adresat în spațiul de adresă al segmentului. Programele rulând pe o Arhitectură Intel 32 (IA32) pot accesa până la 16384 de segmente, de tipuri şi mărimi diferite şi fiecare segment poate avea până la2^{32} (4GB) sau 2^{36} octeți (64GB).

Intern, toate segmentele definite pentru un sistem (program) sunt mapate în spațiul de adrese liniar al procesorului. Astfel procesorul **translatează fiecare adresă logică într-o adresă liniară pentru a accesa o locație de memorie**. Această **translație este transparentă** pentru aplicație (program).

Principala justificare pentru modelul segmentat este creşterea fiabilității (siguranței în funcționare) a programelor şi a sistemului în ansamblu. De exemplu prin plasarea stivei programului într-un segment separat ne asigurăm că aceasta nu va creşte în spațiul de memorie al codului şi /sau datelor, suprascriind instrucțiunile sau variabilele. Plasând codul, datele şi stivele sistemului de operare în segmente separate le protejăm de programul aplicație şi vice-versa. Efectul final este cel de **izolare a efectului eventualelor greşeli de programare**. Pe scurt, în cazul unui model segmentat **nu este posibil**:

- să execuți operații de scriere în spațiul de memorie de program
- să execuți cod din spațiul de memorie de date
- să adresezi memorie dincolo de limitele segmentului

Valorificarea acestor caracteristici este asociată cu un mecanism adecvat de semnalizare a erorilor de această natură (Protection Fault). Prezența acestui mecanism şi necesitatea încărcării permanente a selectorilor de segment înseamnă un consum suplimentar de timp de calcul, deci penalități în viteza de execuție a codului!

În concluzie, atunci când se doreşte securitate maximă a aplicației, fără cerințe prea mari din punct de vedere al performanței este recomandat modelul segmentat propriu-zis. Când, în primul rând se doreşte performanță, ca viteză de execuție şi dimensiune a codului, este recomandat modelul plat sau liniar.

OBSERVAȚIE Indiferent de model (plat sau segmentat) arhitectura IA32 oferă şi facilități pentru a diviza spațiul liniar de adrese în **pagini** şi de a mapa aceste pagini într-o **memorie virtuală**. Dacă un sistem de operare foloseşte **mecanismul de paginare** al Arhitecturii Intel de 32 de biți, existența paginilor este transparentă pentru aplicație.

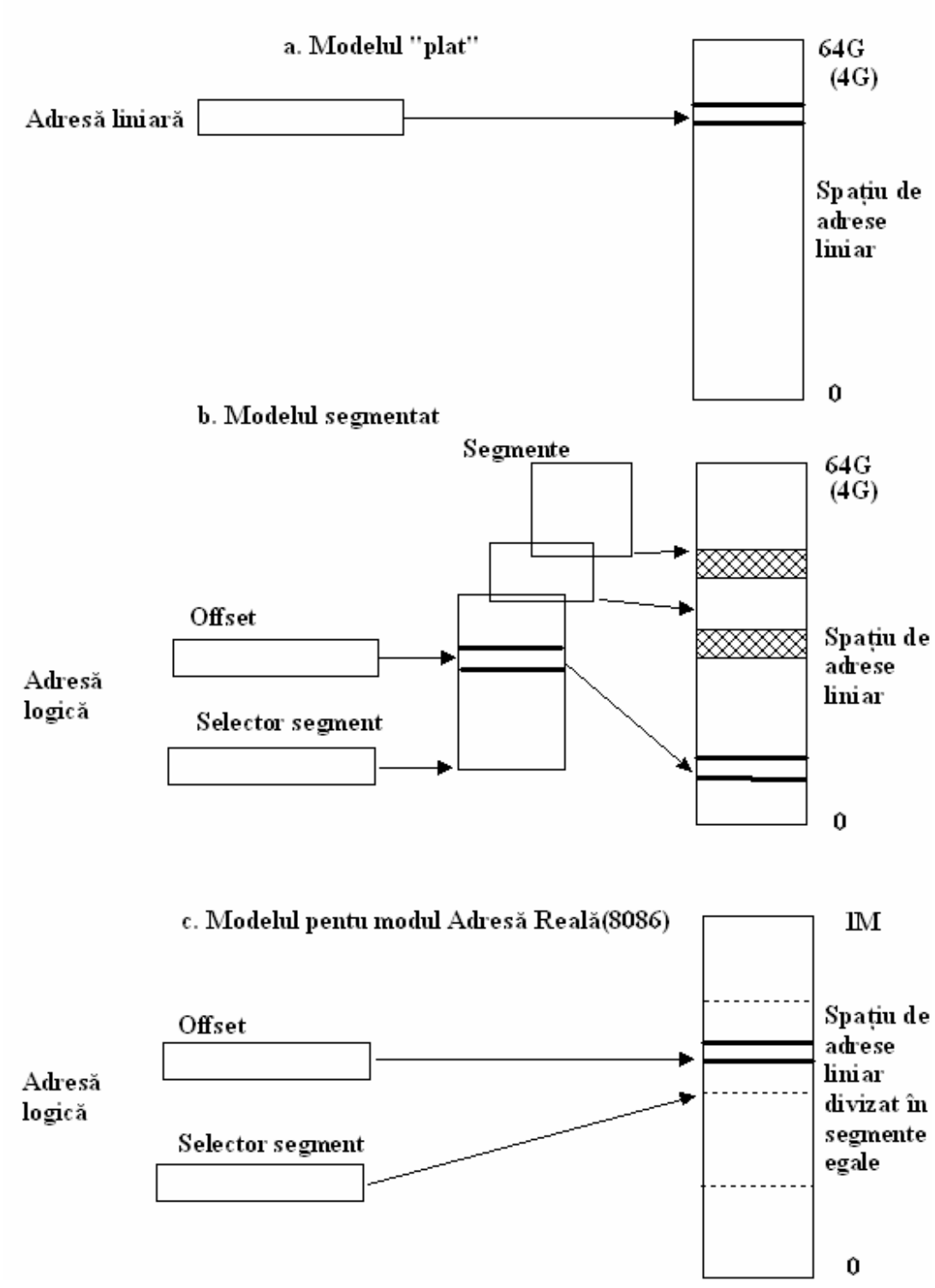


Figura 7

Modelul **mod adresă reală** este **modelul de memorie standard pentru vechile procesoare 8086** (8088, 80186, 80188). El este însă disponibil la toate variantele

ulterioare de procesoare pentru a asigura **compatibilitatea codului** vechilor programe scrise pentru procesoarele 8086. Modul adresă reală foloseşte **o implementare specifică a memoriei segmentate** în care spațiul liniar de adrese pentru program (aplicație) şi /sau sistemul de operare constă într-un **set de segmente fiecare având dimensiunea maximă de până la 64 Kocteți**. **Dimensiunea maximă a spațiului de adrese liniar este de 2 ^{20} (1024KB = 1MB)**.

Vom exemplifica în continuare modul în care este generată adresa fizică de memorie în cazul modelului mod adresă reală (8086/88, 80186/188) acest mod fiind şi cel mai simplu. Adresa fizică este generată prin însumarea (cu un sumator de 20 de biți) offsetului cu selectorul de segment, acesta din urmă fiind deplasat spre stânga cu 4 poziții înainte de însumare. Transportul este ignorat.

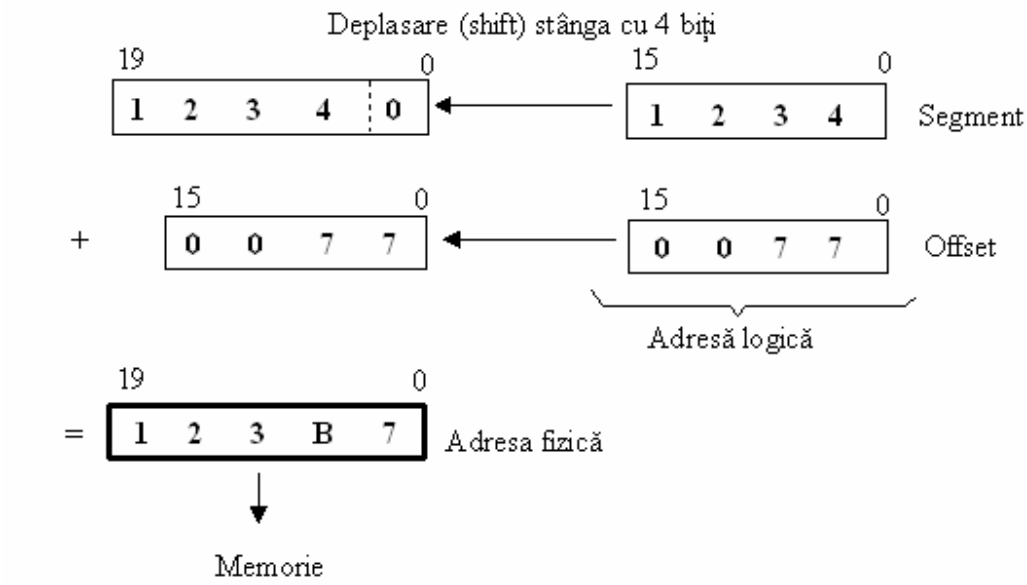


Figura 8

Este evident din exemplul de mai sus că, folosind valori diferite pentru selectorul de segment din CS şi offset-ul din IP, poate rezulta aceiaşi adresă fizică, cu alte cuvinte posibilitatea de suprapunere a segmentelor. De exemplu, adresa fizică 123B7 poate fi obținută nu numai plecând de la perechea 1234 : 0077 ci şi de la perechea 1230: 00B7, etc.

În descrierea care urmează, vom prezenta sumar şi modul în care, **mecanismul de paginare** pe care l-am menționat anterior, este integrat cu modelul segmentat.

În cazul modelului segmentat propriu-zis, o unitate dedicată a procesorului numită **unitate de segmentare** translatează adresa logică din program într-o adresă liniară. Locațiile segmentelor în spațiul de adresă liniar sunt memorate în structurile de date numite **descriptori de segment**.

Unitatea de segmentare calculează adresa folosind descriptorii de segment şi offsetul, extras din instrucţiune. Adresa liniară este trimisă la **o unitate de paginare** şi la unităţile de cache.

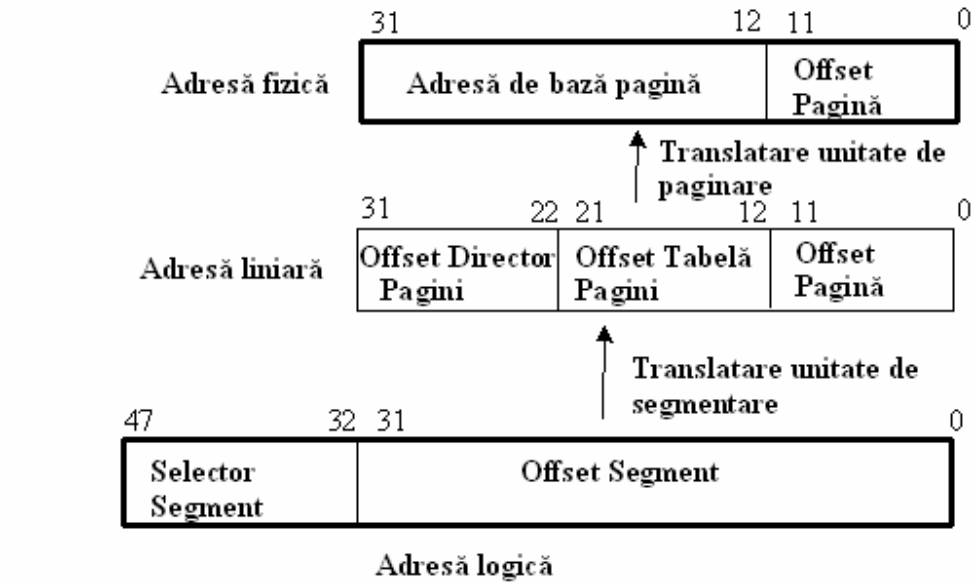


Figura 9

Când un segment este adresat pentru prima oară, descriptorul său de segment este memorat într-un registru al procesorului. Un program poate avea până la 16384 de segmente. La un moment dat pot fi memoraţi în registre ale procesorului până la 6 descriptori de segment. În figură sunt prezentate relaţiile între adresa logică, cea liniară şi cea fizică.

Unitatea de paginare permite accesarea unor structuri de date mai mari decât memoria fizică (RAM) disponibilă, asigurând păstrarea acestora parţial în memorie şi parţial pe un suport extern (hard disk). Paginarea împarte spaţiul de adrese liniar în **blocuri de 4KB numite pagini**. Ea foloseşte structuri de date particulare (în memorie) numite **tabele de pagini** pentru maparea spaţului liniar de adrese în spaţiul fizic. Adresa fizică este plasată pe magistrala de adrese a procesorului prin intermediul unităţii de cache care o utilizează şi ea.

Unitatea de paginare identifică si problemele care pot apărea, cum ar fi un acces la o pagină care nu există în memorie, generând un eveniment numit eroare de pagină ("page fault"). Într-o astfel de situaţie sistemul de operare poate încerca să aducă pagina respectivă de pe suportul extern (hard disk). Dacă este necesar poate şi elibera spaţiu de memorie trimitând altă pagină pe disk. **Dacă paginarea nu este activă, adresa fizică este identică cu adresa liniară**.

Spre deosebire de segmentare, paginarea este invizibilă (transparentă) pentru aplicaţii, dar nu furnizează o protecţie similară acesteia. Paginarea este vizibilă doar pentru sistemul de operare care o foloseşte pentru a satisface cerinţele de memorie ale aplicaţiilor.

4.4 Modurile de funcționare (de operare) pentru arhitectura IA32

Arhitectura Intel, începând cu 80386(IA-32), este caracterizată prin existența a trei moduri de operare (funcționare): protejat, adresă reală şi gestiune sistem. **Modul de operare determină ce instrucțiuni şi caracteristici de lucru sunt disponibile, precum şi modelele de memorie disponibile.**

a. **Modul protejat (sau mod adresă virtuală protejată)** este starea nativă a unui procesor IA32. În acest mod toate instrucțiunile şi caracteristicile de prelucrare sunt disponibile, permițând obținerea performanțelor maxime. Oricare din modelele de memorie descrise poate fi utilizat. Modelul de memorie utilizat depinde de modul de proiectare al sistemului de operare. Când este implementat un sistem de operare multitasking, fiecare task poate utiliza un model de memorie diferit. Este modul recomandat pentru dezvoltarea tuturor noilor aplicații sau sisteme de operare. Printre caracteristici se numără şi aceea de a executa direct codul scris în "modul adresă reală" pentru 8086 într-un context protejat şi multi-tasking. Această caracteristică se numeşte **mod virtual 8086**, deşi nu este un mod de funcționare propriu-zis. El este mai degrabă un atribut al modului de lucru protejat care poate fi activat pentru orice task.

b. **Modul adresă reală (sau mod real)** asigură un context de programare al procesorului 8086 cu câteva mici extensii, cum ar fi posibilitatea comutării în mod protejat sau în mod gestiune sistem. Se poate utiliza doar modelul de memorie mod adresă reală. **Orice procesor 80x86 este plasat în acest mod la reset (inițializare)** sau power -up reset.

c. **Modul gestiune sistem (system management)** reprezintă o caracteristică unică a procesoarelor Intel, apărută începând cu 386SL.

Acest mod asigură sistemului de operare o modalitate transparentă de a implementa funcții specifice platformei de calcul, cum ar fi gestiunea puterii consumate (power management) sau securitatea sistemului. Trecerea în acest mod se face sub controlul unui semnal extern (o cerere de întrerupere externă dedicată).

La trecerea în acest mod, procesorul comută pe un spațiu de adrese separat numit SMRAM (System Management RAM) similar modelului de la modul adresă reală. În acealaşi timp se salvează întregul context al programului care rulează sau al task-ului curent. Codul specific acestui mod poate fi apoi executat în mod transparent. La revenirea din acest mod procesorul îşi restaurează contextul avut înainte de intrarea în acest mod.

4.5 Dimensiunea adreselor şi operanzilor: 16 sau 32 de biți

Un procesor cu Arhitectura Intel IA32 poate fi configurat să funcționeze cu dimensiuni ale adreselor şi operanzilor de 32 de biți sau de 16 biți .

La **funcționarea în modul protejat**, descriptorul de segment pentru segmentul de cod curent executat defineşte (prin intermediul unor valori de bit) dimensiunea implicită a adresei şi a operanzilor. Un **descriptor de segment** este o structură de date asociată sistemului de operare care în mod normal nu este vizibilă pentru aplicație.

Există directive ale asamblorului care permit alegerea dimensiunii implicite a adresei şi operanzilor, generându-se apoi corespunzător descriptorul pentru segmentul de cod care se va executa.

Utilizarea a două prefixe speciale pentru instrucţiuni poate modifica temporar dimensiunea implicită a adreselor şi respectiv a operanzilor într-un program (cea dată de descriptorul de segment). Este astfel posibilă combinarea codului de 16 biţi cu cel de 32 de biţi în cadrul aceluiaşi segment de cod.

Cu adrese si operanzi de 32 de biţi, adresa liniară maximă sau offsetul maxim este FFFF FFFF H (2^{32}), iar dimensiunile operanzilor sunt de 8 sau 32 de biţi.

În varianta cu adrese şi operanzi de 16 de biţi, adresa liniară maximă sau offsetul maxim este FFFF H (2^{16}), iar dimensiunile operanzilor sunt de 8 sau 16 de biţi.

Când se foloseşte o adresare pe 32 de biţi, o adresă logică (sau un pointer FAR) constă dintr-un selector de segment de 16 biţi şi un offset de 32 de biţi; pentru o adresare pe 16 biţi selectorul are tot 16 biţi, dar offsetul numai 16 biţi.

La operarea în mod adresă reală dimensiunea implicită pentru adrese şi operanzi este de 16 biţi. Se poate forţa activarea şi utilizarea unei adrese de 32 de biţi, dar valoarea maximă utilizabilă a adresei este de 0000 FFFF H (tot 2^{16}).

4.6 Tipuri fundamentale de date

Modalitatea de **ordonare a octeţilor dintr-o structură multi-octet** este importantă pentru orice microprocesor sau limbaj de programare. În acest context există două variante:

a. **Little Endian**: cel mai puţin semnificativ octet (LSB-Least Significant Byte) este memorat la cea mai mică adresă, adresă care devine şi adresa structurii multi-octet

b. **Big Endian**: cel mai semnificativ octet (MSB-Most Significant Byte)este memorat la cea mai mică adresă, adresă care devine şi adresa structurii multi-octet

La **toate procesoarele 80x86 (de fapt Intel) cel mai puţin semnificativ octet ocupă cea mai mică adresă în memorie (Low), adresa respectivă fiind şi adresa operandului,** deci se foloseşte o reprezentare **Little Endian**.

OBSERVAŢIE La marea majoritate a microprocesoarelor **Freescale** (ex. **Motorola)** se foloseşte o reprezentare **Big Endian**. In cazul arhitecturii PowerPC se pot folosi ambele reprezentări.

Tipurile fundamentale de date pentru Arhitectura Intel 32 sunt : **octet -byte** (8 biţi), **cuvânt - word** (2 octeţi - 16 biţi), **dublu cuvânt -double word** (4 octeţi - 32 de biţi), **quadruplu cuvânt -quad word** (8 octeţi - 64 de biţi) şi **dublu cvadruplu cuvânt - double quad word** (16 octeţi - 128 de biţi). Este evident din această înşiruire care sunt structurile de tip multi-octet.

OBSERVAŢIE Deşi nu reprezintă un tip de dată propriu-zis, se mai utilizează şi grupul de patru biţi numit **nibble**, cu două particularizări mai întâlnite: cifra (digitul) BCD şi cifra hex.

Un subset al instrucțiunilor IA-32 operează cu aceste tipuri fundamentale fără utilizarea unor definiri suplimentare de tip.

Tipul **quad word** a apărut în Arhitectura Intel odată cu 80486, iar **double quad word** odată cu Pentium III cu extensia SSE.

În figura următoare este prezentată **ordinea octeților** pentru aceste tipuri de date, atunci când sunt folosite ca operanzi din memorie.

Deşi tipurile fundamentale de date pentru o Arhitectură Intel sunt cele menționate mai sus, **anumite instrucțiuni suportă o interpretare suplimentară a acestor tipuri pentru a permite operații asupra unor tipuri numerice de date: întregii cu sau fără semn precum şi numerele în virgulă mobilă**.

Astfel, întregii fără semn (**unsigned integer**) au formatul de reprezentare nativ, coincid cu tipurile fundamentale. La întregii cu semn (**signed integer**), cel mai semnificativ bit al tipului respectiv, este bitul de semn (de exemplu bitul 15 la signed word sau bitul 63 la signed quadword), valoarea fiind reprezentată în cod binar complement față de 2 .

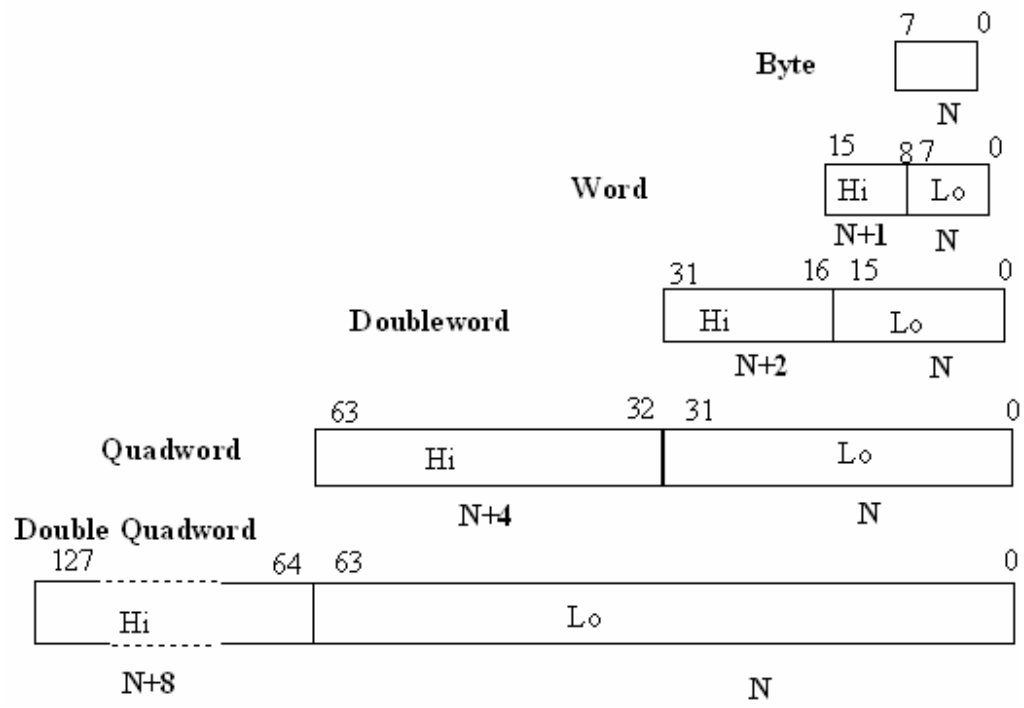


Figura 10

La **formatul virgulă mobilă** (floating point) lucrurile sunt mai complicate. Reprezentarea este conformă standardului **IEEE 754**, existând trei formate: **simplă precizie** - single precision (lungime 32 biți, precizie 24 biți), **dublă precizie** - double precision (lungime 64 biți, precizie 53 de biți) şi **dublă precizie extinsă**- extended double precision (lungime 80 de biți, precizie 64 de biți). Precizia se referă la numărul de biți al mantisei normalizate.

În afară de aceste tipuri numerice mai există şi alte tipuri de date utilizate de unele instrucțiuni: **pointeri** de tip **FAR** (adresă logică) sau **NEAR** (offsetul), **câmpuri de biți -**

bit field (o secvență continuă de biți conținând până la 32 de biți), **şiruri de biți sau octeți - bit or byte strings** (secvențe de biți sau octeți conținând de la 0 până la $2^{32} - 1$ elemente).

Un tip particular de dată, care se întâlneşte şi într-un alt context, este **NAN** (Not A Number) cum ar putea fi caracterizat, de exemplu, rezultatul unei împărțiri la 0.

Mai există şi alte tipuri dedicate de date asociate setului de instrucțiuni (extensiei) MMX sau SIMD, atunci când ele sunt prezente.

4.7 Alinierea în memorie a tipurilor fundamentale de date

Problema alinierii datelor apare deoarece nu este posibil să scriem un program de dimensiuni rezonabile folosind eficient doar un tip de dată, având una şi aceiaşi dimensiune. De regulă vom utiliza tipuri de date de dimensiuni diferite.

Alinierea în memorie a tipurilor menționate de date, de dimensiuni diferite, nu este necesar să fie făcută conform granițelor naturale care rezultă din numărul de octeți asociat. **Granițele naturale** pentru word, double word şi quadword sunt adrese pare, adrese pare divizibile cu 4 şi respectiv adrese pare divizibile cu 8.

Totuşi, **pentru a îmbunătăți performanțele programelor, structurile de date (în special cele de natura unor stive) trebuie aliniate la granițele naturale atunci când este posibil**. Motivul este că **procesorul are nevoie de două cicluri de acces la memorie pentru date nealiniate şi numai de unul pentru date aliniate**. Un operand de tip word care depăşeşte granița de 4 octeți sau double word care depăşeşte granița de 8 octeți este considerat nealiniat şi are nevoie de două cicluri separate de memorie pentru a fi accesat. Un operand de tip word care începe la o adresă impară, dar nu depăşeşte granița de 2 octeți este considerat aliniat şi poate fi încă accesat într-un singur ciclu.

Anumite instrucțiuni care operează pe date de tip double quad word necesită ca operanzii să fie aliniați la granița naturală (orice adresă pară divizibilă cu 16). Aceste instrucțiuni vor genera o eroare de protecție (o excepție) dacă se specifică un operand nealiniat. Alte instrucțiuni care operează pe date de acelaşi tip permit accesul nealiniat, fără a genera eroare, dar pe seama execuției unor cicluri de acces suplimentare față de datele aliniate.

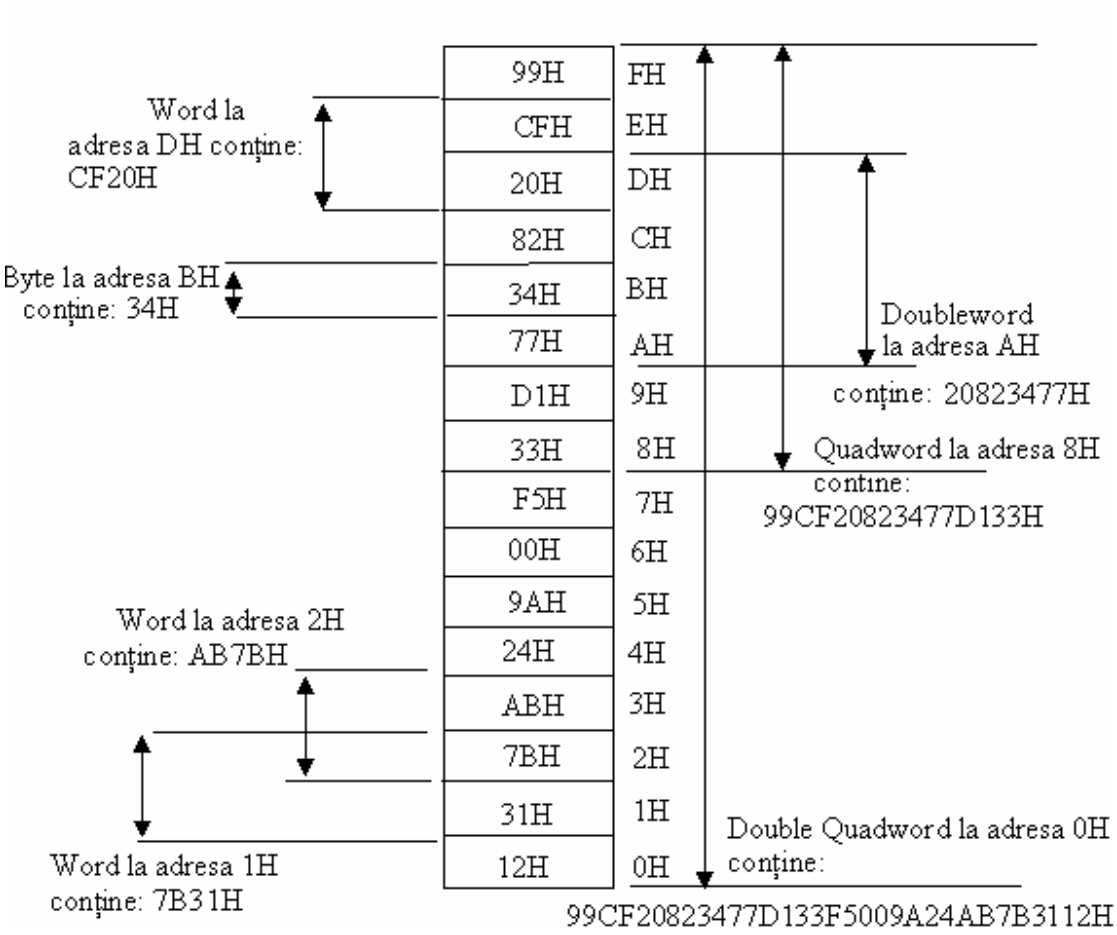


Figura 11

In concluzie, **nealinierea datelor înseamnă, de cele mai multe ori, penalități din punct de vedere al vitezei de execuție a codului**.

5. Arhitectura procesoarelor 80x86

5.1 Câteva instrucțiuni de bază

Setul de instrucțiuni al unui procesor 80x86 poate fi văzut ca alcătuit din peste 100 de instrucțiuni sau din câteva mii, aceasta depinzând de modul în care definim o instrucțiune mașină și varianta de procesor în cauză. Chiar și peste 100 de instrucțiuni par a fi totuși prea multe, dacă trebuie învățate într-o perioadă scurtă de timp. Din fericire pentru a începe să scriem un program în limbaj de asamblare nu este nevoie să le cunoaștem pe toate. Mai mult decât atât, s-a arătat că aproape în orice limbaj de asamblare se poate scrie un program care să realizeze ceea ce dorim folosind doar circa

30 de instrucțiuni diferite. Vom prezenta astfel în continuare câteva astfel de instrucțiuni esențiale.

Cea mai utilizată instrucțiune este cea având mnemonica **MOV**e (mută, pune). Într-un program tipic, între 25 % şi 40% din instrucțiuni sunt de acest tip. Aşa cum numele nu o sugerează, ea copiază de fapt datele dintr-o locație (sursă) într-alta (destinația), locația sursă rămânând nemodificată.

Sintaxa ei este: **MOV operand_destinație, operand_sursă** .

Operand_sursă poate fi un registru, o locație de memorie, o constantă (tot o locație de memorie). Operand_destinație poate fi un registru sau o locație de memorie. Setul de instrucțiuni **80x86 nu permite ca ambii operanzi să fie variabile(locații) în memorie.**

Ea este echivalentă cu următoarea asignare: **operand_destinație = operand_sursă**;

OBSERVAȚIE Principial vorbind instrucțiunea MOV **copiază o dată dintr-o locație într-alta**, astfel că normal ar fi fost să se numească COPY! Din păcate este prea târziu pentru asta şi trebuie să ne împăcăm cu această realitate.

Cea mai importantă restricție pentru această instrucțiune (şi nu numai pentru ea !) **este că ambii operanzi trebuie să aibă aceiaşi dimensiune**. Se poate executa un MOV între două obiecte de 8 biti, între două obiecte de 16 biți sau între două de 32 de biți, dar nu se pot amesteca dimensiunile operanzilor.

Operanzi legali pentru instrucțiunea MOV (80x86)

Sursă	**Destinație**	**Sursă**	**Destinație**
Reg 8	Reg 8	constantă	Reg 32
Reg 8	Mem 8	constantă	Mem 32
Mem 8	Reg 8		
constantă	Reg 8		
constantă	Mem 8		
Reg 16	Reg 16		
Reg 16	Mem 16		
Mem 16	Reg 16		
constantă	Reg 16		
constantă	Mem 16		
Reg 32	Reg 32		
Reg 32	Mem 32		
Mem 32	Reg 32		

OBSERVATIE

1. Reg8 - registru 8 biți ş.a.m.d.; Mem8- locație de memorie 8biți (octet, byte) ş.a.m.d.
2. Dimensiunea constantei trebuie să fie mai mică sau egală cu cea a destinației.

Instrucțiunile **ADD** şi **SUB** permit adunarea şi scăderea a doi operanzi. Sintaxa lor este asemănătoare instrucțiunii MOV:

ADD operand_destinație, operand_sursă .
SUB operand_destinație, operand_sursă .

Efectul instrucțiunii ADD este:
operand_destinație = operand_destinație + **operand_sursă**;
iar cel al unei instrucțiuni SUB :
operand_destinație = operand_destinație - operand_sursă;
Sau folosind o sintaxă similară limbajului C :
operand_destinație + = operand_sursă;
operand_destinație - = operand_sursă;
Se aplică aceleaşi restricții operanzilor ca la instrucțiunea MOV (vezi tabelă).

5.2 Execuția instrucțiunilor 80x86

Pentru a înțelege cel puțin o parte din problemele proiectării unei unități centrale eficiente, vom considera 4 instrucțiuni 80x86 reprezentative: **MOV, ADD, LOOP** şi **JNZ** (Jump on Not Zero -salt dacă diferit de zero). Pentru că MOV şi ADD au fost prezentate anterior ne vom concentra asupra noilor instrucțiuni **LOOP** şi **JNZ**. Ambele instrucțiuni sunt **instrucțiuni de salt condiționat sau condițional**.

Un salt condiționat testează unul din indicatorii de condiție şi realizează un salt la o altă instrucțiune din memorie (alta decât cea următoare) dacă este adevărată condiția, în caz contrar executându-se instrucțiunea următoare. Este ceva similar construcției IF ... THEN dintr-un limbaj evoluat.

JNZ testează starea flagului (indicatorului de condiție) **Zero** din EFLAGS (sau PSW). Se execută saltul dacă flagul respectiv este 0 (condiția testată NZ este adevărată) sau instrucțiunea următoare dacă flagul este 1 (condiția testată NZ este falsă). Programul specifică instrucțiunea de destinație a saltului prin **"distanța" dintre instrucțiunea JNZ şi destinație, exprimată ca un întreg cu semn**. Pentru simplitate vom presupune că această distanță este în domeniul -128..+127, pentru a putea fi exprimată (ca întreg cu semn) pe un singur octet.

LOOP decrementează valoarea registrului ECX şi **transferă controlul** unei instrucțiuni țintă (destinație), tot în domeniul -128 ..+127 față de instrucțiunea LOOP, **dacă după decrementare ECX este diferit de zero**. Ea ar fi echivalentă cu următoarele două instrucțiuni 80x86:

LOOP *Eticheta* ; = = DEC ECX; JNZ *Eticheta* ;
OBS. *Eticheta* trebuie să fie în domeniul -128 ..+127 față de LOOP sau JNZ. Instrucțiunea DEC (DECrement) este similară SUB cu operandul sursă = 1.

Este un exemplu de instrucțiune CISC (comparativ cu RISC), care efectuează mai multe operații în acelaşi timp: 1. decrementează pe ECX 2. execută un salt condițional dacă ECX nu este zero. În realitate însă un DEC şi un JNZ se execută mai rapid decât un LOOP. Tot pentru simplitate vom presupune că ea se execută asemenea unei instrucțiuni RISC .

Instrucțiunile 80x86 nu se execută într-un singur ciclu de ceas al procesorului. De exemplu instrucțiunea **MOV**, care este relativ simplă, ar presupune următorii paşi de execuție:
- preia octetul cu codul instrucțiunii (fetch opcod) din memorie

- actualizează EIP pentru a indica următorul octet
- decodifică instrucțiunea pentru a vedea ce face
- dacă e necesar, preia (fetch) operand de 16 biți din memorie
- dacă e necesar, actualizează EIP pentru a indica dincolo de operand
- dacă e necesar, calculează adresa operandului (ex. EBX+deplasament)
- preia (fetch) operandul
- memorează valoarea preluată într-un registru

Dacă alocăm câte un ciclu de ceas pentru fiecare din paşii de mai sus, o instrucțiune poate dura de la 5 până la 8 cicluri de ceas (trei din paşii de mai sus sunt opționali depinzând de modul de adresare al operanzilor).

Instrucțiunea **ADD** este mai complexă ea presupunând următorii paşi (un ADD reg, reg):
- preia octetul cu codul instrucțiunii (fetch opcod) din memorie
- actualizează EIP pentru a indica următorul octet
- decodifică instrucțiunea
- ia valoarea operandului sursă (registru) şi o trimite la ALU
- ia valoarea operandului destinație (registru) şi o trimite la ALU
- comandă ALU pentru adunare
- memorează rezultatul înapoi în primul operand (registru)
- actualizează indicatorii de condiție conform rezultatului operației de adunare

OBS. ALU (Arithmetic Logic Unit) - Unitatea Aritmetică şi Logică este o componentă esențială a oricărui CPU, la nivelul ei executându-se toate operațiile aritmetice şi logice.

Mai mult dacă **operandul sursă este în memorie** (ADD reg, mem), secvența este mai complicată:
- preia octetul cu codul instrucțiunii (fetch opcod) din memorie
- actualizează EIP pentru a indica următorul octet
- decodifică instrucțiunea
- dacă e necesar, preia un deplasament pentru a-l utiliza în calculul adresei efective
- dacă e necesar, actualizează EIP pentru a indica dincolo de valoarea deplasamentului
- ia valoarea operandului sursă din memorie şi o trimite la ALU
- ia valoarea operandului destinație (registru) şi o trimite la ALU
- comandă ALU pentru adunare
- memorează rezultatul înapoi în primul operand (registru)
- actualizează indicatorii de condiție conform rezultatului operației de adunare

Cea mai complicată secvență avem când sursa este o constantă (ADD mem, const) :
- preia octetul cu codul instrucțiunii (fetch opcod) din memorie
- actualizează EIP pentru a indica următorul octet
- decodifică instrucțiunea
- dacă e necesar, preia un deplasament pentru a-l utiliza în calculul adresei efective
- dacă e necesar, actualizează EIP pentru a indica dincolo de valoarea deplasamentului
- preia valoarea constantă din memorie şi o trimite la ALU
- actualizează EIP pentru a indica în memorie dincolo de valoarea constantei
- ia valoarea operandului destinație din memorie şi o trimite la ALU

- comandă ALU pentru adunare
- memorează rezultatul înapoi în primul operand (memorie)
- actualizează indicatorii de condiție conform rezultatului operației de adunare

Este de menționat că sunt şi alte forme ale instrucțiunii ADD cu secvența lor proprie de faze, dar cele prezentate sunt cele mai semnificative. Aşa cum se vede din exemple pot fi necesare până la 11 faze pentru finalizarea instrucțiunii. Aici este şi avantajul conceptului RISC, marea majoritate a procesoarelor RISC având una sau două forme de ADD (registru-registru şi poate constantă - registru). De asemenea se vede că din punct de vedere al timpului de execuție varianta ADD reg, reg este cea mai avantajoasă.

O instrucțiune **JNZ** ar presupune următoarea secvență de faze :
- preia octetul cu codul instrucțiunii (fetch opcod) din memorie
- actualizează EIP pentru a indica următorul octet
- decodifică instrucțiunea
- preia octetul de deplasament pentru a determina distanța de salt şi-l trimite la ALU
- actualizează EIP pentru a indica următorul octet
- testează flagul Zero pentru a vedea dacă este 0
- dacă Zero = 0, atunci trimite EIP la ALU
- dacă Zero = 0, comandă ALU să adune deplasamentul cu EIP
- dacă Zero = 0, copiază rezultatul adunării înapoi în EIP

Se observă că instrucțiunea JNZ **necesită mai puțini paşi atunci când condiția de salt nu este îndeplinită. Aceasta este tipic pentru toate instrucțiunile de salt condiționat**. Dacă fiecare pas va corespunde unui ciclu de ceas, un JNZ va dura între 6 şi 9 cicluri de ceas, funcție de efectuarea sau nu a saltului propriu-zis. Pentru că JNZ nu permite tipuri diferite de operanzi există o singură secvență de paşi (spre deosebire de ADD).

Instrucțiunea **LOOP** poate folosi o secvență cum ar fi :
- preia octetul cu codul instrucțiunii (fetch opcod) din memorie
- actualizează EIP pentru a indica următorul octet
- decodifică instrucțiunea
- preia valoarea lui ECX şi o trimite la ALU
- comandă ALU să decrementeze această valoare (ECX)
- trimite rezultatul înapoi în ECX şi setează un flag special intern dacă valoarea nu este 0
- preia octetul de deplasament pentru a determina distanța de salt şi-l trimite la ALU
- actualizează EIP pentru a indica următorul octet
- testează flagul special pentru a vedea dacă ECX a fost diferit de zero
- dacă flagul este setat, trimite (copiază) EIP la ALU
- dacă flagul este setat, comandă ALU să adune deplasamentul cu EIP
- dacă flagul este setat, copiază rezultatul adunării înapoi în registrul EIP

Deşi un anume tip de procesor din familia 80x86 poate să nu execute paşii concreți menționați mai sus, toate instrucțiunile presupun executarea unei anume secvențe de operații. Fiecare operație necesită un anumit timp pentru execuție, în general **un ciclu de ceas per operație** sau **fază** cum vom mai numi paşii de mai sus. Este

evident că, odată cu numărul de paşi (faze) creşte şi timpul de execuţie al instrucţiunii. Acesta este motivul pentru care, în general, instrucţiunile complexe se execută mai lent decât instrucţiunile simple.