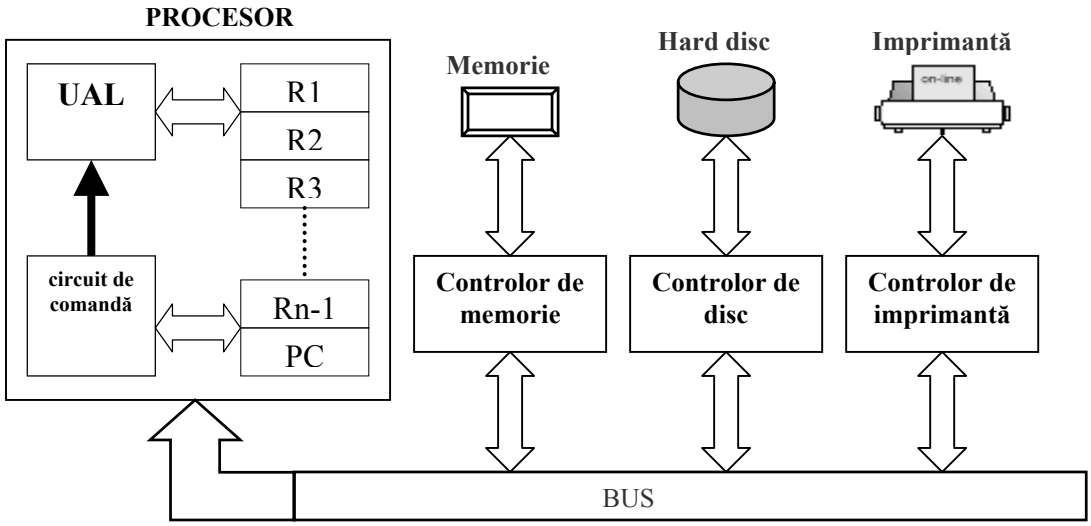
Organizarea şi accesarea regiştrilor

Cele mai vizibile şi accesibile componente ale unui microprocesor, din punctul de vedere al unui programator, sunt regiştrii acestuia. Pentru a putea prelucra instrucţiunile unui program procesorul trebuie să conţină un tip de memorie în care să aducă instrucţiunile şi datele din memoria principală. Acest tip de memorie poartă denumirea de regiştrii microprocesorului. Aceştia sunt formaţi din 8, 16, 32 sau 64 de biţi şi sunt întrebuinţaţi la diferite operaţii. Fiecare registru este specializat, adică folosit de procesor într-un scop bine determinat. Registrele sunt împărţite în două mari categorii: registre de uz general şi registre speciale. Registrele de uz general sunt folosite pentru a memora variabilele locale cât şi rezultatele intermediare ale calculelor.

În continuare vom discuta despre procesoarele 8088/8086, 80188/80186, 80286, şi 80386/80486/80586/Pentium, precum şi despre procesoarele de tip RISC, specificând în fiecare context despre care tip de microprocesor este vorba.

În istoria familiei de microprocesoare Intel, fiecare generaţie de microprocesoare a moştenit structura celei dinainte îmbunătăţind-o. De exemplu familiile de microprocesoare Intel 8088/8086, 80188/80186, 80286 au avut regiştrii de 8-16 biţi iar familiile de microprocesoare Intel 80386/80486/80586/Pentium au avut sau au regiştrii extinşi de până la 32 de biţi.

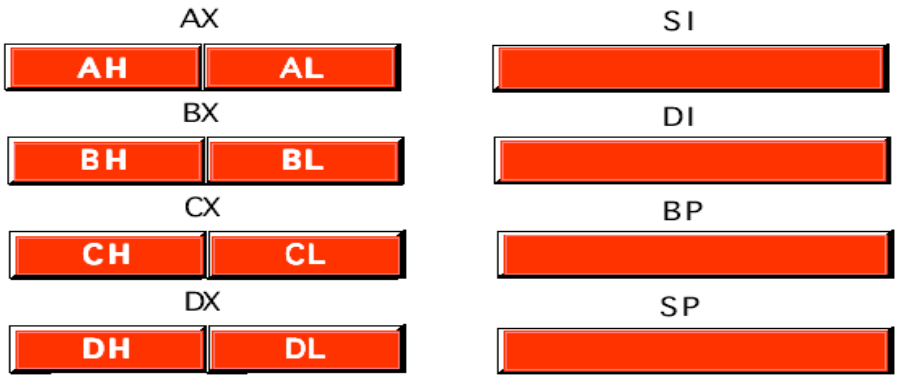
Mai jos este prezentată structura generală a unui calculator împreună cu legăturile



interne şi externe ale microprocesorului.

Proiectanţii microprocesoarelor Intel au clasificat regiştrii în trei categorii: regiştrii de

uz general care sunt folosiți pentru a memora variabilele locale cât şi rezultatele intermediare ale calculelor, regiştrii segment cu ajutorul cărora se pot accesa blocuri de memorie principală numite segmente, şi doi regiştrii speciali.



regiştrii de uz general la 8086

Regiştrii de uz general

Procesoarele 8086 au următorii 8 regiştrii de uz general de 16 biți: ax, bx, cx, dx, si, di, bp şi sp. Fiecare dintre ei pot fi folosiți pentru orice operație posibilă, dar sunt folosiți de către programatori în următoarele scopuri:

- AX (*Accumulator Register*) este registrul *acumulator* şi este principalul registru aritmetic, în el făcându-se majoritatea calculelor aritmetice şi logice.
- BX (*Base Register*) este folosit la memorarea adreselor de memorie (accesarea indirectă a memoriei).
- CX (*Count Register*) joacă rol de contor în bucle, adică contorizează numărul de iterații a unei instrucțiuni de ciclare LOOP sau specifică numărul de caractere dintr-un şir.
- DX (*Data Register*) are două întrebuințări speciale: se foloseşte la împărțire şi înmulțire şi împreună cu AX conține produsul, respectiv deâmpărțitul pe 16 de biți şi se mai foloseşte la memorarea adreselor de I/O atunci când se accesează date de pe magistrala de I/O.
- SI şi DI (*Source Index şi Destination Index*) sunt folosiți la fel ca registrul BX pentru accesarea indirectă a memoriei şi mai sunt folosiți împreună la prelucrarea şirurilor, cu SI specificând şirul sursă şi DI şirul destinație.

- BP (*Base Pointer*)este indicator de bază în cadrul local de stivă adică este folosit pentru accesarea parametrilor şi a variabilelor locale dintr-o procedură.
- SP (*Stack Pointer*) indicator de stivă, conţine adresa de memorie a stivei programului în curs de execuţie. Acest registru nu trebuie întrebuinţat în alte scopuri.

Regiştrii ax, bx, cx, dx pot fi folosiţi şi la jumătate din capacitate, adică pot fi împărţiţi în câte doi regiştrii de câte 8 biţi: AH, Al, BH, BL, CH, CL, DH, DL (H – High byte, L – Low byte).

Regiştrii de segment

Procesorul 8086 are patru *regiştrii de segment*: cs (Code Segment), ds(Data Segment), es(Extra Segment), şi ss(Stack Segment). Aceştia au dimensiunea de 16 biţi şi sunt folosiţi pentru selectarea blocurilor(segmentelor) de memorie din memoria principală.

- De exemplu CS registru segment de program şi conţine adresa începutului blocului (segmentului) de memorie ce conţine programul principal.
- DS registru segment de date şi conţine adresa variabilelor globale ale programului.
- ES registru segment suplimentar de date şi este folosit în cazul în care nu se pot modifica ceilalţi regiştrii.
- SS registru segment de stivă şi conţine adresa de bază (început) a stivei. Stiva este o parte din memorie în care procesorul memorează adresa de întoarcere dintr-o procedură (subrutină), parametrii procedurilor şi variabile locale.

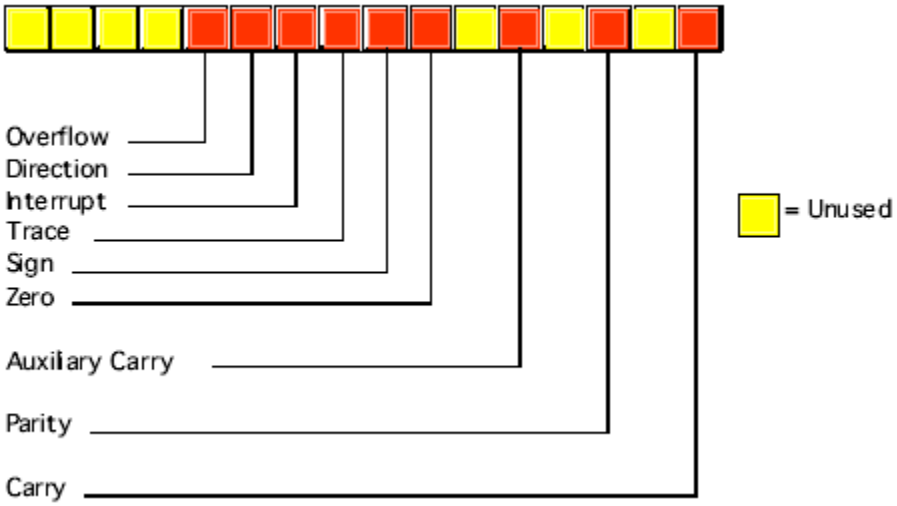
Nu este indicat să se modifice la întâmplare conţinutul regiştrilor segment deoarece aceştia conţin adresele segmentelor de memorie folosite în program.

Regiştrii speciali

Există doi regiştrii speciali:

- instruction pointer (ip) registru pe 16 biţi care conţine adresa instrucţiunii curente.
- Registrul fanion (flags register) are 16 biţi şi este un colector de informaţii de un bit ce reprezintă stările procesorului. La descrierea instrucţiunilor microprocesorului se va specifica la fiecare instrucţiune care bit este afectat şi în ce mod.

Programatorul nu are acces la aceşti regiştrii, ei pot fi modificaţi automat de către microprocesor.

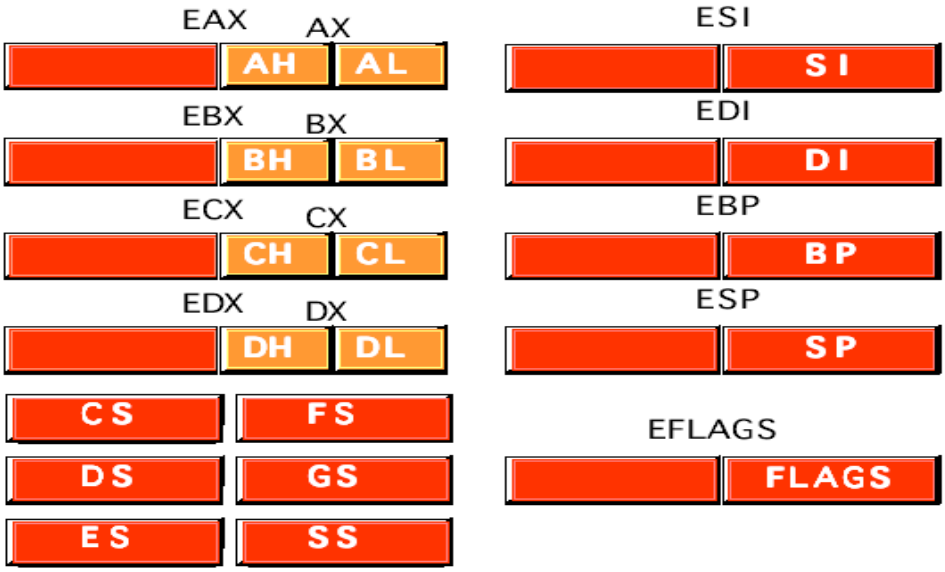


Registrul fanion (flags register)

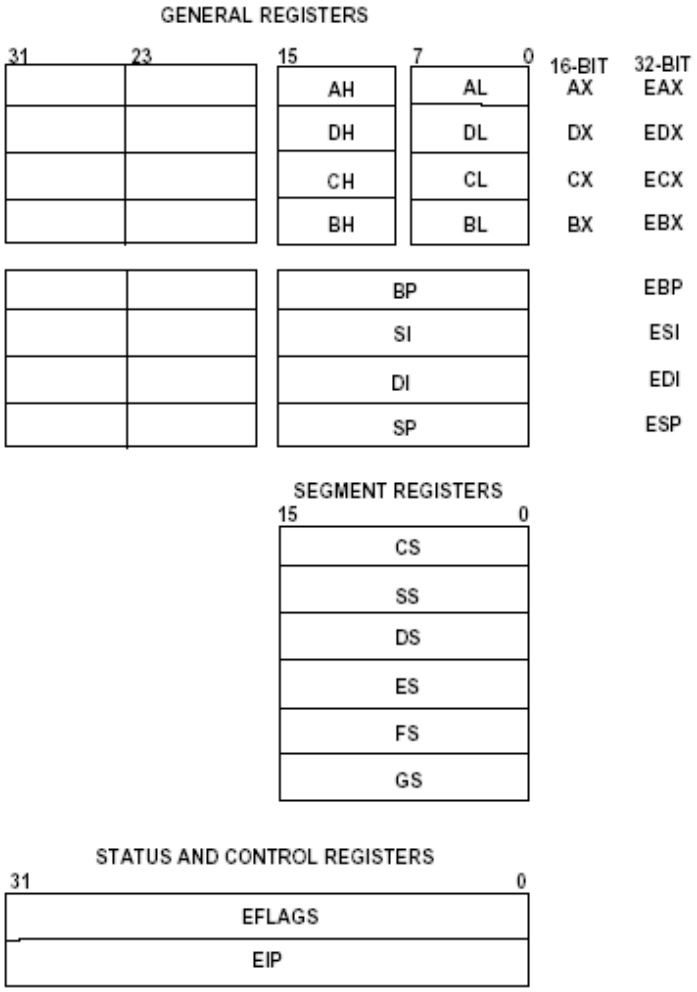
Familiile de microprocesoare Intel 80386/80486/80586/Pentium au următoarele modificări față de microprocesorul 8086:

(a) regiştrii extinşi până la 32 de biți astfel: regiştrii ax, bx, cx, dx, si, di, bp, sp, flags şi ip extinşi la 32 biți. Aceste noi versiuni sunt denumite eax, ebx, ecx, edx, esi, edi, ebp, esp, flags, şi eip.

(b) În plus doi regiştrii de segment de 16 bişi fiecare: FS şi GS care împreună cu ceilalți patru regiştrii de segment permite programatorului să poată accesa şase segmente de memorie deodată.



regiştrii microprocesorului 80386



regiştrii microprocesorului Pentium I

Memoria Principală RAM

Memoria este acea parte a unui calculator în care pot fi păstrate programele şi datele pe o perioadă mai scurtă sau mai lungă de timp.

Memoria în care sunt păstrate programele înainte de execuție se numeşte **RAM (Random Access Memory)**, memorie cu acces aleator. Acest tip de memorie se mai numeşte şi **memorie principală**. Fără o memorie din care procesoarele să poată citi şi în care să poată scrie informații nu ar putea fi proiectat nici un calculator numeric cu program memorat.

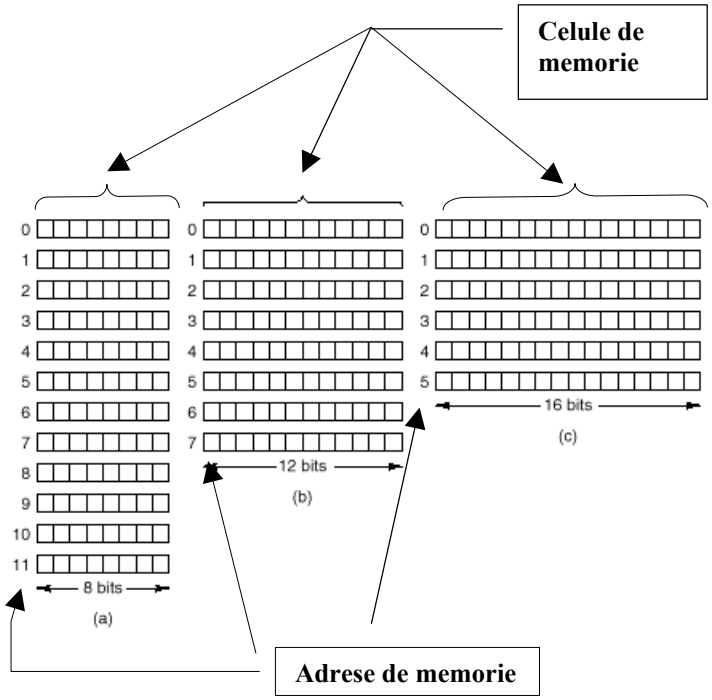
Unitatea elementară a memoriei este bitul. Un bit poate conține valoarea 0 sau 1. Informația numerică poate fi păstrată făcând distincție între două valori diferite ale unei mărimi fizice continue, cum ar fi tensiunea sau curentul. Cu cât numărul de valori distincte

este mai mare, cu atât nivelul de separație între două valori consecutive este mai redus, şi memoria este mai puțin sigură.

În acest capitol vom vorbi despre memoria principală (RAM).

Adrese de memorie

Memoriile sunt alcătuite din grupuri de biți care se numesc celule sau locații de memorie. Acestea pot păstra, fiecare o parte a informației . Fiecare celulă are un număr numit *adresă*, prin care programele se pot referi la ea. O memorie cu *n* celule, are adresele cuprinse între 0 şi *n*-1. Toate celulele unei memorii pot conține acelaşi număr de biți. Dacă o celulă poate conține *k* biți, ea poate memora oricare dintre cele 2^k combinații diferite. În figura următoare sunt prezentate trei organizări diferite pentru o memorie de 96 de biți. Celulele de memorie adiacente au adrese consecutive.



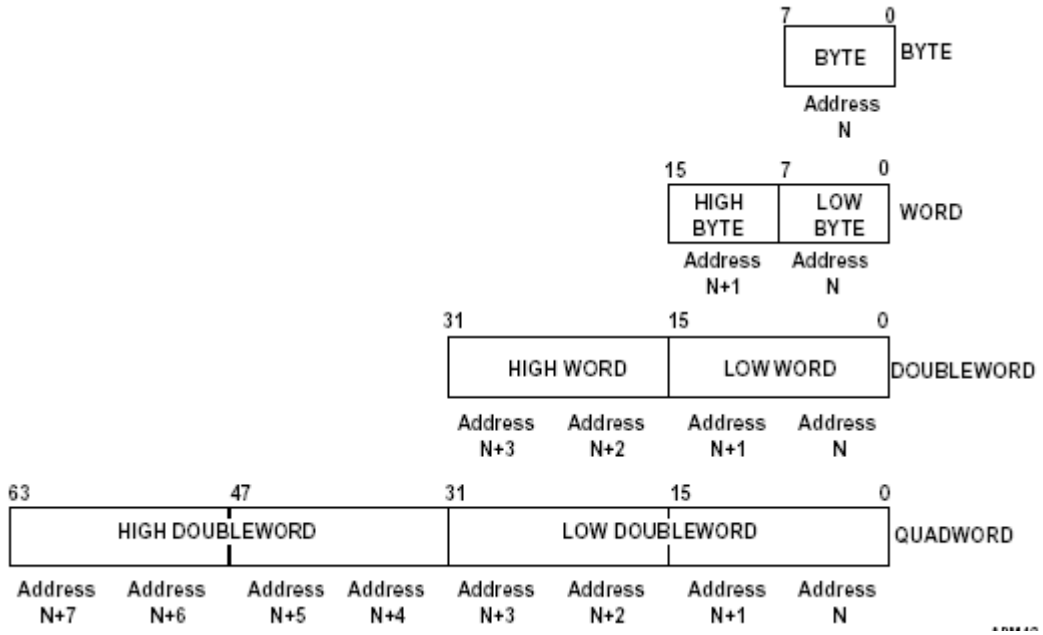
Trei moduri de organizare pentru o memorie de 96 de biți.

Dacă o adresă are *m* biți atunci numărul maxim de celule adresabile este 2^m. De exemplu, o adresă folosită pentru a referi memoria din figura a are nevoie de cel puțin 4 biți ca să poată exprima toate numerele de la 0 la 11. Pentru figura b şi c este suficientă o adresă de 3 biți. Numărul de biți din adresă determină numărul maxim de celule adresabile direct în

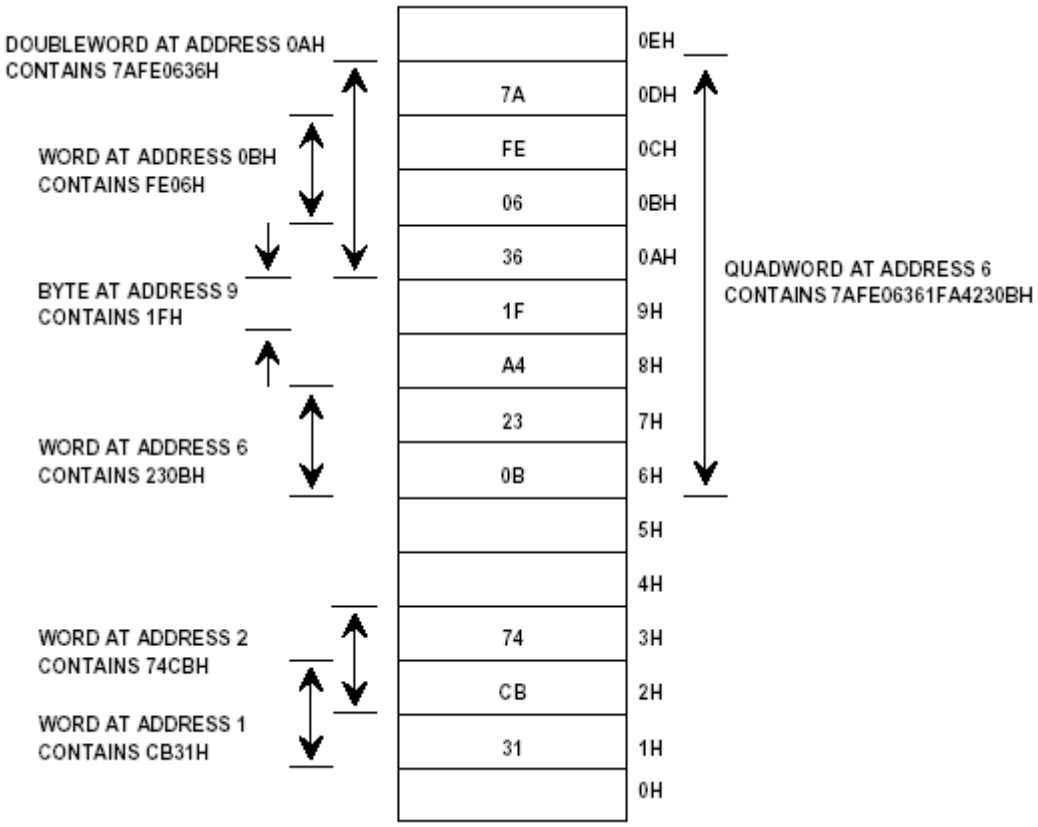
memorie şi nu depinde de numărul de biți ai celulei. Atât o memorie cu 2^{12} celule de câte 8 biți fiecare, cât şi o memorie cu 2^{12} celule de câte 64 de biți fiecare, necesită adrese pe 12 biți.

Celula de memorie este cea mai mică unitate de memorie adresabilă. Toți producătorii de calculatoare au adoptat *standardul celulei de 8 biți, adică de un octet (byte).* Octeții sunt grupați în cuvinte de câte 32 sau 64 de biți. Un calculator cu un cuvânt de 32 de biți are 4 octeți/cuvânt, iar unul cu un cuvânt de 64 de biți are 8 octeți/cuvânt. Importanța cuvântului provine din faptul că majoritatea instrucțiunilor operează cu cuvinte întregi, de exemplu adună două cuvinte. Astfel, o maşină pe 32 respectiv 64 de biți va avea registre de 32 respectiv 64 de biți şi instrucțiuni pentru prelucrarea cuvintelor de 32 respectiv 64 de biți.

În următoarele două figuri sunt prezentate tipurile de date ce pot fi reprezentate în memorie (a) precum şi aranjarea lor în memorie (b).



(a)



(b)

Tipuri şi capsule de memorie

Începând din anii “90 memoria primară este fabricată ca un singur cip (circuit integrat) care aveau o capacitate cuprinsă între 1K şi 1Mb care puteau fi înfipte într-unul din soclurile de memorie ale plăcii de bază a calculatorului.

În prezent se foloseşte o altă variantă. Un grup de cipuri, de obicei 8 sau 16, sunt cablate pe o placă de dimensiuni reduse şi vândute ca o singură unitate. Această unitate se numeşte **SIMM (Single Inline Memory Module) sau DIMM (Dual Inline Memory Module),** în funcție de cum este plasată seria de conectori – pe o singură parte sau pe ambele părți ale plăcii.

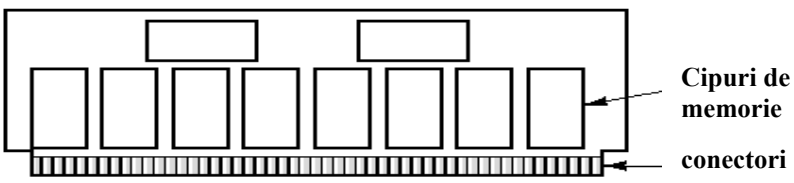


Fig 4.2 Un modul de memorie cu o singură serie de conectori (SIMM) cu capacitatea de 32Mb. Două cipuri controlează SIMM-ul.

Memoriile pot avea o capacitate de 32Mb, 64Mb sau 128Mb, 256 Mb, … . O altă caracteristică a memoriei primare este viteza de lucru, care poate fi de 60ns, 70ns sau de 80ns.

Organizarea fizică şi logică a memoriei principale la microprocesoarele 80x86

CPU este conectat la memoria principală prin magistrala de adrese şi date. Pentru a accesa un element de memorie procesorul transmite pe magistrala de adrese adresa elementului de memorie specificat. Memoria principală poate fi văzută ca un tablou de octeți. Folosind sintaxa Pascal avem:

```
Memory : array [0..MaxRAM] of byte;
```

Unde MaxRAM reprezintă numărul maxim de octeți de memorie principală care pot fi accesați prin intermediul magistralei de adrese.

Scrierea în memorie este echivalentă cu

```
Memory [address] := Value_to_Write;
```

Citirea unui element de memorie este echivalentă cu

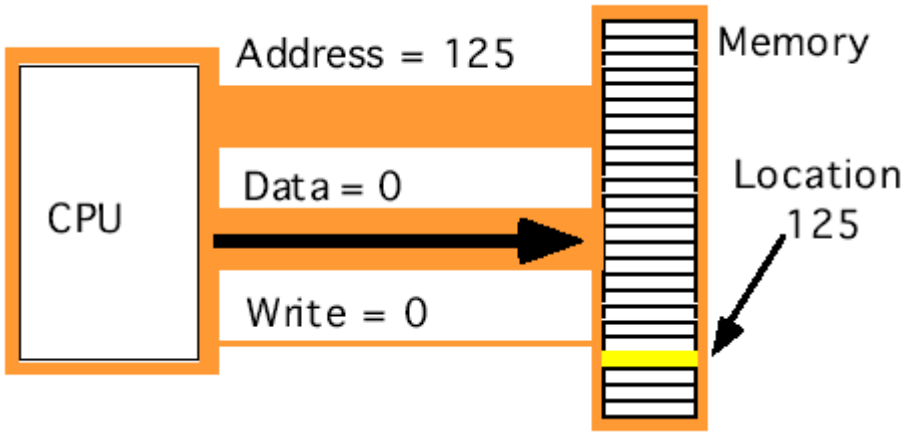
```
Value_Read := Memory [address];
```

Cu o singură linie pe magistrala de adrese, procesorul poate crea exact două adrese unice: 0 şi 1. Cu un număr de n linii de adresă pe magistrală, un procesor poate cerea, şi deci accesa , 2^n adrese unice. Deci numărul linii de adrese de pe magistrala de adrese determină numărul maxim de celule de memorie adresabile şi de locații de I/O. De exemplu, un procesor cu 20 de linii de adresă poate accesa până la 1,048,576 (sau 2^{20}) de locații de memorie. În continuare este prezentat un tabel cu numele procesorului, dimensiunea magistralei de adrese şi cantitatea maximă de memorie adresabilă.

Procesor	Mărime registre interne	Lățime magistrală date	Lărgime magistrală adrese	Max memorie adresabilă	Număr de tranzistoare
8088	16 biți	8 biți	20 biți	2^{20}= 1 048 576=1 Mb	29 000
8086	16 biți	16 biți	20 biți	2^{20}= 1 048 576=1 Mb	29 000
286	16 biți	16 biți	24 biți	2^{24}= 16 777 216=16 Mb	134 000
386 SX	32 biți	16 biți	32 biți	2^{32}=4 204 976 206=4 Gb	257 000
486	32 biți	32 biți	32 biți	2^{32}=4 204 976 206=4 Gb	1 600 000
Pentium I	32 biți	64 biți	32 biți	2^{32}=4 204 976 206=4 Gb	3 100 000
Pentium III	32 biți-128 biți	64 biți	32 biți	2^{32}=4 204 976 206=4 Gb	5 500 000
Pentium IV	32 biți-128 biți	64 biți	36 biți	2^{36}=32 Gb	>10 000 000

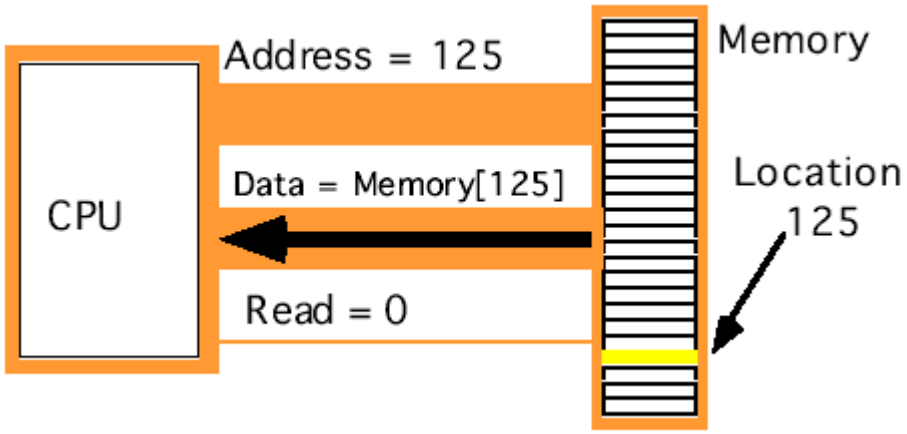
În mod real nu toată memoria disponibilă este folosită pentru instrucțiunile şi datele programelor. O anumită parte din memorie este folosită în alte scopuri.

Pentru a scrie în memorie, Memory [125] := 0, CPU plasează valoarea 0 pe magistrala de date, adresa 125 pe magistrale de adrese şi activează linia de scriere în memorie (write) din magistrala de comenzi.



Scrierea unei date în memoria principală

Pentru a citi din memorie, CPU:=Memory [125] , CPU plasează valoarea adresei, adică 125 pe magistrale de adrese şi activează linia de citire din memorie (read) din magistrala de comenzi după care citeşte data pusă pe magistrala de date.



citirea unei date din memoria principală

Moduri de organizare a memoriei

Una dintre cele mai importante caracteristici ale memoriei este modul de organizare al ei şi implicit modul de accesare a informației din memorie.

Aşa cum am văzut, memoria este organizată ca un set de locații (celule) de memorare numerotate consecutiv începând de la 0. *Adresa fizică* a unei locații de memorie este reprezentată printr-un număr asociat locației respective, iar mulțimea totală a adreselor fizice reprezintă spațiul de adrese fizice al memoriei care este limitat de dimensiunea magistralei de adrese. Prin urmare trebuie făcută observația că numărul de cuvinte adresabile depinde numai de numărul de biți de adresă şi nu de numărul de cuvinte disponibile în memoria respectivă.

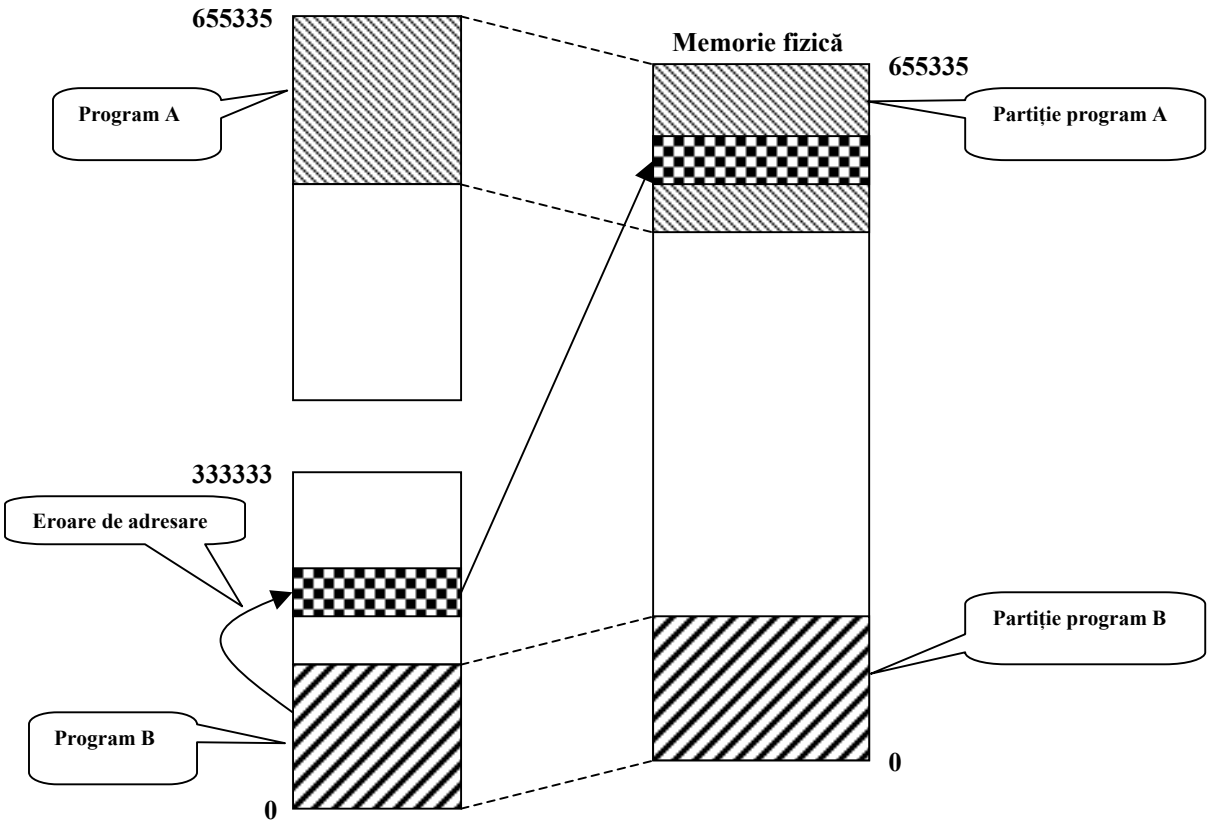
O *adresă logică* este o adresă utilizată într-o instrucțiune dintr-un program iar mulțimea totală a acestora reprezintă spațiul adreselor logice. Organizarea acestui spațiu defineşte arhitectura memoriei principale şi este de o importanță majoră pentru orice programator.

Organizarea spațiului de adrese fizice este determinată de tehnologia utilizată pentru fabricarea memoriei. Organizarea spațiului de adrese logice este determinată de sistemul de operare şi de structura programelor ce vor rula în memorie. La primele calculatoare spațiul de adrese logice era identic cu spațiul de adrese fizice.

Memoria liniară sau modul real de organizare al memoriei

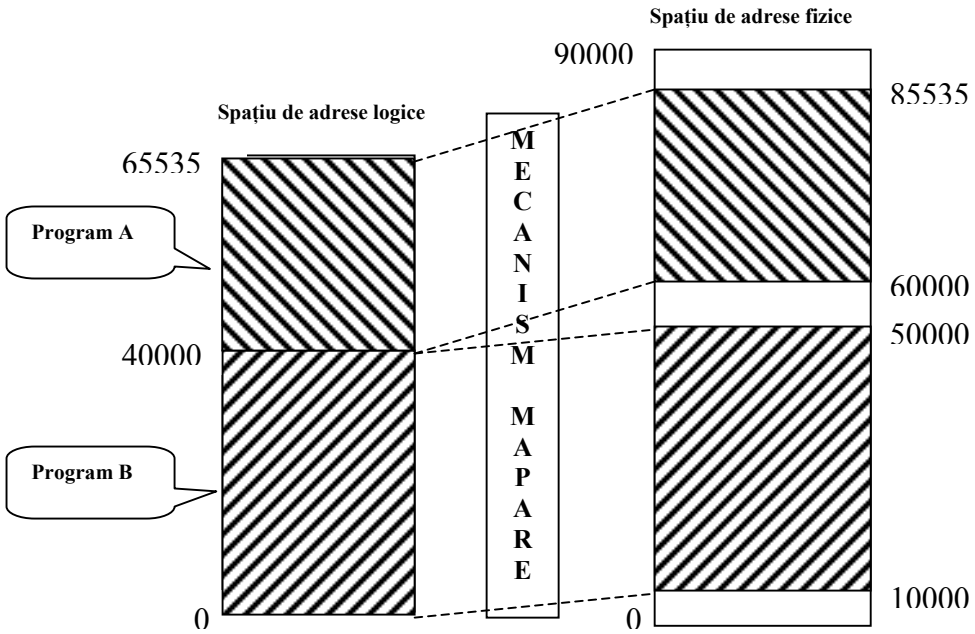
In cazul primelor calculatoare memoria era organizată într-un spațiu liniar şi continuu de adrese care porneau de la 0 şi se continuau fără întreruperi până la limita superioară impusă

de magistrala de adrese (numărul total de biți dintr-o adresă logică). În acest caz spațiul de adrese logice coincidea cu spațiul de adrese fizice. Dezavantajul unui astfel de memorii era că dacă două programe care aveau codurile plasate la adrese diferite făceau din greşeală un acces la partiția celuilalt (eroare de adresare) (vezi figura de mai jos), rezultatele erau catastrofale.



Maparea memoriei liniare

Maparea este procesul prin care adresele logice pot fi translatate în adrese fizice, adică este un mecanism de realocare a spațiului de adrese logice peste spațiul de adrese fizice.
Acest proces este foarte util în sistemele în care mai multe programe îşi partajează memoria fizică (multitasking), fără a interfera între ele. Astfel, fiecare program are propriul spațiu de adrese logice, independent de celelalte programe.



Exemplu de mapare

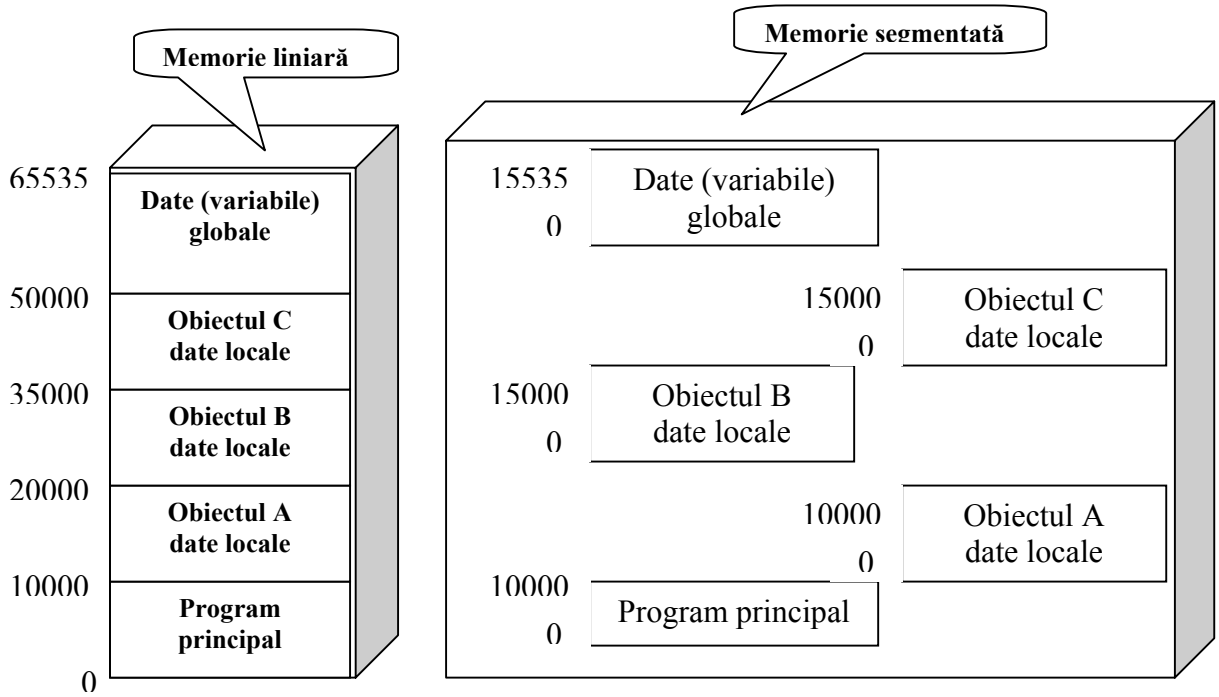
Memoria segmentată

Segmentarea memoriei este un mecanism de management al memoriei care permite programatorilor să-şi partiționeze programele în module care pot opera independent unul de celălalt şi care pot creşte sau descreşte în dimensiune.

În acest mod memoria este divizată logic în mai multe bucăți numite *segmente de memorie*. În fiecare segment adresarea componentelor sale (cuvinte de memorie) este liniară. O adresă de memorie este compusă în acest caz din selectorul segmentului de memorie şi deplasamentul (offset-ul) din cadrul segmentului respectiv, şi are forma :

Selector_ segment:offset.

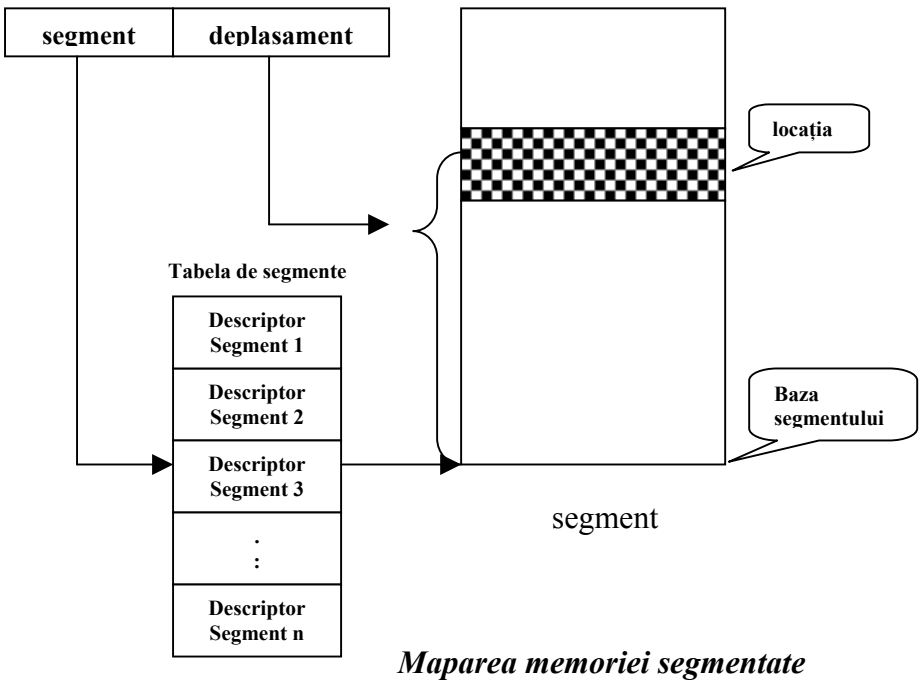
Avantajul acestei organizări a memoriei este acela că programele pot fi încărcate în memorie în segmente astfel încât să reflecte structura logică a lor (programelor). Astfel un segment poate conține codul programului principal, altul codul unei proceduri, altul codul unui obiect al programului şi aşa mai departe. Dacă ne gândim că noile tendințe ale programării sunt programarea orientată pe obiecte, atunci putem atribui fiecărui segment de memorie un obiect care în cursul execuției programului poate creşte sau descreşte în dimensiune.



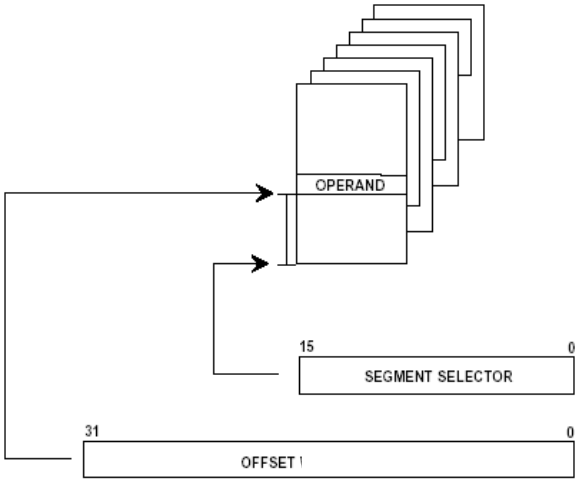
comparație între memoria liniară şi cea segmentată

Maparea memoriei segmentate

În acest caz maparea se face printr-o tabelă de segmente care conține câte o intrare numită descriptor de segment (la care se referă selectorul de segment) pentru fiecare segment existent în memorie. Descriptorul de segment conține adresa de început a segmentului, lungimea acestuia şi drepturile de acces la segmentul respectiv. Drepturile de acces la un segment se referă la posibilitatea unui program de a accesa sau nu segmentul respectiv.



Maparea memoriei segmentate



Modul de adresare a memoriei segmentate

Până la procesoarele 80386 exclusiv selectorul segmentului şi adresa offset-ului erau scrise pe 16 biți. De la 80386 adresa de offset este scrisă pe 32 de biți.

Dimensiune maximă a unui segment este dată de dimensiunea offset-ului, de exemplu pentru un offset de 16 biți avem $2^{16}=2^6 2^{10}$=64Kb/segment, iar pentru un offset de 32 de biți avem $2^{32}=2^2 2^{30}$=4 Gb/segment

Memoria virtuală (sau modul protejat)

Aşa cum am văzut la maparea memoriei liniare unui cod de program poate ocupa un spațiu de adrese logice diferite de cele fizice in memoria principală a unui calculator. Ce se întâmplă dacă programul face o referire la o adresă logică a cărei mapare depăşeşte spațiul adreselor fizice existent ? Acest lucru ar cauza o eroare şi programul nu ar mai putea rula în continuare. Ce se întâmplă dacă programul are nevoie de mai multă memorie fizică decât cea existentă ?

La începutul calculatoarelor soluția la problemele enunțate era ca programatorul să folosească memorii secundare de tipul hard discului. Programul trebuia împărțit de programator în mai multe componente care încăpeau în memoria RAM, numite faze (overlays).

Programatorul trebuia să împartă programul in faze, să stabilească locul din memoria secundară (hard disc) unde să fie păstrată fiecare fază şi, în mare, să gestioneze tot procesul de

lucru cu fazele. Evident că această gestionare a memoriei implica din partea programatorilor o muncă foarte mare.

În 1961 un grup de cercetători din Manchester, Angila, a propus o metodă automată de înlocuire a fazelor numită astăzi *memorie virtuală (virtual memory)* şi care elibera programatorul de munca enormă de gestionare a fazelor.

Prin urmare, dacă pe o maşină fără memorie virtuală s-ar face un salt la o adresă de memorie care nu ar avea corespondentă o adresă fizică a memoriei RAM, s-ar genera o eroare. Pe o maşină cu memorie virtuală s-ar executa următorii paşi:

1. Conținutul memoriei principale (faza din RAM) este memorat pe (hard disc).
2. Adresa de memorie cerută este localizată pe hard disc.
3. Cuvintele care se găsesc în aceeaşi fază cu cuvântul de la adresa de memorie cerută sunt încărcate în memoria principală.
4. Se schimbă corespondența de adrese astfel încât adresele din faza încărcată în memorie să corespundă adreselor de la 0 la memMax.
5. Execuția programului continuă.

Tehnica de gestionare automată a fazelor se numeşte *paginare (paging) iar bucățile de program citite de pe disc se numesc pagini*.

În cazul paginării programele pot citi sau scrie date la adrese de memorie virtuale chiar dacă acestea nu se găsesc în momentul respectiv în RAM. Programatorul nu mai trebuie să gestioneze paginarea, calculatorul comportându-se ca şi cum ar avea destulă memorie. Fiecare calculator cu memorie virtuală are un dispozitiv care face translatarea de la memoria virtuală la memoria fizică. Acest dispozitiv se numeşte MMU (Memory Managemernt Unit) (Unitate de gestiune a memoriei) şi poate fi localizat înăuntrul CPU sau în exteriorul ei. În cazul memoriei virtuale adresa de localizare a unei informații se numeşte *adresă virtuală.*

Memoria virtuală la procesoarele Pentium

Procesorul Pentium are un sistem de memorie virtuală sofisticat, compus din mecanisme de paginare şi segmentare a memoriei virtuale. Prin urmare definiția memoriei virtuale necesită o actualizare. Memoria virtuală se defineşte ca fiind „totalitatea hărții

memoriei care poate fi formată prin concatenarea numărului maxim de segmente , fiecare presupus la dimensiunea sa maximă, pe care un utilizator le poate crea prin folosirea unei adrese virtuale". Adresa virtuală este o generalizare a adresei logice . Dimensiunea memoriei virtuale este cu mult mai mare decât dimensiunea memoriei fizice principale, ea fiind compusă din toate resursele de memorie ale calculatorului, atât interne cât şi externe.

Un task (proces) este o secvență de acțiuni coerente care conduc la îndeplinirea unui scop folosindu-se în mod dinamic de resursele calculatorului: CPU, RAM, memorie externă, … .

Memoria virtuală permite folosirea unor concepte avansate cum ar fi multiprocesarea (multitasking). Acest concept se referă la capacitatea calculatorului de a executa mai multe procese simultan (adică comutarea , secvențială, de la o secvență de instrucțiuni a unui proces (task) la altă secvență de instrucțiuni aparținând altui proces, apoi se comută din nou la primul task, şi aşa mai departe. Acest mecanism este transparent utilizatorului care are impresia execuției simultane a mai multor procese.

Multiprocesarea poate însemna:

A. Pe un calculator cu un singur procesor (execuția secvențială)
    a. Mai mulți utilizatori simultan pe un calculator fiecare cu procesul lui.
    b. Un singur utilizator care are nevoie de mai multe procese (programe) simultan.

B. Pe un calculator multiprocesor (Execuția in mod real a mai multor procese simultan, în paralel (în acelaşi timp)).
    a. Mai mulți utilizatori simultan pe un calculator fiecare cu procesul lui.
    b. Un singur utilizator care are nevoie de mai multe procese (programe) simultan.

Un alt concept important este mecanismul de protecție al memorie care constă din trei aspecte de bază:

- controlul informației (instrucțiuni sau diverse tipuri de date);
- izolarea utilizatorilor unul față de altul (protecție „inter-task");
- izolarea software-ului de sistem de softul se aplicații (protecție „intra-task").

Mecanismul de protecție al memoriei este compus dintr-un mecanism de protecție al segmentelor şi dintr-un mecanism numit privilegiile multi-nivel care asigură izolarea software-ului de sistem de cel de aplicații prin ierarhizarea pe mai multe niveluri de privilegii a programelor (coduri şi date). Aceste mecanisme vor fi descrise în paragraful următor.

Modul de adresare al memoriei virtuale la procesoarele Pentium

Memoria virtuală se împarte logic în două spații :

- Spațiul adreselor globale - folosit pentru date şi procese de sistem, incluzând sistemul de operare şi alte servicii de sistem care sunt partajate de către toate taskurile (programele).
- Spațiul adreselor locale – cuprinde segmente de program şi date pentru fiecare proces în parte.

Sistemul de memorie virtuală cuprinde două tabele:

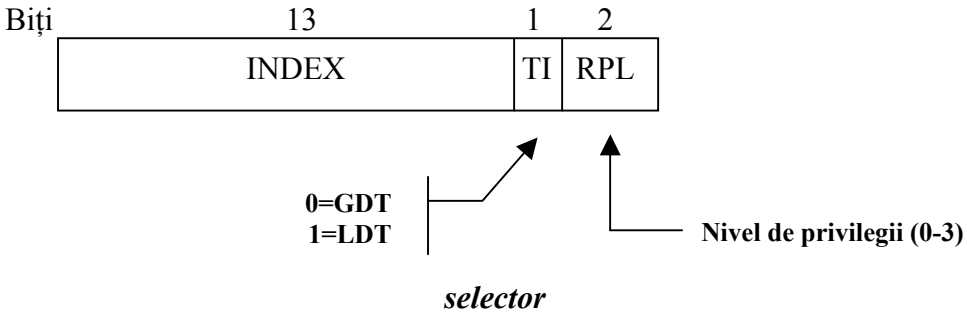
LTD (Local Descriptor Table – tabela descriptorilor locali) asociată fiecărui program, descriind segmentele locale ale lui (segmentele cu codul, datele , stiva, …).

GDT (General Descriptor Table – tabela descriptorilor generali) descrie segmentele sistemului de operare.

Fiecare program are, deci, o tabelă LTD, însă există o singură copie a tabelei GDT partajată de toate programele de pe calculator.

O adresă virtuală are 48 de biți şi este compusă din două componente:

- adresa efectivă (sau offsetul adresei virtuale) pe 32 de biți;
- selectorul pe 16 biți, care înlocuieşte adresa segment.

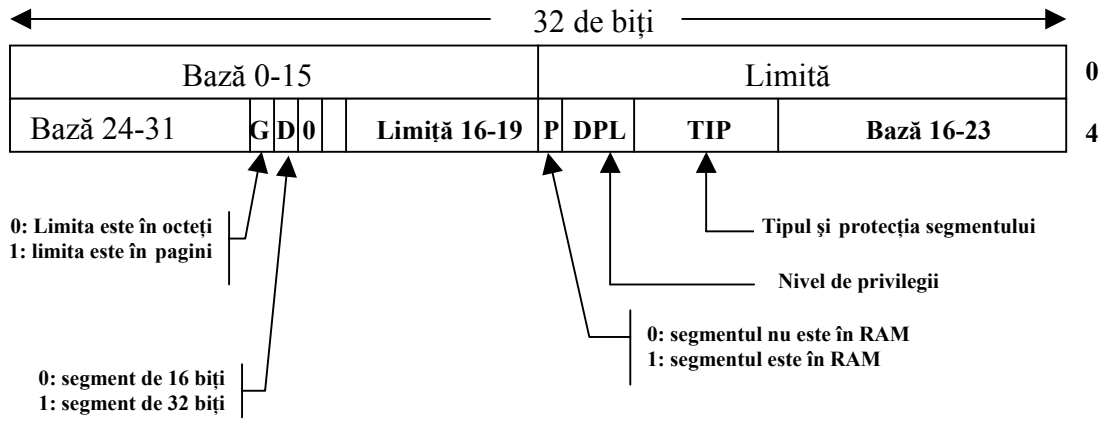


selector

Unul dintre biții selectorului (TI) ne spune dacă este vorba de un segment local sau global, adică dacă segmentul este LDT sau GDT. Ceilalți 13 biți pentru Index plus bitul TI precizează numărul intrării în LDT sau GDT. Prin urmare avem posibilitatea de a forma

2^{14} segmente =16Kb de segmente şi deci, dimensiunea memoriei virtuale pe care o pot adresa Intel Pentium este dată de numărul maxim de segmente, fiecare presupus la dimensiunea maximă: 2^{14} segmente * 2^{32}B/segment = 2^{46}B = 64 Tb. Mai există 2 biți pentru protecție (RPL).

Un selector acționează ca un index (intrare sau adresă) într-un tabel de tip LDT sau GDT. În momentul în care un selector este încărcat într-un registru de segment, descriptorul lui este preluat din LDT sau GDT şi memorat in registrele interne ale MMU, pentru a putea fi accesat mai repede. Un descriptor este construit din 8 octeți aşa cum este prezentat mai jos.

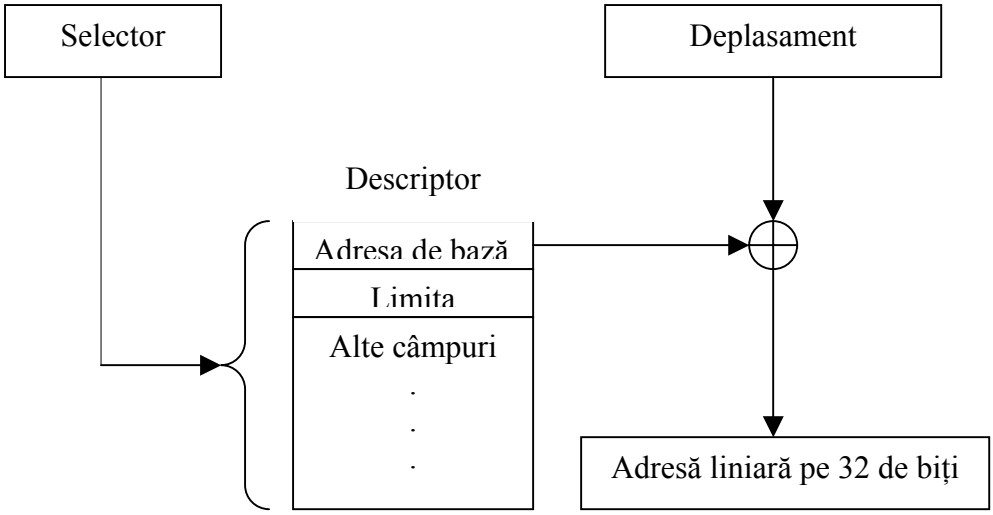


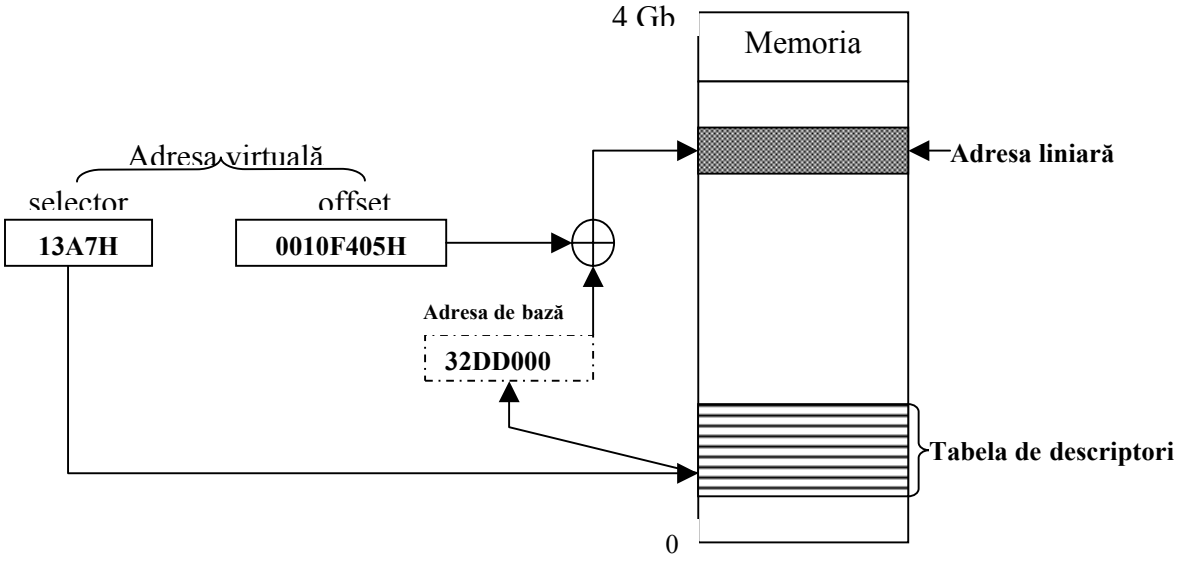
descriptor de segment de cod

După ce sa găsit descriptorul de segment corespunzător selectorului se testează dacă câmpul P este 0. Dacă câmpul P este 0 atunci se cere sistemului de operare să găsească segmentul în memoria externă. Se testează, folosind câmpul G (granularitate) şi câmpul LIMITĂ, dacă adresa efectivă (deplasamentul) se încadrează în limita segmentului astfel :

- dacă G =0 câmpul LIMITĂ reprezintă dimensiunea exactă a segmentului până la 2^{20}=1 Mb.
- dacă G=1 câmpul LIMITĂ reprezintă dimensiunea segmentului în pagini şi nu în octeți.

Dacă adresa efectivă (deplasamentul) este corect şi segmentul este adus în RAM, procesorul Pentium adună la adresa efectivă câmpul Bază (adresa fizică a bazei segmentului) de 32 de biți din descriptor pentru a forma ceea ce se numeşte o adresă liniară. Acest proces este prezentat în figura de mai jos.



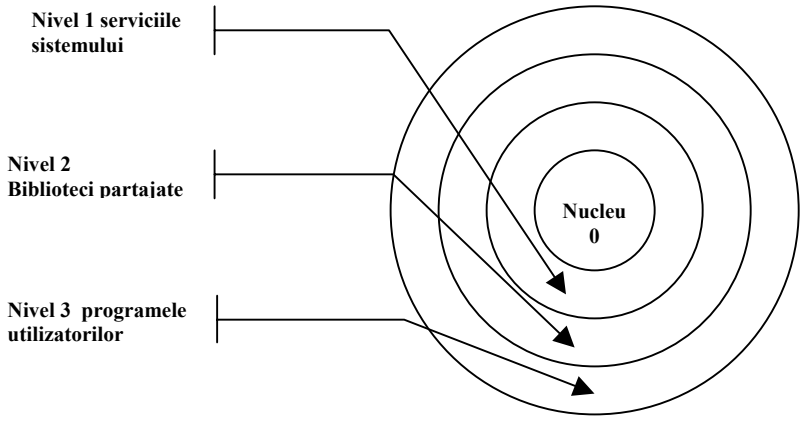


Explicația în două desene a modului de translatare a adresei virtuale în adresă liniară

Deoarece pentru acest tip de microprocesor adresa fizică este pe 32 de biți, harta memoriei fizice conține 2^{32} biți= 4 Gb.

Nivelurile de privilegiu

Câmpul RPL din descrierea selectorului reprezintă nivelul de privilegiu al segmentului respectiv cuprins între (0 – 3), 0 fiind cel mai privilegiat nivel iar 3 cel mai puțin privilegiat. Mecanismele de protecție ale procesorului Pentium se bazează pe conceptul de ierarhie de privilegii. Aceste privlegii acționeză astfel: un program are dreptul să folosească segmente de acelaşi nivel de privilegiu cu al său sau mai mare. Invers este interzis. Pe nivelul 0 se află nucleul sistemului de operare care gestionează operațiile de I/O, operațiile cu memoria şi alte operații critice. La nivelul 1 se găsesc serviciile sistemului de operare disponibile utilizatorilor. La nivelul 2 se găsesc biblioteci de proceduri care pot fi partajate de programele aflate în rulare. Pe nivelul 3 se găsesc programele utilizatorilor (programe de aplicații).

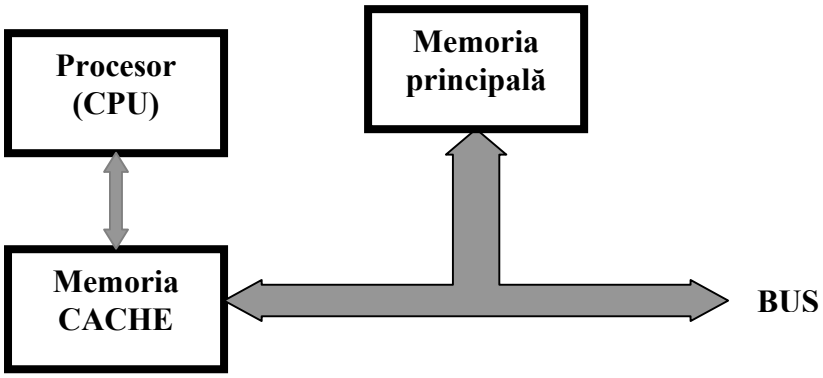


Protecția la Pentium

Memoria intermediară

Memoria intermediară se situează între procesor şi RAM. Deoarece procesoarele au o viteză de lucru mai mare decât memoriile RAM, sau dezvoltat tehnici de combinare a unei cantități mici de memorie rapidă cu o cantitate mare de memorie lentă, astfel încât să se obțină aproape viteza memoriei rapide şi capacitatea celei mari. Memoria mică rapidă se numeşte memorie intermediară sau memorie cache. Această memorie este formată din cipuri de memorie cu viteză de lucru mare şi care păstrează copia ultimelor date accesate de către procesor. Astfel, cuvintele de memorie cele mai des utilizate sunt păstrate în memoria

intermediară. Atunci când UCP are nevoie de un cuvânt, îl caută mai întâi în memorie intermediară, iar în cazul în care nu îl găseşte îl caută în memoria principală. Astfel timpul mediu de acces la date este redus semnificativ. Instrucţiunile unui program nu accesează memoria absolut la întâmplare. Dacă o anumită referire la memorie este la adresa A, atunci este foarte probabil ca următoarea referire la memorie să se afle în vecinătatea lui A. Acest principiu conform căruia referirile la memorie făcute într-un interval scurt de timp tind să folosească doar o porţiune mică din întreaga memorie, se numeşte principiul localităţii şi formează baza tuturor sistemelor de memorie intermediară. El funcţionează în următorul mod: atunci când un cuvânt este referit, atât el cât şi o parte din vecinii săi sunt aduşi din memoria mare şi lentă (RAM) în memoria intermediară, astfel încât la următoarea utilizare să poată fi accesat rapid. Memoria cache este plasată în mod logic între UCP şi memoria principală, iar din punct de vedere fizic ea poate fi plasată în mai multe locuri.



Plasarea memoriei intermediare.

Dacă un cuvânt este scris sau citit de k ori într-un interval scurt de timp, calculatorul va avea nevoie de 1 referire la memoria lentă şi *k-1* referiri la memoria intermediară (cache). Deci, cu cât *k* este mai mare cu atât performanţa globală este mai bună.

Folosind principiul localităţii datelor, memoriile principale şi cele intermediare sunt împărţite în **blocuri de memorie de dimensiune fixă**. Blocurile memoriei intermediare sunt numite **linii de memorie intermediară (cache lines)**. La apariţia unei erori întreaga linie este reâncărcată din memoria principală, nu doar cuvântul căutat. De exemplu, cu o linie de 64 de octeţi, o referire la adresa de memorie 260 va încărca într-o linie de memorie intermediară toţi octeţii cuprinşi între 256 şi 319. Acest mod de funcţionare este mult mai eficient decât extragerea cuvintelor individuale, deoarece extragerea a *k* cuvinte odată se face mai rapid

decât extragerea unui cuvânt de *k* ori. O memorie cache poate avea capacități cuprinse între 32KB şi câțiva megaocteți.

O altă caracteristică importantă este plasarea informațiilor în memoria cache. Astfel există două variante: prima este cea cu memorie intermediară unică în care datele şi instrucțiunile folosesc aceeaşi memorie intermediară, şi cealaltă variantă este cu memorie intermediară divizată, cu instrucțiunile într-o memorie intermediară şi datele în alta. Ultima variantă se mai numeşte arhitectură Harvard.
Fizic memoria cache poate fi localizată în interiorul cipului UCP (tip L1) şi/sau în exteriorul lui (tip L2).