UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI
Instrumente Structurale 2007-2013

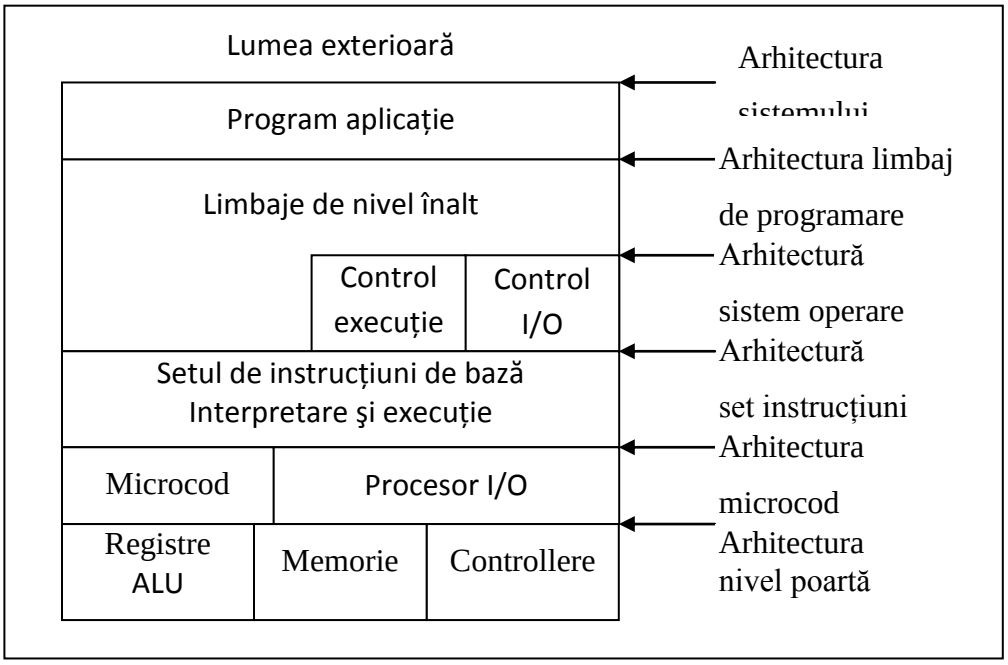
Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Programare în limbaj de asamblare

7. Arhitectura calculatoarelor şi a memoriei.

Arhitectura calculatoarelor

Termenul „*arhitectura calculatoarelor*“ este adesea utilizat cu semnificația, simplă, de „organizarea şi proiectarea calculatoarelor“. În practică într-un sistem de calcul există mai multe arhitecturi distincte, fiecare fiind definită de o legătură între diferitele niveluri ale sistemului. Figura prezintă o schemă abstractă a unui sistem de calcul, unde cele mai simple operații şi funcții sunt plasate în partea inferioară, iar cele mai complexe operații, dependente de utilizator, ocupă nivelurile superioare. Fiecare nivel foloseşte funcțiile furnizate de nivelul anterior (cel ierarhic inferior).



Arhitecturi ale unui sistem de calcul

O arhitectură este, deci, legătura sau interfața între două astfel de module funcționale. Ea poate fi definită ca „vederea funcțională a sistemului sub o interfață (nivel) pentru un utilizator care se situează deasupra interfeței (nivelului)“. Proiectanții care utilizează interfața (nivelul) arhitecturală nu sunt, în general, preocupați de detaliile sistemului inferioare nivelului arhitecturii sistemului, sau de orice arhitecturi la niveluri inferioare. Ei sunt preocupați de funcționarea sistemului la nivelul interfeței imediat sub nivelul utilizator.

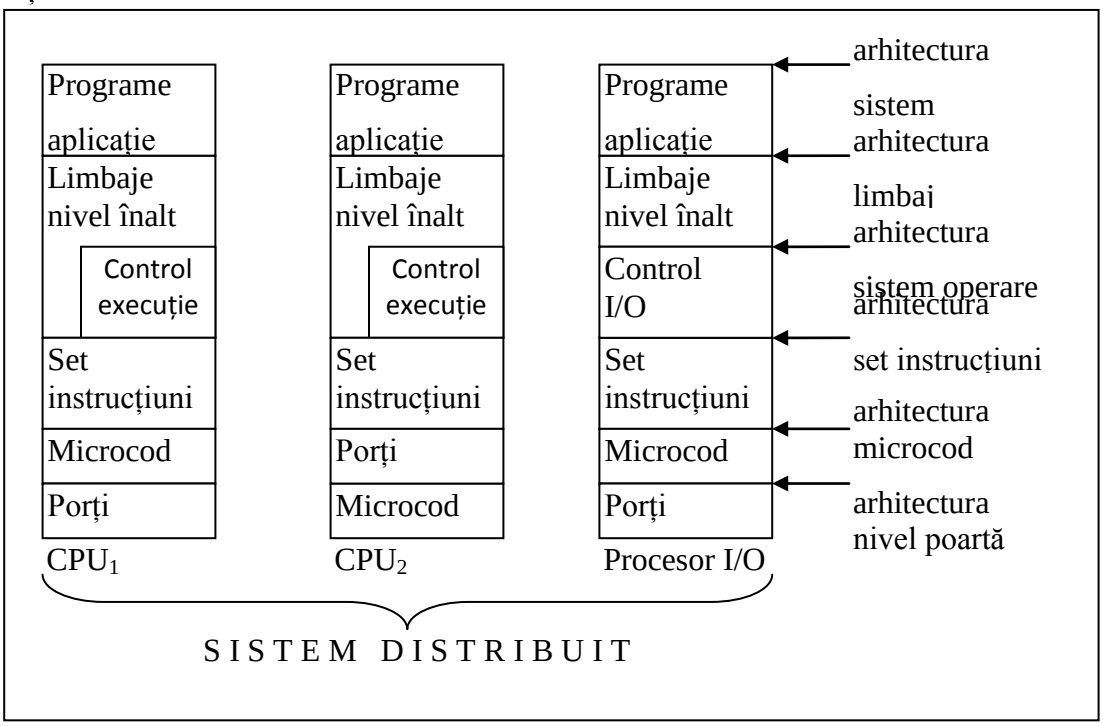
Arhitectura globală este interfața între întregul sistem de calcul şi lumea exterioară; aceasta este denumită arhitectura sistemului. Undeva, mai jos, este interfața între programul de aplicație şi limbajul de programare de nivel înalt (presupunând că aplicația este scrisă într-un limbaj de nivel înalt). Această limitare defineşte arhitectura limbajului de programare. Programatorul este plasat la acest nivel. În continuare, este interfața între limbajul de programare şi diferite funcții de administrare a resurselor, în timpul execuției, care sunt furnizate de către sistemul de operare. Acest nivel este definit ca arhitectura sistemului de operare.

Următoarea interfață este deosebit de importantă, deoarece în calculatoarele convenționale ea defineşte limita dintre hardware şi software. Este nivelul elementar, la care instrucțiunile recunoscute de calculator sunt decodificate şi executate. Acesta este nivelul

arhitecturii setului de instrucțiuni. Celelalte două niveluri inferioare de arhitecturi (arhitectura microcod şi arhitectura la nivel de porți) definesc mai în detaliu alte funcții primitive, dar care nu sunt interesante pentru cei mai mulți dintre programatori.

În general, însă, termenul de arhitectura calculatorului defineşte granița dintre hard şi soft. Arhitectura calculatorului este nivelul sistemului de calcul ce este văzut de un programator în limbaj de asamblare sau de unul care scrie un compilator.

O separare pe verticală a componentelor din figura următoare permite definirea de sisteme multiprocesor sau distribuite; această „împărțire“ se poate spune că defineşte arhitectura configurației.



Arhitectura configurației

Arhitectura memoriei

Una dintre cele mai importante caracteristici ale arhitecturii unui calculator este modul de organizare al memoriei şi modul în care se obține acces la informația din memorie.

Memoria principală este organizată ca un set de locații de memorare, numerotate consecutiv, începând de la 0. Numerele asociate locațiilor fizice reprezintă adresa fizică, iar mulțimea totală a adreselor fizice constituie spațiul de adrese fizice.

O adresă logică este o adresă utilizată de către programator într-o instrucțiune. Adresele ce pot fi utilizate de un program constituie spațiul de adrese logice. Organizarea acestui spațiu defineşte arhitectura memoriei.

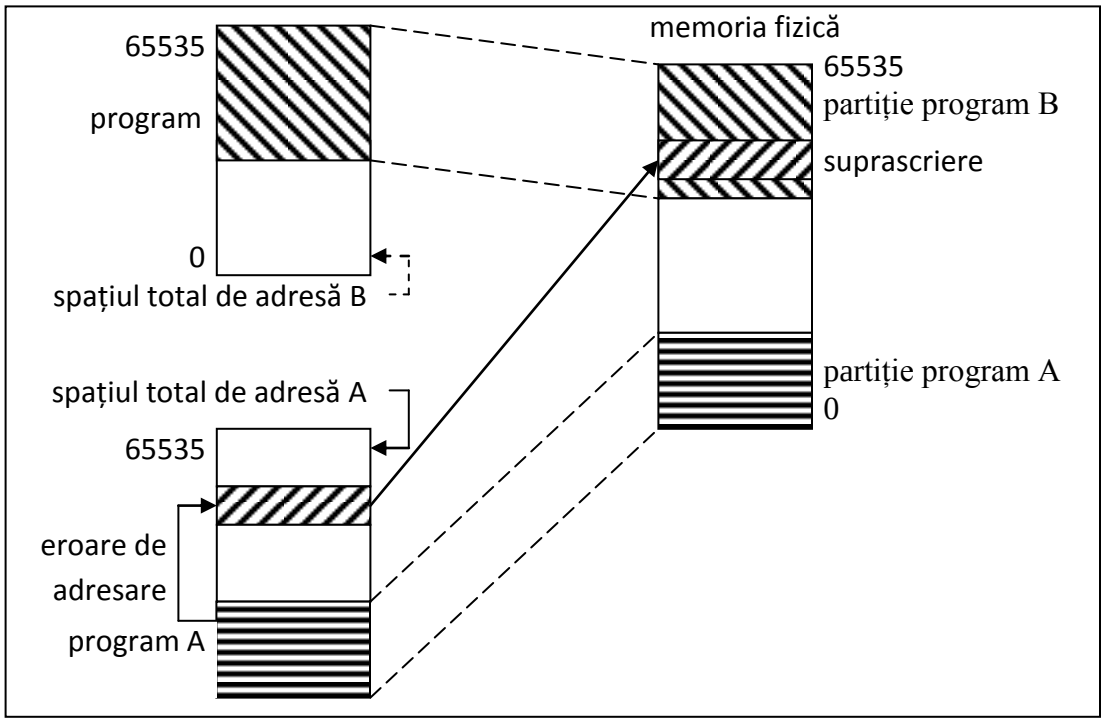
Organizarea spațiului de adrese fizice este determinată de tehnologia utilizată pentru memorie şi de costul ei, dar spațiul de adrese logice nu este condiționat de nici una dintre aceste considerații. Din contră, organizarea memoriei logice este determinată de structura programelor ce vor rula în memorie. Inițial, spațiul de adrese logice era identic cu spațiul de adrese fizice.

Memoria liniară

Este cea mai obişnuită organizare a spațiului de adrese logice, cea mai obişnuită arhitectură

pentru memorie: un spațiu liniar, continuu de adrese. Adresele pornesc de la zero şi se succed, într-un mod liniar, fără goluri sau întreruperi, până la limita superioară, impusă de numărul total de biți dintr-o adresă logică. Un program, adesea constând din mai multe proceduri, şi datele sale sunt plasate în acest spațiu de adresă unic. Spațiul de adrese logice al unei memorii liniare are, astfel, aceeaşi organizare ca memoria fizică.

Astfel, pentru o memorie cu 16 linii de adresă pot fi generate 65535 de adrese distincte. O astfel de adresă generată de un program este utilizată direct de hardware-ul memoriei pentru a plasa data. Iată ce se întâmplă în cazul a două programe, având codurile plasate la adrese diferite de memorie şi care nu se suprapun; să presupunem că unul dintre cele două programe (B) face o operație într-o locație în afara codului. În figură se prezintă ce se poate întâmpla în acest caz: vulnerabilitatea memoriei „nemapate“.

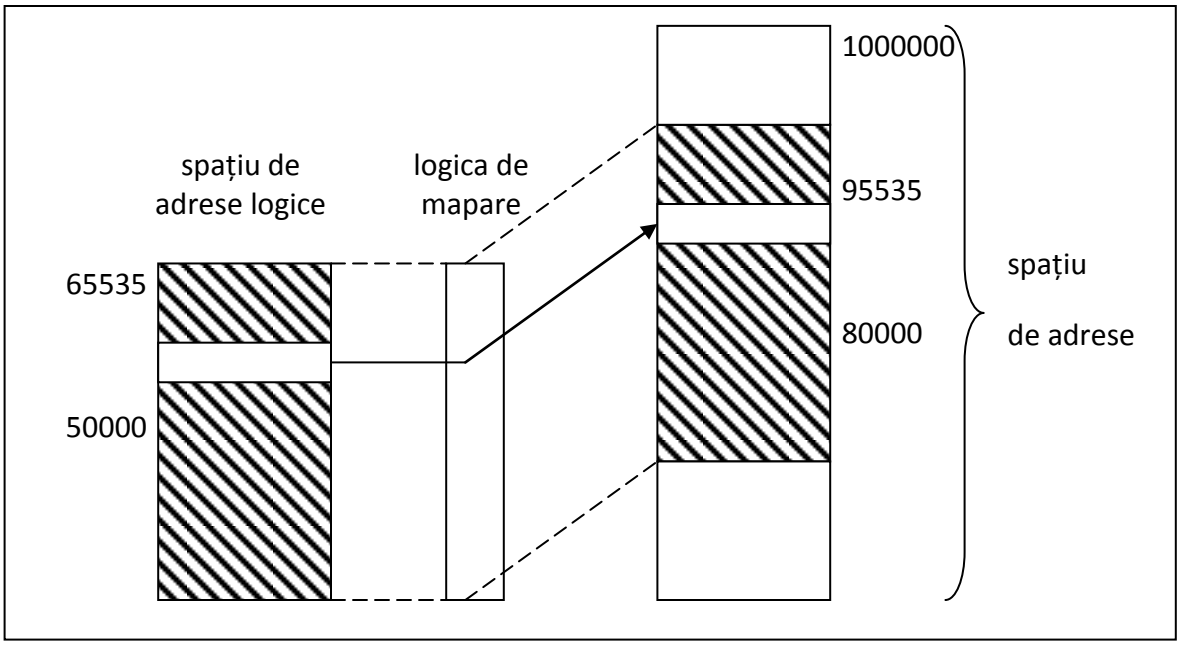


Vulnerabilitatea memoriei 'nemapate'

Maparea memoriei liniare

Maparea este, în esență, procesul de translatare a adreselor logice în adrese fizice. În exemplul anterior adresele logice sunt echivalate cu adresele fizice; dar prin exploatarea „mapării“, o adresă logică poate fi atribuită la o adresă fizică arbitrară. Deci „maparea“ este un mecanism pentru realocarea spațiului de adrese logice peste spațiul de adrese fizice.

În figura următoare se prezintă o operație de mapare foarte simplă. Întregul spațiu de adrese logice ale unui program (locațiile 0-65535, în acest caz) este mapat (suprapus) peste locațiile fizice 30000-95535. Astfel, o referință a unui program la locația 50000, va prelua datele, efectiv de la locația fizică 80000 (30000+50000).

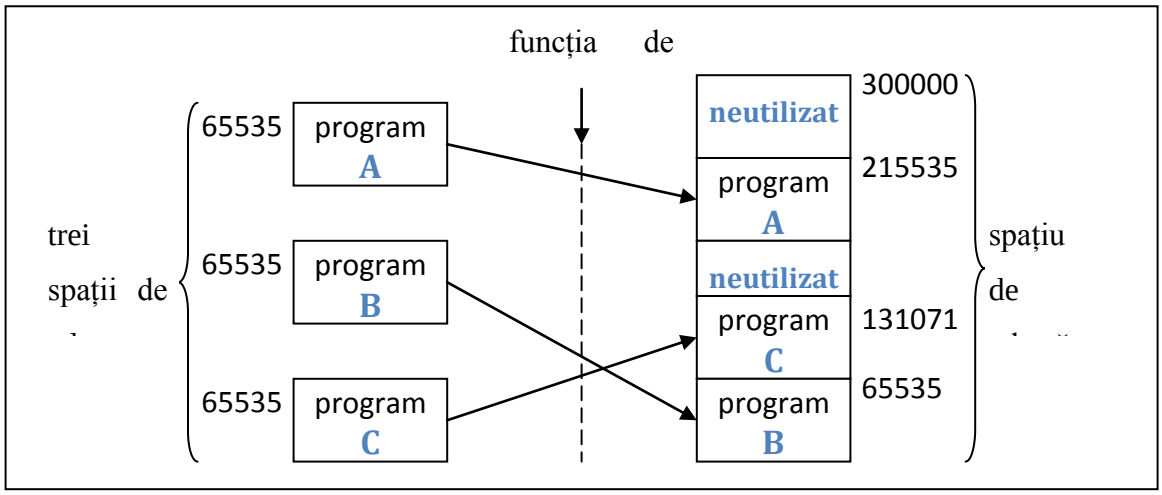


O schemă simplă de mapare

Această operație este foarte utilă în sistemele multiprogram (sau multitask), pentru care a fost dezvoltată maparea. Astfel, fiecare program are propriul său spațiu logic de adrese, care este complet independent de spațiul oricărui alt program. Ca atare, mai multe programe pot partaja memoria fizică fără posibilitatea de a interfera între ele. Procesul de mapare potrivește (suprapune) tot spațiul de adrese logice într-o zonă de memorie fizică, dar procesul este transparent pentru program. Iată, de exemplu, cum se realizează maparea mai multor programe.

Maparea bazată pe pagini

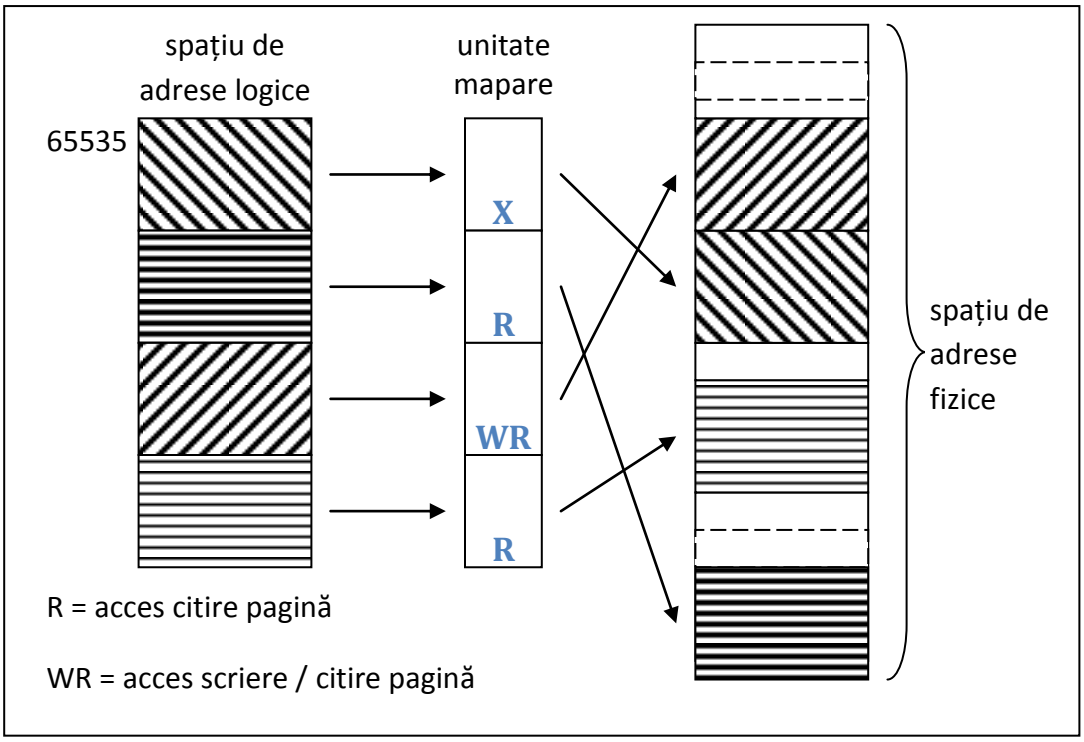
În locul mapării întregului spațiu logic de adrese ca o unitate, ca în figura de mai jos, mecanisme mai avansate de translație a adresei mapează „pagini“ de dimensiune fixă, mai mici, ale spațiului de adrese logice, în pagini de memorie fizică. Astfel, un program mare nu trebuie să fie realocat într-o zonă (porțiune) continuă de memorie, care poate fi greu găsită într-un cadru cu programe multiple, decât, mai degrabă, în mai multe secțiuni de memorie, mai mici, care sunt mult mai ușor de găsit, disponibile. Sunt mai ușor de găsit 20 de pagini de 1 Ko, decât un bloc de 20 Ko.



Un cadru de mapare multiprogram

Drepturi de acces, bazate pe pagini

Mecanismul de paginare poate furniza baza pentru protecția memoriei într-un spațiu logic de adrese. Fiecare pagină poate avea asociate atribute (denumite şi drepturi de acces) care indică modul în care se poate obține acces la pagină. Aceste atribute pot permite numai citire, citire / scriere sau pot împiedica orice acces.



Protecția bazată pe pagini

Memoria virtuală

Spațiul de adrese logice este mult mai mare decât memoria fizică. Memoria virtuală este un mecanism pentru a extinde limitele memoriei fizice. Într-un sistem cu memorie virtuală, aceasta apare utilizatorului ca şi cum întregul spațiu logic de adrese este disponibil pentru memorare. Dar, de fapt, doar câteva pagini din spațiul logic de adrese sunt mapate peste spațiul fizic la un moment dat. Alte pagini nu sunt prezente în memoria principală; în schimb, informația din aceste pagini este memorată într-o memorie secundară, cum ar fi discul, al cărei cost pe bit este mult mai economic.

Ori de câte ori se obține acces la o pagină care lipseşte, softul sistemului de operare încarcă pagina respectivă de pe disc şi memorează pe disc o altă pagină, la care nu s-a făcut recent referire. Utilizatorul are impresia unei memorii fizice uriaşe, dar mai lente.

Memoria segmentată

O altă formă de organizare a memoriei logice este memoria segmentată. Motivația acesteia o reprezintă faptul că programele nu sunt scrise ca o secvență liniară de instrucțiuni şi date, ci mai degrabă ca bucăți (secvențe) de cod şi bucăți de date. De exemplu, poate exista o secțiune principală

de cod şi mai multe proceduri separate. Aceste module de cod şi date pot fi de diferite dimensiuni. Spațiul logic de adrese este despărțit în mai multe spații liniare de adrese, fiecare având o anumită dimensiune. Fiecare dintre aceste spații de adrese liniare este denumit segment. Fiecare element dintr-un segment este accesibil printr-o adresă cu două componente:

- selectorul segmentului, care specifică adresa de început a segmentului;
- deplasamentul, care specifică adresa relativă, față de baza segmentului, a elementului selectat.

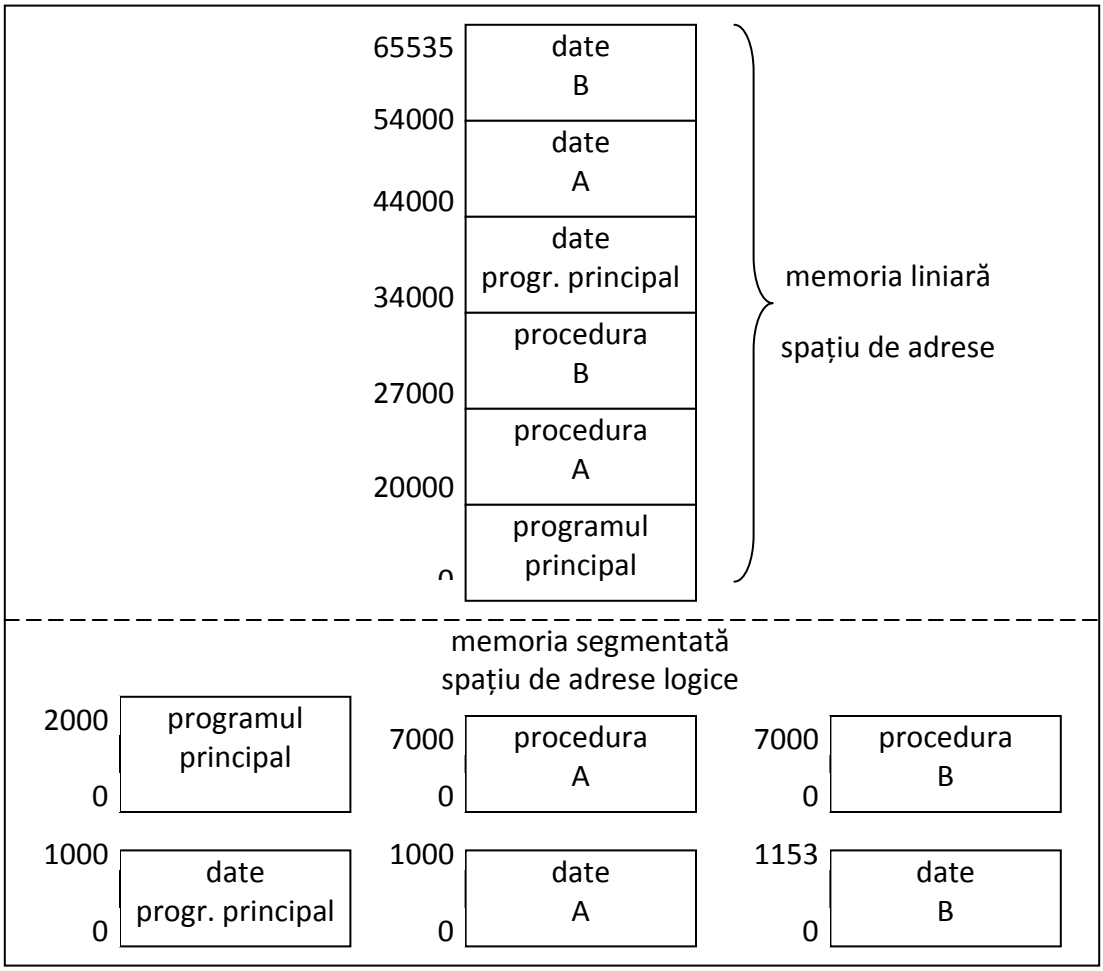
Fiecare segment poate fi asociat unui modul de date sau de program. Astfel, programul poate avea procedura principală într-un segment, celelalte proceduri în segmente distincte, şi fiecare structură importantă de date în segmentul său. Astfel, structura adreselor logice reflectă organizarea logică a programului.

Mecanismele de protecție pentru memoria liniară sunt bazate, de obicei, pe pagini de lungime fixă, a căror dimensiune este determinată pe criterii hardware şi nu au vreo relație cu structura logică a programului.

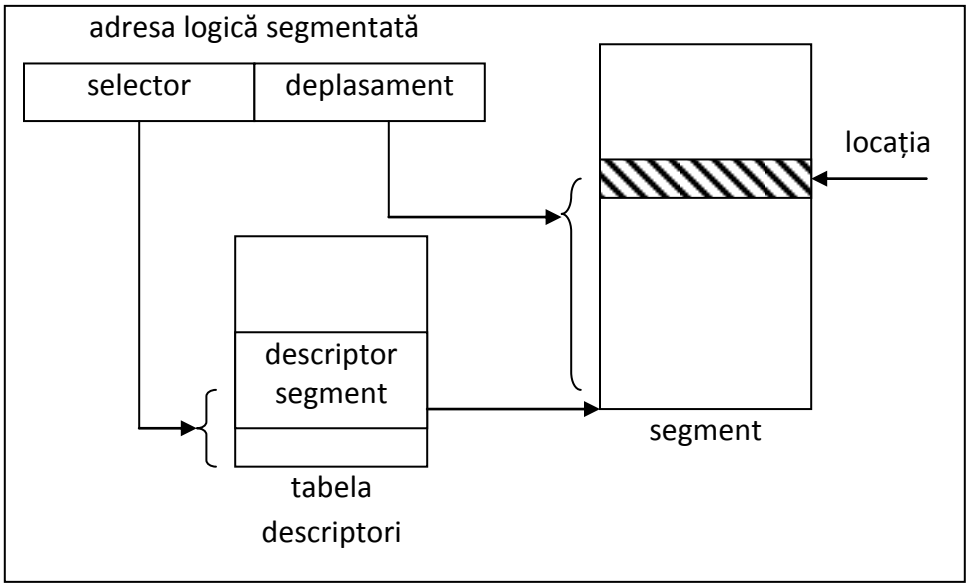
Problema descompunerii spațiului de adrese logice în pagini este că mecanismul de protecție nu poate proteja cu exactitate modulul de program. Dimensiunea paginii este determinată din rațiuni hardware, deci nu are legătură cu structura logică a programelor. În schimb, întrucât fiecare segment are o anumită lungime, este uşor să-l protejăm de alte programe. Mecanismele de memorie virtuală pot fi implementate şi pentru arhitecturi segmentate. În acest caz, segmentul este unitatea de memorie ce se interschimbă (swapping) la şi de la memoria externă.

Maparea memoriei segmentate

Maparea, în acest caz, este implementată printr-o tabelă de segment, care păstrează un descriptor pentru fiecare segment. Descriptorul conține adresa de început a segmentului şi lungimea acestuia. Componenta selectorului de segment a unei adrese logice este utilizată ca un index pentru a selecta descriptorul în tabela (descriptorilor) de segment. Apoi deplasamentul este adunat la adresa de start a segmentului, furnizată de descriptor, pentru a calcula adresa fizică a operandului referit. Deplasamentul este verificat hardware, pentru a exista siguranța că referința nu depăşeşte lungimea segmentului.



Compararea memoriei liniare cu cea segmentată



Maparea memoriei segmentate

Tipuri de segmente şi drepturi de acces

Descriptorul de segment mai conține, pe lângă adresa de bază a segmentului şi limita sa, şi atribute referitoare la tipul segmentului. Drepturile de acces sunt, deci, asociate, în particular, fiecărui segment, în ciuda faptului că modulele programului referă acele segmente. Dacă un segment este de tip read-only (numai pentru citire), de exemplu, îşi menține tipul pentru toate modulele care fac referire la acesta. Această asociere a drepturilor de acces cu segmentele este un dezavantaj, deoarece putem dori să dăm diferitelor module acces la acelaşi segment, dar cu diferite drepturi de acces.

Controlul accesului

Un alt dezavantaj al mecanismului de mapare a segmentelor este dificultatea de a limita accesul unui program la segmentele altui program. Deoarece tabela de segmente conține toți descriptorii de segment, orice program poate accesa orice segment prin simpla indexare în tabela de segmente. Soluția este ca fiecare program să aibă propriile sale tabele. Dar aceasta înseamnă că, ori de câte ori un segment este realocat în memoria fizică, toate programele partajează segmentul vor trebui să actualizeze descriptorii lor de segment. Pentru a reduce numărul operațiilor de acest gen, fiecare task vede spațiul de memorie divizat în două: un spațiu global, care conține serviciile de sistem, sistemul de operare şi un spațiu local asociat fiecărui task.

Memoria virtuală şi alocarea dinamică a memoriei

Fiecare modul de program are propriul său cadru de acces, cadru în care (de exemplu, 32 de linii de adresă) se pot genera 232 adrese diferite. Deci spațiul total de adrese, cel virtual, este egal cu dimensiunea maximă a unui segment înmulțită cu numărul total de segmente. De exemplu la 386/486 şi Pentium: cu o magistrală de adrese de 32 de biți şi un selector de 16 biți, dintre care 14 sunt utilizați pentru adresare, rezultă un spațiu total de adresare virtuală de 64 To. Fiecare descriptor de segment trebuie să conțină un câmp de biți, actualizați hardware, utilizați de sistemul de operare pentru a implementa memoria virtuală:

- valid (sau prezent), identifică dacă segmentul se găseşte în acest moment în memorie;
- memorie alocata, indică dacă s-a asociat memorie cu acest descriptor;
- accesat, indică dacă segmentul a fost accesat de un program;
- modificat, indică dacă informația din segment a fost modificată sau nu de un task.

Sistemul de operare poate utiliza aceşti biți:

- valid / memorie alocată, pentru a detecta când un segment fizic nu este prezent în memorie;
- accesat / modificat, pentru a decide care dintre segmentele existente în acest moment ar trebui interschimbat (swapped) sau pur şi simplu, dacă n-au fost modificări, scrierea segmentului nou peste cel vechi (neutilizat curent).

În plus, câteva câmpuri din descriptor pot fi utilizate de sistemul de operare, pentru a memora alte informații folositoare despre segment (de exemplu: frecvența de utilizare), care pot fi utilizate în algoritmul de interschimb.

Cantitatea de memorie alocată unui modul nu trebuie să fie fixă la momentul compilării; ea se poate modifica dinamic. Mecanismul de alocare dinamică a memoriei face memoria virtuală mult mai eficientă prin alocarea segmentelor numai atunci când este necesar, împiedicând astfel utilizarea memoriei de către segmentele la care nu se face referire.