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

8. Segmentare şi paginare. Spațiul de I/O.

Segmentarea

Să considerăm un sistem multitasking, în care sunt rulate 4 task-uri (aplicații), definite astfel:

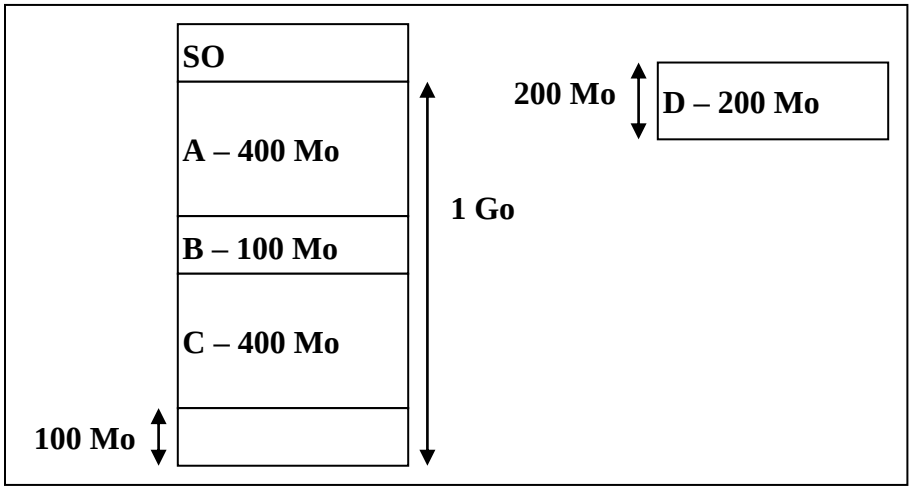
A – necesită 400 Ko de memorie;
B – necesită 100 Ko de memorie;
C – necesită 400 Ko de memorie;
D – necesită 200 Ko de memorie;

Sistemul are disponibil 1 Mo de memorie pentru execuția aplicațiilor. Presupunem că jumătate din spațiul unei aplicații reprezintă programul în sine, iar cealaltă jumătate reprezintă date. Deoarece necesitățile de memorie ale celor 4 aplicații depăşesc spațiul disponibil de 1 Mo, ele nu se pot găsi simultan în memorie.

După ce se încarcă A, B şi C, nu mai rămâne spațiu disponibil pentru tot task-ul D (figura de mai jos).

Sistemul de operare poate încărca porțiunea de cod a task-ului D (100 Ko), dar nu şi segmentul de date. El creează descriptori atât pentru cod, cât şi pentru datele task-ului D, marcând descriptorul de segment de date ca non-prezent (NP=0).

Acesta este un sistem multitasking, astfel că adresa de start (CS:EIP) a fiecărui task este transmisă către porțiunea planificatorului din sistemul de operare, după care începe execuția. Task-ul A începe execuția şi este executat timp de câteva milisecunde. Planificatorul îi ia apoi controlul şi permite execuția task-ului B, timp de câteva milisecunde. Totuşi, pe durata alocată, task-ul B citeşte o intrare, a operatorului, de la tastatură; deoarece nici o tastă nu a fost apăsată, sistemul de operare preia controlul şi marchează task-ul B ca fiind suspendat. Apoi lanificatorul cedează controlul task-ului C, pe durata de timp alocată, după care se va transfera controlul task-ului D. Acesta începe execuția, dar imediat ce încearcă referirea la segmentul de date, procesorul generează întreruperea non-prezent (NP). Sistemul de operare determină task-ul care se executa la apariția întreruperii, precum şi motivul întreruperii. El va stabili că task-ul D necesită acces la segmentul său de date, astfel că evaluează stările celorlalte task-uri. Întrucât task-ul B este suspendat, sistemul de operare decide să-l înlocuiască temporar din memorie pentru a face loc segmentului de date al task-ului D.

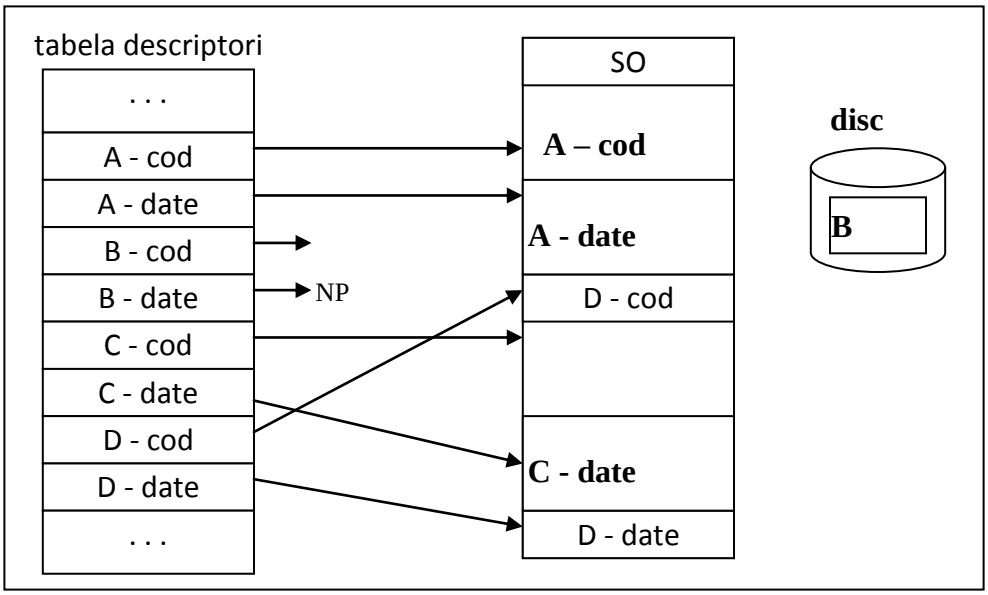


Task-urile inițiale, încărcate în memorie

Imaginea din memorie a lui B este scrisă pe disc, iar descriptorul pentru B este marcat ca non-prezent. Se spune că task-ul B a fost scos (*swapped out*), iar sistemele de operare care implementează memoria virtuală într-o astfel de manieră implementează interschimbarea (*swapping*).

Segmentul de date pentru D este copiat în memorie la adresa fizică eliberată de B, iar descriptorul pentru D este actualizat pentru a reflecta noua adresă de bază şi pentru a arăta că acest segment este acum prezent în memorie. Starea actuală a sistemului este prezentată în figura următoare.

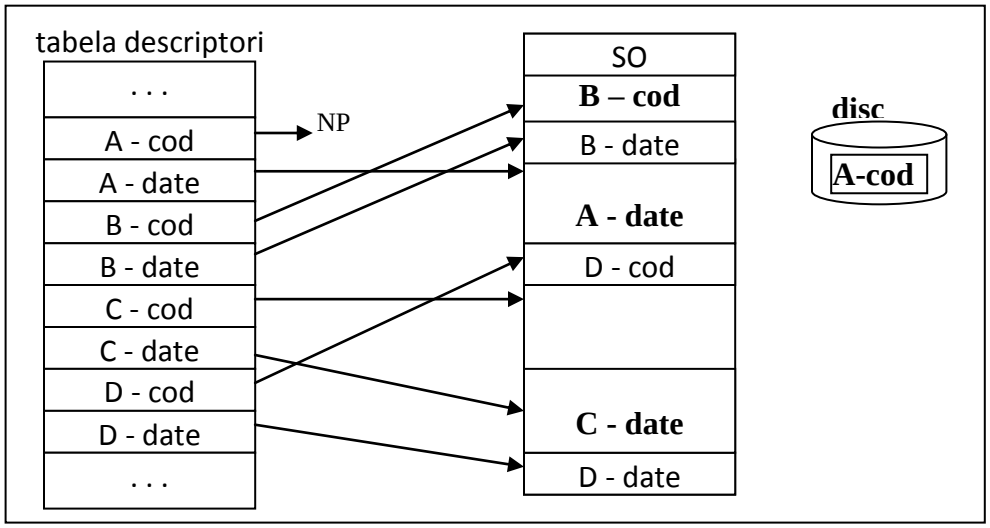
Acum planificatorul comută timpul de execuție între task-urile A, C şi D. La un moment dat operatorul, care vede promptul pentru intrarea de la task-ul B, va apăsa o tastă. Această acțiune determină o întrerupere hard şi sistemul de operare observă că acum trebuie să planifice task-ul B. Totuşi, deoarece nici unul dintre celelalte task-uri nu este suspendat, sistemul poate alege să suspende temporar task-ul A. Deoarece task-ul B este mic, el înlocuieşte doar o parte din A. Segmentul de cod al task-ului A este marcat ca non-prezent, task-ul B este interschimbat, iar descriptorii pentru task-urile A şi B sunt actualizați corespunzător.



Interschimbarea task-urilor B şi D

Task-ul B rulează acum la o adresă fizică diferită de adresa de început. Aceasta este invizibilă aplicației, totuşi, deoarece selectoarele încărcate în registrele segment nu se modifică şi, deoarece offseturile de memorie utilizate de instrucțiuni în segmentul de cod sunt relative față de punctul de start al segmentului, originea fizică a segmentului nu are importanță.

Sistemul va continua să opereze ca mai sus; dacă nu există condiții externe care să determine interschimbarea de segment, sistemul de operare poate interschimba segmente, bazându-se fie pe durata de rulare a celui mai lung task, fie pe un alt sistem de priorități.



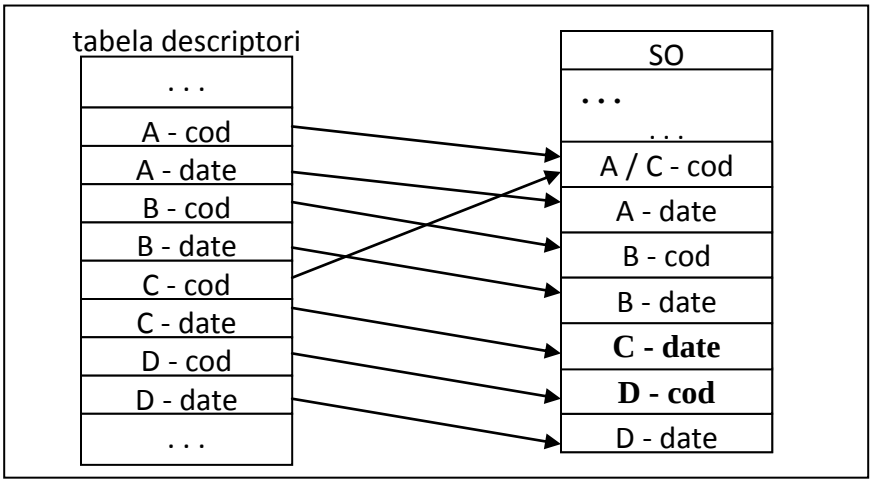
Interschimbarea task-urilor A şi B.

Considerații de performanță

Memoria virtuală utilizează memoria secundară (de obicei discul) pentru a suplini memoria principală (RAM) şi dă senzația unei cantități de memorie principală superioară celei existente în sistem. Costul pentru aceasta este timpul necesar pentru a muta datele între memoria principală şi cea secundară. Cu cât timpul de interschimbare este mai mare, cu atât este mai mic timpul pentru execuția aplicațiilor.

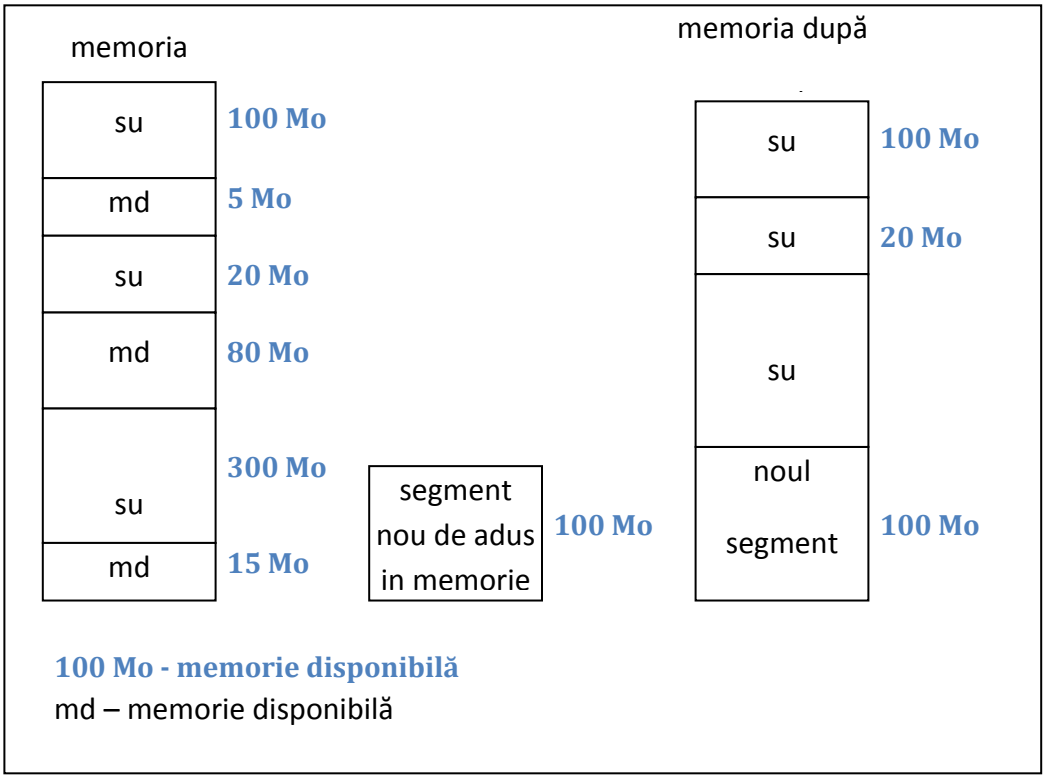
Performanța sistemului de operare se poate îmbunătăți. De exemplu, la mecanismul de protecție de la Intel, segmentele de cod sunt nemodificabile. Din acest motiv ele nu trebuie scrise pe disc când sunt extrase din memorie. Este posibilă refacerea conținutului pornind de la imaginea executabilă originală a programului (de pe disc). Este necesar doar accesul la memoria secundară pentru interschimbarea lui în memorie. Rezultă că sistemul de operare va interschimba de două ori mai repede segmentele de cod decât cele de date. De asemenea, unele segmente de date pot fi marcate, în descriptorul lor, ca read-only. Deci, nu este necesară nici salvarea acestora pe disc la interschimbarea lor.

O altă facilitate se referă la tehnica partajării segmentului, care permite ca două sau mai multe task-uri să partajeze acelaşi cod. Aceasta este eficientă în sistemele multiutilizator. Să presupunem, în exemplul anterior, că task-urile A, B, C şi D reprezintă utilizatori ce rulează aplicații. Să presupunem că utilizatorii A şi C rulează aceeaşi aplicație, de exemplu o foaie de calcul. Cei doi utilizatori operează pe date diferite şi necesită segmente de date separate. Ei execută, totuşi, acelaşi cod. În figură se arată cum toate cele 4 aplicații pot intra în memoria fizică, în acest caz. Utilizatorii păstrează descriptori separați pentru codurile şi datele lor, dar adresele de bază ale segmentelor de cod pentru A şi C fac referire la aceeaşi locație.



De asemenea, un sistem de memorie virtuală, orientat pe segmente, poate furniza un mod pentru a compacta memoria. Compactarea memoriei permite rezolvarea unei probleme numite fragmentare. Fragmentarea memoriei apare când memoria discontinuă este disponibilă pentru a rula aplicații suplimentare. Cu alte cuvinte, mici porțiuni de memorie disponibilă sunt disipate în memoria fizică, ca în figura de mai jos. Pentru a putea fi folosite este necesar ca aceste porțiuni să fie contigue.

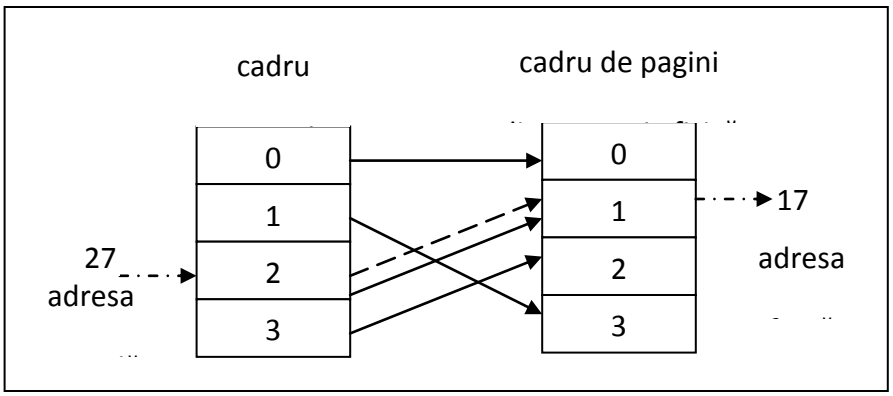
Deoarece aplicațiile operează cu adrese virtuale, ele nu sunt afectate de o modificare a locației de start. Procesul acesta consumă, totuşi, timpul procesorului.



Fragmentarea memoriei

Paginarea

Paginarea este utilizată pentru a implementa memoria virtuală bazată pe blocuri de dimensiune fixă, denumite pagini. Asemănător segmentării, paginarea translatează adresele virtuale în adrese fizice. Acestea sunt translatate prin maparea (suprapunerea) blocurilor de memorie de lungime fixă în locațiile de memorie fizică, denumite cadru pagină. Segmentarea şi paginarea sunt similare: un nume și un offset sunt translatate într-o adresă. Să considerăm o memorie fizică compusă din paginile 0, 1, 2, 3, fiecare având câte 10 octeți; pentru convertirea unei adrese virtuale într-una fizică se utilizează o tabelă de pagini. De exemplu, adresa virtuală 27 este translatată în adresa fizică 17:



Translația adresei virtuale în elemente de dimensiune fixă

Avantaje şi dezavantaje

Avantajul esențial al paginării față de segmentare este reprezentat de dimensiunea fixă a paginării. Deoarece pentru memoria virtuală se utilizează discul se pot alege dimensiuni de pagină care corespund dimensiunii sectorului de pe disc. De asemenea, paginarea evită problema fragmentării memoriei, specifică segmentării. Ori de câte ori o pagină este scoasă din memorie, altă pagină se încadrează exact în spațiul eliberat.

Un alt avantaj este că alocarea pentru un obiect mare nu trebuie realizată într-un spațiu continuu de memorie. Paginarea este invizibilă programatorului. Spre deosebire de segmentare, care necesită cunoaşterea numelui virtual (segment) şi a unui offset pentru un obiect din memorie, paginarea necesită cunoaşterea doar a unei adrese. Adresa virtuală este descompusă în componentele sale de mecanismul memoriei virtuale.

Utilizarea paginării înseamnă pierderea inelelor de protecție, implementate cu segmentarea.

De asemenea, ea este legată de un alt tip de fragmentare, denumit „fragmentare internă“, care apare la memorarea unor obiecte care nu se încadrează într-o pagină sau secvență de pagini. De exemplu, dacă pagina este de 10 octeți, un obiect de 11 octeți necesită două pagini, care irosesc, deci, spațiu de memorie.

Paginarea necesită mai multă memorie. La segmentare, tabela de translație, pentru conversia unei adrese virtuale într-una fizică este necesară când se încarcă un nou segment. La paginare, însă, conversia (translația) trebuie realizată pentru fiecare acces la memorie; acesta nu este un inconvenient dacă întreaga tabelă de pagini poate fi memorată în procesor, dar necesită tabele de pagini foarte mari.

Pentru a depăşi aceste probleme se poate implementa o schemă de protecție simplă, care utilizează numai paginarea, sau se pot utiliza în paralel cele două mecanisme: paginare şi segmentare.

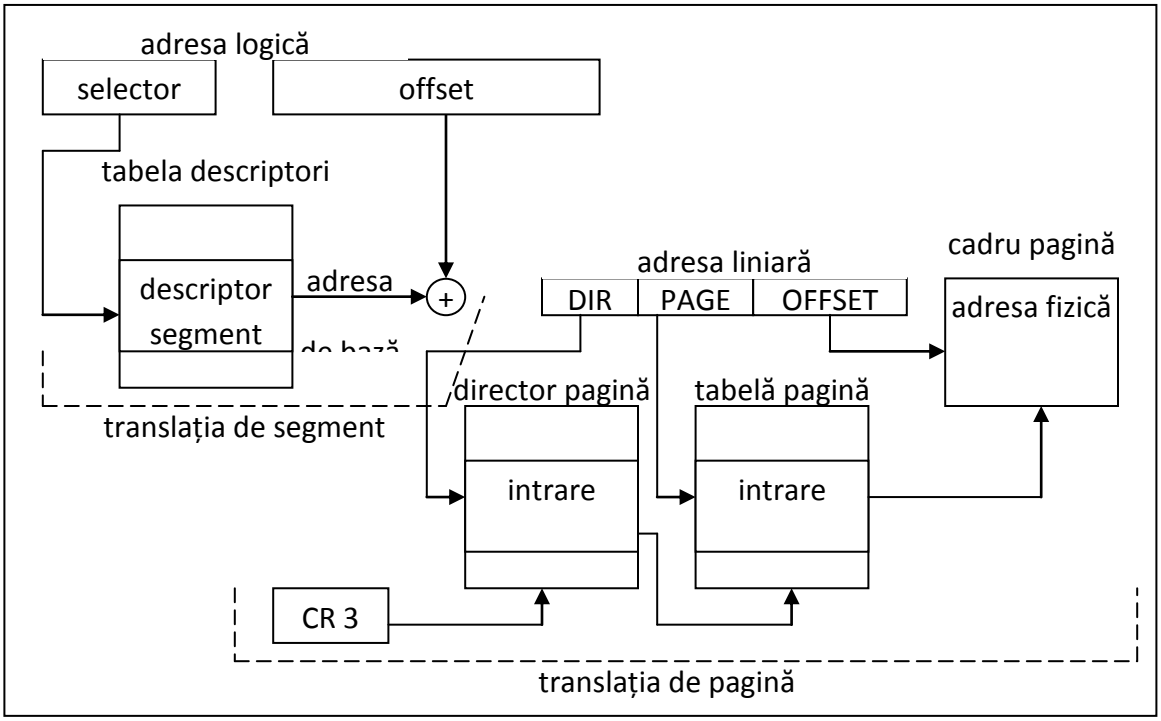
Pentru a reduce spațiul necesar translației de pagini (de fapt, fragmentarea internă) se utilizează paralelismul intern al procesoarelor şi o memorie cache specială, denumită „buffer de translație adrese“ (TLB-Translation Lookahead Buffer), ce nu trebuie confundată cu memoria cache internă a procesoarelor 486 / Pentium.

Translația paginii

Cea de-a doua etapă, a translației de adresă, (prima fiind cea de segmentare) – transferă adresa liniară într-o adresă fizică. Aceasta implementează caracteristicile de bază necesare pentru sistemele de memorie virtuală, orientate pe pagină, şi protecția la nivel de pagină.

Un *cadru pagină* este o unitate de 4 Ko de adrese continue din memoria fizică.

O adresă liniară face referire, indirect, la o adresă fizică, prin specificarea unei tabele de pagini, o pagină în cadrul acestei tabele şi un offset în acea pagină. Mecanismul complet, de segmentare şi paginare, este prezentat în figură:



Mecanismul translației adresei logice în adresă fizică

Din figură se poate observa cum procesorul converteşte câmpurile *DIR*, *PAGE* şi *OFFSET* ale unei adrese liniare în adresă fizică, prin consultarea a două tabele de pagini:

- câmpul *DIR* este un index într-un director de pagini;
- câmpul *PAGE* este un index într-o tabelă de pagini, determinată de directorul de pagină;
- câmpul *OFFSET* este utilizat pentru a realiza o adresare în pagina determinată de tabelele de pagină.

O tabelă de pagini este un tablou de specificatori de pagină, de 32 de biți. O tabelă de pagini este ea însăşi o pagină, deci conține 4 Ko, adică 1K intrări de câte 32 de biți.

Întrucât acest mecanism utilizează două niveluri de tabele, director de pagini şi tabele de pagini, fiecare tabelă putând adresa 1K, rezultă că toate tabelele adresate de un director de pagină pot adresa 1M pagini.

Întrucât fiecare pagină conține 4 Ko, rezultă că tabelele unui director de pagină pot acoperi întregul spațiu fizic de adrese de 4 Go.

Adresa fizică a directorului de pagină curent este memorată în registrul CR3, denumit şi registru de bază al directorului de pagină (PDBR – Page Directory Base Register). Softul de administrare a memoriei poate opta între utilizarea unui director de pagină pentru toate task-urile, un director de pagină pentru fiecare task, sau o combinație a acestor două soluții.

Organizarea spațiului de I/O

Procesorul are două spații de adrese fizice distincte: memoria şi I/O (intrări/ieşiri). Dispozitivele periferice (consola, ecranul, tastatura, imprimanta etc.) sunt controlate şi comandate de procesor, în general, prin operații de citire/scriere la anumite locații, diferite de spațiul de memorie.

Acest spațiu este denumit spațiu de I/O, iar accesul la locațiile respective se face cu instrucțiuni specifice (In, Out). Locațiile din acest spațiu sunt denumite registre dispozitiv sau porturi de I/O.

În general, perifericele sunt plasate în acest spațiu, deşi procesorul poate permite maparea (suprapunerea) în memorie a perifericelor.

Spațiul de I/O constă din 64 Ko, şi poate fi împărțit în 64 Kporturi de 8 biți, 32 Kporturi de 16 biți sau, la 386/486, 16 Kporturi de 32 biți. Pentru a obține acces la acest spațiu nu se utilizează registre segment şi implicit nici mecanismul de segmentare sau pagină. Pinul M/IO specifică spațiul de memorie adresat (fizică sau I/O), deci dacă se adresează o locație de memorie sau un port de I/O. Instrucțiunile I/O, IN şi respectiv OUT, pot furniza adresa direct în instrucțiune, ca o constantă de 8 biți (deci pentru porturile din spațiul 0-255), sau indirect prin registrul DX (deci pentru tot spațiul de 64 K).

Pe lângă realizarea operațiilor de ieşire utilizând instrucțiunile specifice spațiului de I/O, se mai pot realiza transferuri de date prin acces direct la memorie (DMA-Direct Memory Access). DMA este mod special de organizare care constă în maparea spațiului de I/O peste memorie. În acest mod, echipamentele periferice citesc şi scriu direct în memorie, fără să se mai treacă prin procesor. Pentru echipamentele foarte rapide, transferul prin intermediul procesorului (care se realizează element cu element) poate fi prea lent. Pentru aceste situații a fost prevăzut mecanismul de acces direct la memorie DMA. Acesta permite desfăşurarea în paralel de alte activități ale procesorului, inclusiv de I/O, pentru a mări viteza de lucru a întregului sistem. Totuşi, cele două module, procesorul şi DMA, nu pot utiliza simultan magistralele de date şi adrese. Deci, prelucrarea paralelă poate să apară doar dacă procesorul are memorie cache şi execută linii de program sau obține acces la date din aceasta, fără a utiliza magistrala. Chiar dacă procesorul trebuie să aştepte ca DMA să finalizeze un transfer, acesta este executat mai rapid de DMA decât sub controlul procesorului, fie în spațiul de I/O propriu, fie în cel mapat peste memorie.