5 Indicatorii de stare

Toate microprocesoarele conţin un număr de indicatori de stare (*flags*), grupaţi sub forma unui registru special (F) asociat, de regulă, unităţii aritmetice şi logice. Din punct de vedere "hard", un indicator de stare este un circuit basculant bistabil independent iar din punct de vedere "soft", este o variabilă logică de supraveghere a unei stări (şi numai una), care ia valoarea 0 sau 1.

Fiecare indicator are o stare activă (de regulă 1) şi una pasivă (0). De exemplu, steguleţul galben al unui arbitru de tuşă (la fotbal), este menţinut *coborât* cât timp balonul este în teren (stare pasivă) şi este *ridicat* imediat ce balonul iese în afara terenului de joc (stare activă).

La iniţializarea microprocesorului, toţi indicatorii trec automat în starea pasivă; în timpul execuţiei programului un indicator basculează în starea activă când se produce evenimentul specific sarcinii sale. Valoarea logică a fiecărui indicator poate fi testată direct sau indirect, prin intermediul instrucţiunilor. De regulă, în funcţie de valoarea unui indicator de stare se ia o decizie cu privire la modul în care se vor desfăşura prelucrările următoare.

Valorile logice ale indicatorilor de stare formează "cuvântul de stare" al microprocesorului, care are ca sediu "registrul de stare"; acesta are diferite denumiri date de proiectanţii de microprocesoare:

- Registrul cuvântului de stare;
- Registrul de condiţii - CR (*Condition Register*);
- Cuvântul de stare al programului - PSW (*Program Status Word*);
- Registrul de control - CR (*Control Register*);
- Registrul F (*Flags*);

Iată indicatorii de stare prezenţi la microprocesoarele de 8 biţi, Intel 8080, Intel 8085, Zilog - Z80, preluaţi ulterior şi de microprocesoarele evoluate, alături de noi indicatori:

1. Indicatorul de zero (*Zero*): testează dacă rezultatul unei operaţii aritmetice este nul (Z=1 dacă rezultatul este 000...0);

2. Indicatorul de transport (*Carry*): marchează apariţia unui bit de transport (depăşirea lungimii normale a rezultatului) la adunare sau de împrumut la scădere;

3. Indicator de semn (*Sign*): arată semnul rezultatului unei operaţii efectuate între operanzi cu semn (S=0 pentru "+" şi S=1 pentru "-");

4. Indicatorul de depăşire a capacității (*Overflow*): semnalizează valorile interzise ale rezultatului în cazul operațiilor în complement față de 2;

5. Indicator de transport la jumătate (*Auxiliary Carry*): marchează apariția unui bit de transport intermediar (între doi digiți compactați); este util când se lucrează în cod BCD;

6. Indicator de paritate (*Parity*): arată dacă numărul cifrelor binare "1" dintr-un cuvânt este par (P=1) sau impar (P=0).

Evoluția microprocesoarelor a impus introducerea de noi indicatori care au fost adăugați la lista de bază. La Intel 386, de exemplu, apar 7 indicatori în plus:

7. Indicatorul de funcționare "pas cu pas" (*Trap*): intervine în cazul rulării pas cu pas a programului, pentru depanare;

8. Indicatorul de întreruperi (*Interrupt*): autorizează sau interzice tratarea întreruperilor mascabile;

9. Indicatorul nivelului de privilegiu (*I/O Privilege Level*): stabileşte nivelul de protecție al unui program şi drepturile de acces la memorie;

10. Indicatorul de direcție (*Direction*): validează sensul de examinare (prin incrementarea sau prin decrementarea adresei) pentru un şir de operanzi din memorie;

11. Indicatorul de imbricare a task-urilor (*Nasted Task*): intervine în modul de lucru protejat;

12. Indicatorul de reluare (*Resume*): se utilizează în operațiile de depanare a programelor;

13. Indicatorul modului virtual (*Virtual Mode*): pentru modul protejat.

La Intel 486 se adaugă încă un indicator:

14. Indicatorul de aliniere (*Alignment Check*): sesizează alinierea datelor stocate la adrese divizibile cu un anumit număr.

Lista poate continua încă destul de mult dacă am dori să ținem cont de specificitatea tuturor microprocesoarelor, însă ea conține deja tot ceea ce este important.

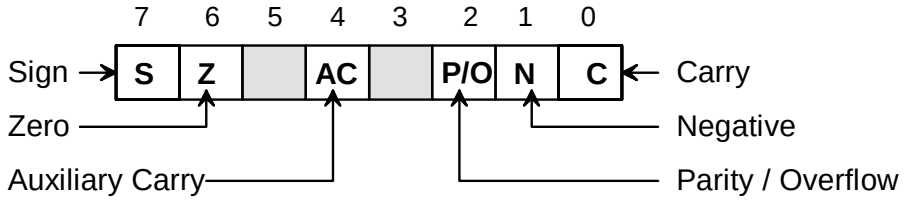


Fig. 1 Registrul indicatorilor de condiții la microprocesorul Z80

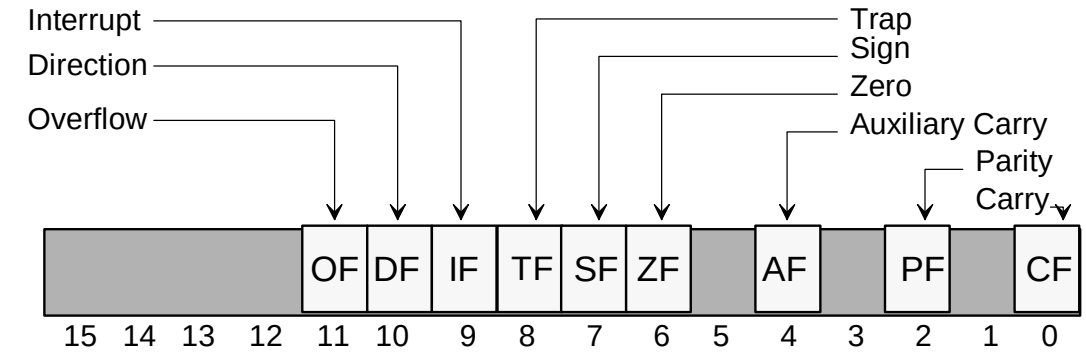


Fig. 2 Registrul indicatorilor de condiții la microprocesoarele **Intel 8086/8088**

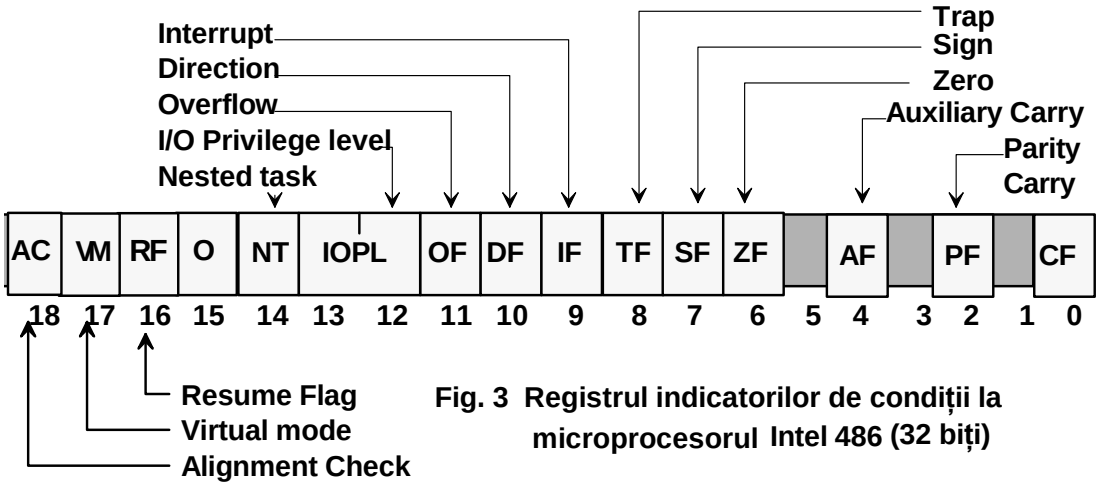


Fig. 3 Registrul indicatorilor de condiții la microprocesorul Intel 486 (32 biți)

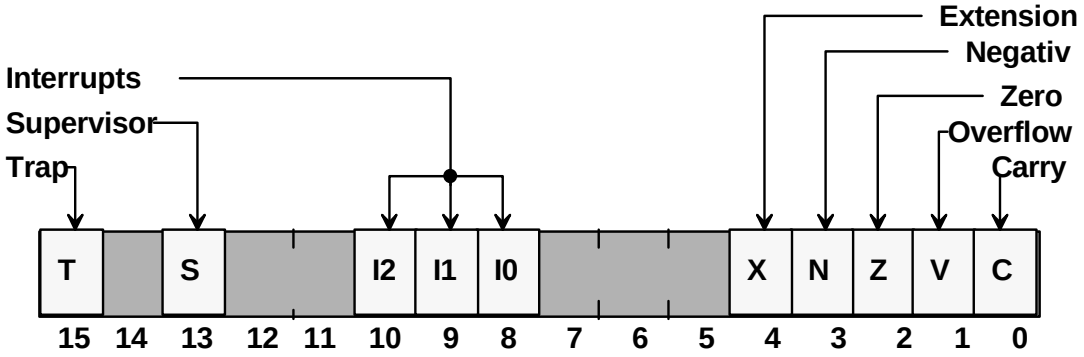


Fig. 4 Registrul indicatorilor de condiții la Motorola 68 000

Sunt prezentați în continuare, în detaliu, indicatorii de bază care necesită informații suplimentare privind rolul lor în funcționarea microprocesoarelor.

5.1 Indicatorul Z (Zero)

El supraveghează rezultatul unei operații aritmetice sau logice: are starea pasivă "0" şi se poziționează automat în "1" când rezultatul ultimei operații aritmetice este nul (0 0 0 0 . . . 0). Este de remarcat că nu conținutul unui registru poziționează indicatorul Z, ci **numai rezultatul** unei operații aritmetice (dacă se încarcă în acumulator, de exemplu, un număr nul, indicatorul Z nu comută în "1", pentru că operația de transfer nu este aritmetică sau logică).

Una din aplicațiile tipice ale indicatorului Z este supravegherea unui registru numărător, care este decrementat la fiecare operație elementară executată; când numărătorul ajunge la 000. . . 0, procesul trebuie oprit, deoarece s-au executat toate operațiile programate.

Pentru stabilirea momentului opririi, se testează după fiecare etapă indicatorul Z ; dacă este "0", se continuă procesul repetitiv iar dacă este "1", procesul trebuie oprit.

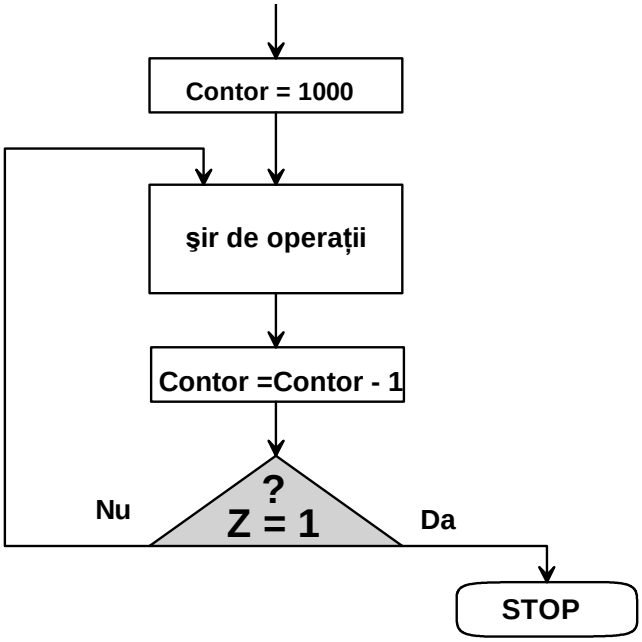


Fig. 5 Organigrama unei secvențe de testare a valorii zero pentru un contor

Pentru a detecta dacă un rezultat este nul, se utilizează în mod curent o poartă SAU cu un număr de intrări egal cu dimensiunea registrului în care se află rezultatul. Ieşirea porții SAU poziționează în "1" circuitul bistabil Z, dacă toate intrările sunt în "0" logic.

Observație: în mod natural, o incrementare (se adună 1) sau o decrementare (se scade 1) sunt operații aritmetice; la unele microprocesoare însă aceste operații nu afectează indicatorul Zero.

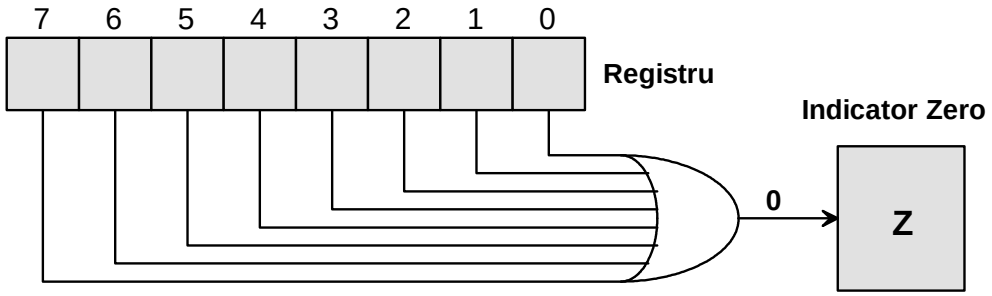


Fig. 6 Poziționarea indicatorului Z când registrul are conținut nul.

5.2 Indicatorul de transport C (Carry)

Atunci când se execută o adunare, este posibil să rezulte un transport spre rangul superior, care depăşeşte dimensiunea registrului ce conține rezultatul. Bitul de transport nu se pierde, deoarece el va poziționa în "1" un indicator, cel de transport C. În mod asemănător, la scădere apare necesitatea unui împrumut la rangul superior, care poziționează în "1" logic indicatorul C.

```
  0100 1000+          1000 1000+
  0000 0011           1000 0011
 ----------          ----------
= 0100 1011         = [1] 0000 1011
```

adunare fără transport (Carry=0) **adunare cu transport (Carry=1)**

Rezultatul corect al unei adunări sau scăderi include întotdeauna şi indicatorul Carry.

Indicatorul Carry este strict necesar când se efectuează operații de adunare sau scădere între numere reprezentate pe mai mulți octeți decât permite unitatea aritmetică. În aceste cazuri, operația se efectuează octet cu octet (sau cuvânt cu cuvânt) şi se include în operație valoarea precedentă a lui Carry (fig. 7).

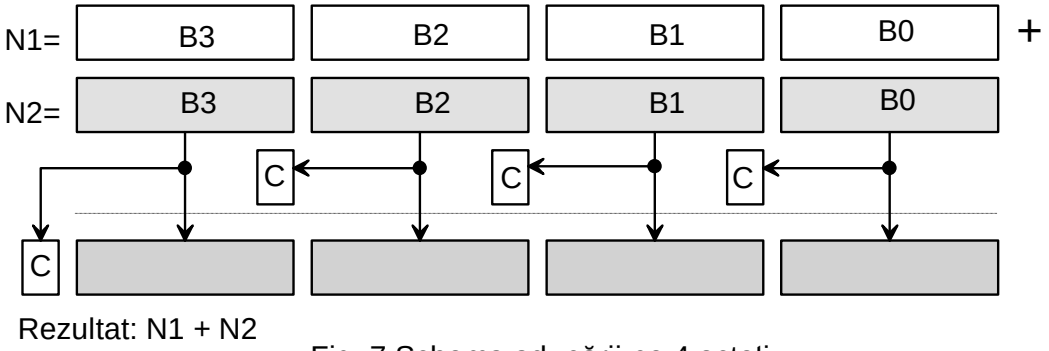


Fig. 7 Schema adunării pe 4 octeți

Pentru operațiile de adunare şi scădere, în setul de instrucțiuni al microprocesoarelor există câte două variante de instrucțiuni (exemplele sunt de la *Intel 8086*):

- Adunare simplă (ADD), fără a fi luat în considerație Carry;
- Adunare cu Carry (ADC), în care, la suma a doi operanzi se adună şi valoarea curentă a lui Carry (bitul de transport, 0 sau 1);
- Scădere simplă (SUB), fără a fi luat în considerație Carry;
- Scădere cu împrumut (SBB), în care, din rezultatul scăderii se scade şi valoarea curentă a lui Carry (bitul de împrumut, 0 sau 1);

5.3 Indicatorul de semn S (Sign)

Intervine în operațiile aritmetice între numere cu semn. Conform convenției, se utilizează "0" pentru + şi "1" pentru semnul minus, bitul de semn fiind cel de pe poziția cea mai semnificativă (stânga).

Indicatorul *Sign* este copia valorii bitului de semn al rezultatului unei operații aritmetice (bitul 7, la octeți sau bitul 15 la cuvinte de 16 biți). Dacă se lucrează cu operanzi fără bit de semn, indicatorul Sign va fi egal totuşi cu bitul cel mai semnificativ al rezultatului, deoarece "*el nu ştie*" dacă programatorul consideră operanzii cu sau fără semn.

Pentru a testa dacă un număr este pozitiv sau negativ (fig. 8), se examinează indicatorul S, după o operație care nu schimbă valoarea numărului (de exemplu adunare cu 0).

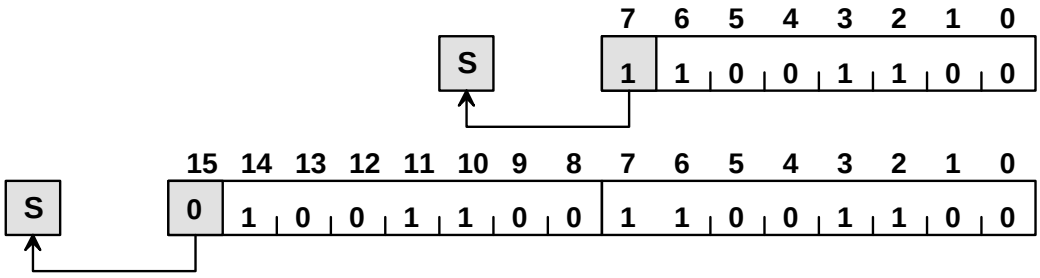


Fig. 8 Testarea indicatorului de semn

5.4 Indicatorul de depăşire (Overflow)

Când se execută operaţii aritmetice între operanzi cu semn, este posibil să apară bit de transport către bitul de semn, ceea ce face ca rezultatul să fie eronat. Indicatorul de depăşire supraveghează transportul de la bitul 6 către 7 (sau 14 către 15, etc.) sau generarea unui transport la bitul 7, 15, 31, etc.

În cazul operanzilor cu semn, în reprezentarea pe 8 biţi, domeniul valorilor este -128 . . +127. Dacă rezultatul unei operaţii aritmetice nu se încadrează în acest domeniu, el este eronat iar indicatorul de depăşire se poziţionează în "1". În cazul reprezentării pe 16, 32, 64 de biţi, domeniul este mai larg, dar de asemenea limitat iar indicatorul de depăşire are acelaşi rol.

Numerele negative se reprezintă intern în complement faţă de 2, astfel încât o scădere este înlocuită prin adunarea cu complementul.

Exemple:

0 1 1 0 0 0 0 0 +	(96 Z)	|	1 0 0 1 1 1 0 0+	(-100 Z)
0 1 0 0 0 0 0 0	(64 Z)	|	1 1 1 0 0 0 0 0	(- 32 Z)
=1 0 1 0 0 0 0 0	- 96 Z	|	= 0 1 1 1 1 1 0 0	+124

În primul exemplu, apare transport spre bitul de semn, rezultatul devine negativ, se interpretează în complement faţă de 2, se obţine -96, rezultat definitiv fals!.

În al doilea exemplu, deşi nu apare transport spre bitul de semn, rezultatul este fals deoarece apare transport din poziţia de semn, către bitul 8 (inexistent).

Analiza propagării bitului de transport de la bitul 6 la bitul 7 nu este suficientă pentru a decide că rezultatul este eronat. În exemplul de mai jos se face suma (-1) + (-1) = -2, rezultatul fiind corect, chiar dacă apare transport de la poziţia 6 la 7.

1 1 1 1 1 1 1 1+	(-1 în complement faţă de 2)
1 1 1 1 1 1 1 1	(-1 în complement faţă de 2)

= 1 1 1 1 1 1 1 0 (-2 în complement față de 2)

Algoritmul de stabilire a depăşirii:

Pentru a detecta cazurile în care rezultatul este eronat, se observă:

- starea indicatorului Carry;
- dacă există transport de la poziția 6 la poziția 7.

Să considerăm două numere A şi B, de 8 biți, reprezentate în complement față de 2 (algoritmul este valabil pentru orice dimensiune) şi C = A + B (fig. 9):

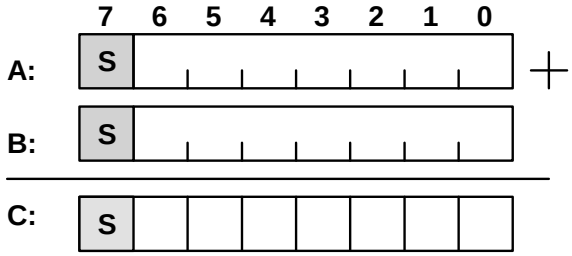


Fig. 9 Adunarea a două numere cu semn, pe 8 biți

Tab. 1

Caz	A7	B7	C7	Overflow
1	0	0	0	0
2	0	0	1	**1**
3	0	1	0	0
4	0	1	1	0
5	1	0	0	0
6	1	0	1	0
7	1	1	0	**1**
8	1	1	1	0

În tabelul de mai sus, sunt considerate toate cazurile posibile pentru biții de semn ai termenilor (A7, B7) şi bitul de semn al sumei.

Se disting trei cazuri:

1. A7 şi B7 sunt complementari: nu există depăşire de capacitate;
2. A7 = B7 = C7, nu există depăşire de capacitate;
3. A7 = B7 diferit de C7, există depăşire de capacitate.

Folosind notația **C** pentru Carry şi **R** pentru bitul de transport de la bitul 6 la 7, rezultă formula logică a indicatorului de depăşire:

$$\mathbf{O} = \mathbf{C} \oplus \mathbf{R},$$

unde operația este "sau exclusiv" iar **O** = Overflow.

Indicatorul este determinat:

- de A7, B7, C7 pentru operanzi de 8 biți;
- de A15, B15, C15 pentru operanzi de 16 biți;
- de A31, B31, C31 pentru operanzi de 32 biți;
- de A61, B61, C61 pentru operanzi de 64 biți, etc.

5.5 Indicatorul de paritate (Parity)

Controlul de paritate este o metodă elementară, care permite detectarea unor erori ce pot apărea la transferul datelor pe magistrale.

Să considerăm un octet căruia îi adăugăm al noulea bit calculat astfel:

- se numără biții de valoare "1" din octet, N;
- dacă N este par, bitul 9 ia valoarea "1";
- dacă N este impar, bitul 9 ia valoarea "0".

Cuvântul de 9 biți astfel obținut, conține totdeauna un număr impar de biți "1" logic.

- Dacă numărul de erori este par (2, 4, 6, 8), acestea nu pot fi detectate prin controlul de paritate:

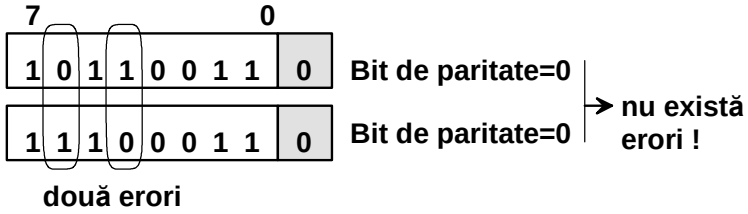


Fig. 10 Număr par de erori - nu se detectează

- Dacă numărul de erori este impar (1, 3, 5, 7), se detectează existența erorilor, fără să se poată preciza numărul biților afectați:

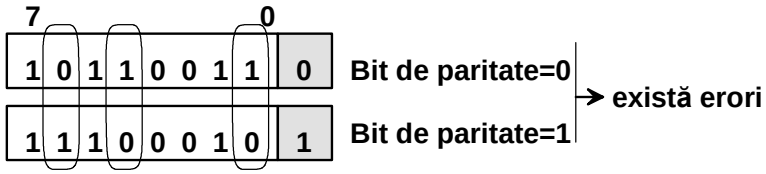


Fig. 11 Număr impar de erori - se detectează

Indicatorul de paritate a fost introdus tocmai pentru verificarea parității; el se poziționează automat în "1" sau "0", întocmai ca bitul 9 de mai sus. Programul poate testa prin intermediul instrucțiunilor bitul de paritate şi poate decide dacă un cuvânt are sau nu paritatea corectă.

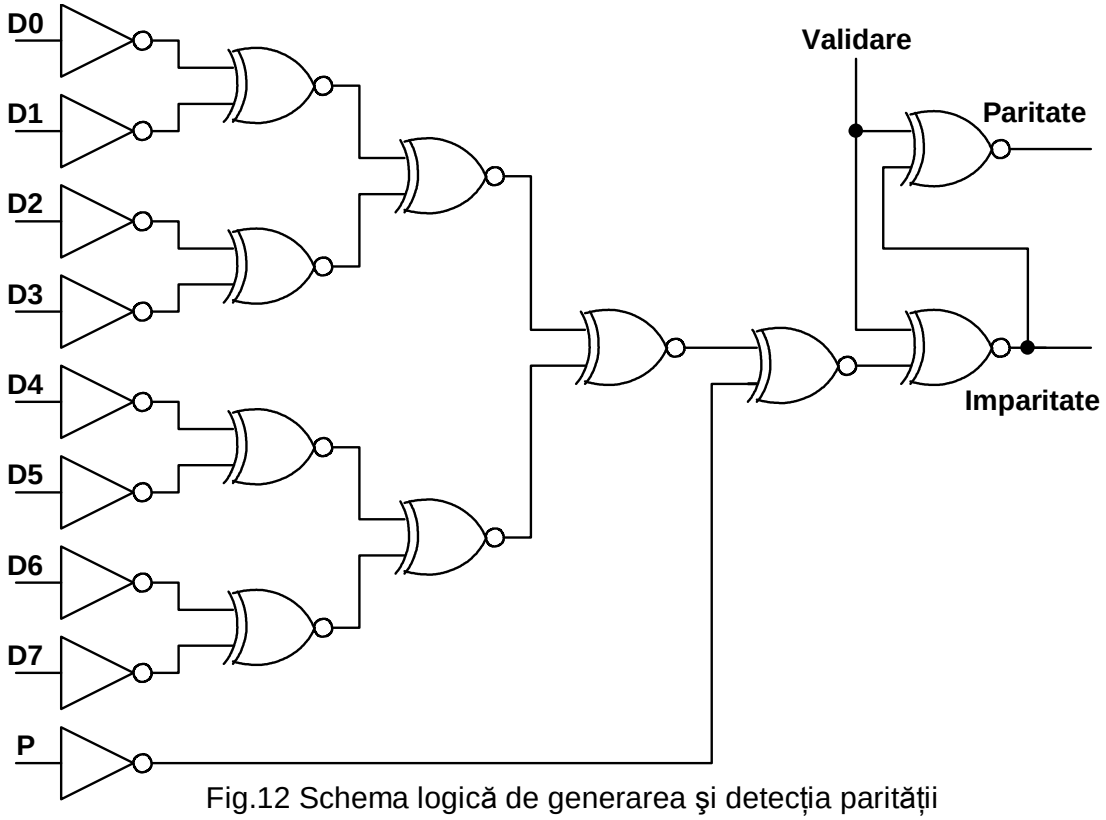


Fig.12 Schema logică de generarea şi detecția parității

Logica de generare şi de detecție a parității este foarte simplă şi utilizează porți SAU EXCLUSIV şi inversoare. În schema din fig.12 este prezentată această logică pentru date de 8 biți. Principiul este acelaşi pentru 16 sau 32 de biți.

5.6 Indicatorul de transport la jumătate AC (Auxiliary Carry)

În cod BCD (zecimal codat binar), cifrele 0, 1, . . . ,9 se codifică pe 4 biți de la 0000 la 1001. Când se execută operații în BCD, pot interveni erori dacă apare transport sau împrumut între rangurile zecimale.

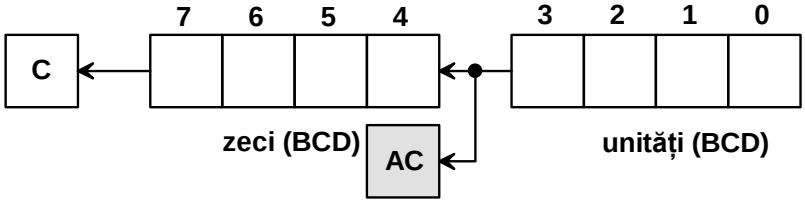


Fig. 13 Rolul indicatorului de transport la jumătate AC

Indicatorul AC memorează bitul de transport de la rangul 3 la rangul 4, în vederea corectării rezultatului după o operație aritmetică în cod BCD.

În cazul operațiilor aritmetice în BCD pot să apară 3 categorii de erori:

- depăşirea domeniului 0, . . . , 9, care se manifestă prin coduri pe 4 biți inexistente în BCD: 1010, 1011, 1100, 1101, 1110, 1111;
- transport către rangul 8, memorat de Carry;
- transport către rangul 4, memorat de Auxiliary Carry.

Toate aceste erori sunt previzibile şi corectabile, deoarece orice microprocesor execută operațiile aritmetice numai în binar natural.

Corecția rezultatului se face prin introducerea în program a unor instrucțiuni specifice de corecție, după fiecare operație aritmetică ce se execută cu operanzi exprimați în BCD.

Eroarea de cod interzis apare din cauză că în BCD nu există 6 combinații de 4 biți, dar ele pot să apară în operațiile aritmetice. Aceste combinații trebuie detectate. Principiul detecției este prezentat în fig.14

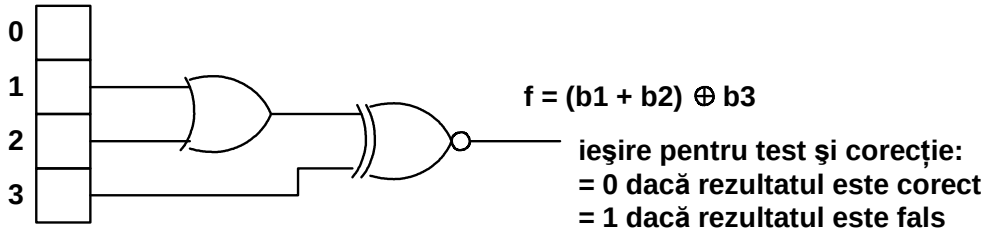


Fig. 14 Principiul de detecție a erorii de cod BCD

Dacă b3=1, biții b1 şi b2 trebuie să fie "0" pentru ca reprezentarea în BCD să fie corectă.

Eroarea generată la transport pe octet este semnalizată de setarea indicatorului Carry (C=1) şi produsă de cvartetul superior D4 - D7, care depăşeşte valoarea 1111, cod interzis.

Exemplu:	1 0 0 1 0 0 0 0 +	90+
	1 0 0 0 0 0 0 0	80
	1 0 0 0 1 0 0 0 0	**1** 10 (fals !)

Eroarea este detectată prin examinarea indicatorului Carry.

Eroarea generată la transport pe qvartet este semnalizată de setarea indicatorului Auxiliary Carry (AC=1) şi produsă de cvartetul inferior D0 - D3, care depăşeşte valoarea 1111, cod interzis.

Exemplu:	0 1 1 0 0 1 1 1+	67+
	0 0 1 0 1 0 0 1	29
	1 0 0 **1** 0 0 0 0	90 (fals !)

Eroarea este detectată prin examinarea indicatorului AC.

Tab. 2 Cele 4 cazuri de corecție pentru BCD

Caz	Situațiile posibile	Corecția se face prin adunare la rezultat:	
		în zecimal	în binar
1	C=0, AC=0 şi nu există cod interzis	0 0	0000 0000
2	C=0, AC=1, sau cod interzis la qvartet inferior	0 6	0000 0110
3	C=1, AC=0, sau cod interzis la qvartet superior	6 0	0110 0000
4	Caz 2 şi 3 simultan	6 6	0110 0110

La microprocesoarele Intel, instrucțiunile de corecție sunt: DAA (*Decimal Adjust for Addition*) şi DAS (*Decimal Adjust for Substraction*). Acestea fac analiza rezultatului şi corecția sa (Tab 2).

5.7 Indicatorul de întrerupere (I - Interrupt)

O întrerupere este un eveniment care perturbă temporar activitatea microprocesorului, pentru tratarea prioritară urgentă a unui subprogram.

Când un eveniment extern solicită procesorului o întrerupere, acesta examinează indicatorul de întreruperi pentru a şti dacă acestea sunt autorizate de program. Aşadar, programul este cel care în anumite perioade comută indicatorul, în scop de autorizare sau interzicere a cererilor de întrerupere, de exemplu:

- În "0" logic pentru autorizarea întreruperilor;
- În "1" logic pentru interzicerea temporară a întreruperilor.

5.8 Indicatorul de ''pas cu pas'' (T - Trap)

La construirea unui program complex, este foarte probabil ca el să conțină erori de concepție (cele de sintaxă sunt detectate automat de către compilatoare sau interpretoare). Aceste erori duc la o comportare bizară a programului sau chiar nefuncționare. Pentru a detecta erorile de acest tip, o metodă eficientă este rularea instrucțiune cu instrucțiune, cu posibilitatea analizării operațiilor ce se execută, "pas cu pas", fără limită de timp. Pentru ca procesorul să execute o instrucțiune şi apoi "stop", pentru a permite analiza efectelor, s-a introdus indicatorul ***Trap (T)*** sau *Step*, sau *Single Step*. Dacă el este activ ("1"), microprocesorul, care îl consultă permanent, se opreşte după fiecare instrucțiune executată. Dacă el este inactiv ("0"), microprocesorul rulează normal, la viteza dată de frecvența de tact (zeci, sute MHz).

5.9 Indicatorul de reluare (R - Resume)

Acest indicator este utilizat (ca şi Trap) pentru depanarea programelor. El este testat de procesor la întâlnirea unui punct de oprire.

Un *punct de oprire* (*Breakpoint*) este un marcaj temporar introdus în program, la care procesorul se opreşte şi aşteaptă o comandă de reluare a rulării. Acest mod de lucru este foarte util la verificarea funcționării normale a secvențelor de program dintre două puncte de oprire.

Indicatorul R permite luarea în considerație a punctelor de oprire (R="0") sau ignorarea acestora (R="1").

5.10 Indicatorul de direcție (D - Direction)

Când se transferă blocuri lungi de date dintr-o zonă de memorie în alta, transferul se poate face prin incrementarea adreselor sursă şi destinație sau prin decrementarea acestora. Dacă D=0, adresele vor fi modificate prin incrementare iar dacă D=1, adresele vor fi decrementate după fiecare transfer de octet.

5.11 Indicatorul de mod virtual (VM-Virtual Mode)

Intel a realizat o serie de microprocesoare începând cu 8086/8088 şi care a continuat cu 80186, 286, 386, 486, şi Pentium. Ele sunt compatibile în mod ascendent, adică unul mai puternic este capabil să execute orice program scris pentru cele precedente, însă nu şi invers.

Microprocesorul *Intel* 8086, de exemplu, poate să lucreze numai în modul numit *real*. El poate adresa maxim 1MB de memorie centrală şi nici un octet în plus. Intel 486 este un procesor de 32 de biți, dispune de o magistrală de adrese de 32 de linii şi este capabil să adreseze până la 4 GB (4096 MB) de memorie reală.

Pentru ca un 486 să poată executa un program scris pentru 8086, este necesar fie ca el să comute într-un mod *real*, fie să simuleze acest mod, rămânând în modul său normal de lucru, care este protejat de măsuri de securitate privind accesul la memorie. În acest caz, 486 va utiliza un *mod virtual de funcționare*, în care simulează pe 8086 în spațiul său de memorie protejat.

Indicatorul de mod virtual (**VM** - *Virtual Mode*) autorizează prin VM = 1 sau interzice, prin VM = 0, modul de lucru virtual.

5.12 Indicatorul de aliniere (AC - Alignment Check)

Datele şi instrucțiunile unui microprocesor pot avea lungimi diferite (în octeți), ceea ce produce unele dificultăți la transfer.

Să presupunem că un microprocesor lucrează cu cuvinte de 64 de biți; el citeşte din memorie primii 64 de biți, apoi următorii 64 de biți şi aşa mai departe. Rezultă atunci că adresele cu care lucrează sunt divizibile cu 8, deoarece 64 biți = 8 octeți (8 x 8 biți = 64). Un bloc de octeți (cu o anumită lungime, nu neapărat de 8 octeți) ce se transferă simultan se numeşte adesea *pagină*. Trecerea la blocul următor se numeşte *salt de pagină*.

Dacă instrucțiunea sau data se află la început de pagină, totul este O.K., fără complicații. Dar dacă nu se întâmplă aşa, trebuie să se calculeze adresa la care se află, față de începutul paginii, să se numere octeții ce constituie data sau instrucțiunea şi să se transfere numai ceea ce este necesar. Aceste operații complică funcționarea.

De aceea se preferă "alinierea" datelor la început de pagină, adică să înceapă sistematic la adrese divizibile prin lungimea paginii, de exemplu 64. Nu toate datele pot fi însă "aliniate".

În acest caz, se face apel la un indicator special, de aliniere, care atunci când este activ (AC = 1) arată că locația de memorie adresată nu este "aliniată".

Acest indicator, apare numai la microprocesoarele evoluate.

5.13 Indicatorul de împachetare (NT - Nested Task)

Tipic microprocesoarelor Intel, indicatorul de împachetare a task -urilor, NT, este folosit în *modul protejat* de funcționare. El este activat (NT = 1) când pachetul de programe curent (task) este inclus în cel precedent. Astfel, procesorul ştie că la terminarea execuției pachetului curent va relua execuția celui precedent.

5.14 Indicatorul de nivel de privilegiu (IOPL - Input / Output Privilege Level)

La procesoarele sale avansate, *Intel* a introdus numeroase măsuri de protecție privind accesul la resursele software. Un program de aplicație, de exemplu, nu are acces la resursele sistemului; drepturile lui sunt sever reglementate. Drepturile de acces sunt diferențiate în 4 categorii, numite *niveluri de privilegiu* sau de acces. Nivelul "0", central, este cel care permite totul (privilegii maxime) iar nivelul extern, "3", nu permite aproape nimic. Programul de aplicație are nivelul 3.

Tipic procesoarelor *Intel*, *indicatorul de privilegiu*, IOPL, este utilizat în *modul protejat*. El specifică prin cei doi biți ai săi, nivelul de privilegiu care permite execuția unei instrucțiuni de intrare-ieşire. Cei doi biți codifică 4 niveluri posibile: 00, 01, 10, 11.

5.15 Indicatorul de mod supervizor (S - Supervisor)

Tipic microprocesorului Motorola 68 000, de data aceasta, acest indicator plasează sistemul:

- În modul "utilizator", dacă este "0" logic;
- În modul "supervizor", dacă este "1" logic; este permis accesul la toate resursele sitemului.

6 Întreruperi

O *întrerupere* este în esență un eveniment extern, care intervine la un moment neprevăzut, în timp ce microprocesorul execută programele sale. Un obiect din sistem sau conectat la el, solicită atenție şi cere procesorului întreruperea activității curente pentru a fi servit cu prioritate. Procesorul dă curs cererii, dar nu înainte de a-şi lua unele precauții.

După ce încheie tratarea întreruperii, procesorul revine la programul curent pe care îl abandonase un timp şi pe care îl continuă ca şi cum nimic nu s-ar fi întâmplat. Ceea ce complică puțin lucrurile este numărul mare de întreruperi, gradele diferite de prioritate, modul de tratare, etc.

Întreruperile constituie unul din cele trei moduri de schimb de informație ce se aplică în tehnica microprocesoarelor:

- **Schimbul programat**. Pe baza programului în execuție, unitatea centrală are inițiativa de dialog cu un element periferic.

- **Întreruperile**. Un periferic are inițiativa dialogului, în momente ce nu pot fi prevăzute în programe, care depind de funcțiile sale.
- **Accesul direct la memorie** (DMA - *Direct Memory Access*). Se face la inițiativa unui dispozitiv inteligent (alt procesor) şi are ca avantaj principal viteza mare de transfer de date.

6.1 Schimbul programat

Schimbul de date este prevăzut în program; o instrucțiune va cere procesorului emiterea unui mesaj către un periferic, care este adesea un mesaj de interogare. De exemplu, interogarea tastaturii privind apăsarea unei taste, interogarea modemului privind datele recepționate sau emise.

Procedura de schimb programat este următoarea:

- Adresarea perifericului vizat prin încărcarea adresei pe magistrală;
- Procesorul aşteaptă un moment pentru stabilizarea adresei;
- Pe magistrala de comandă se emite un ordin de validare a adresei pentru toate perifericele, dar numai cel vizat se va activa;
- Procesorul plasează datele pe magistrala de date, dacă este vorba de scriere într-un port sau invers, perifericul este cel care plasează date pe magistrală. Să presupunem că este o scriere în port:
- Procesorul activează o comandă de validare a datelor (datele sunt stabile) pentru utilizator;
- Schimbul a avut loc şi este încheiat, dacă este cazul transmiterii unui singur cuvânt.

Procedura este ilustrată în fig. 1.

Schimbul programat pare simplu, dar sunt necesare câteva condiții pentru ca el să aibă loc la momentul potrivit:

1. Instrucțiunea să apară în program când trebuie şi de câte ori este necesar.

2. Instrucțiunea să apară suficient de des; de exemplu, pentru a sesiza dacă utilizatorul foloseşte tastatura, se introduce o instrucțiune de citire a tastaturii la fiecare zecime de secundă.

3. Dacă ar fi un singur periferic, lucrurile ar fi încă simple; sunt însă, de regulă, peste 10, ceea ce impune o organizare sistematică a accesului, fiecare periferic având propriile sale caracteristici.

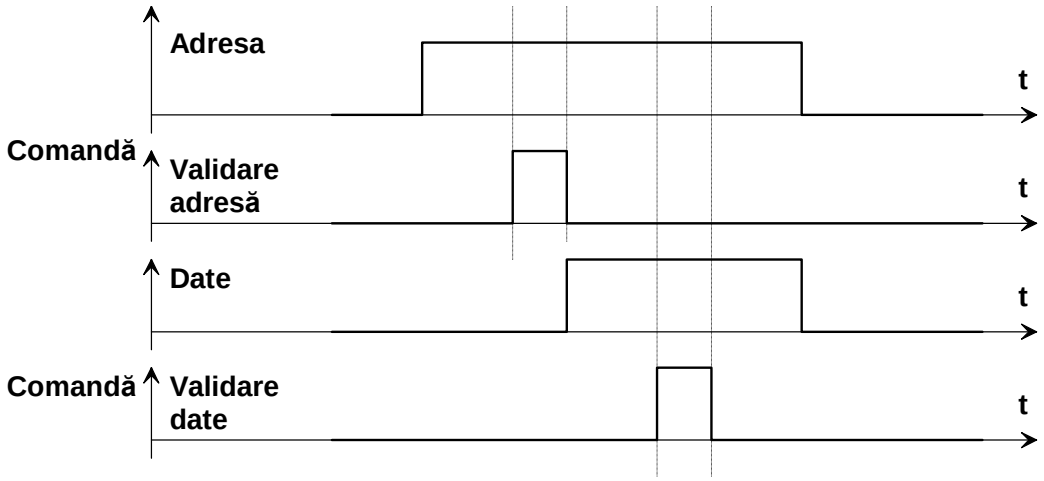


Fig. 1 Principiul schimbului programat. Procesorul plasează o adresă pe magistrală şi apoi emite un ordin de validare pe magistrala de control. Nu a fost reprezentată comanda de scriere.

Avantaje: programul conține toate apelurile necesare, nu apar surprize; conceperea şi verificarea programelor este simplă.

Dezavantaje: tratarea schimburilor aleatoare, asincrone, este mai dificilă. Interogarea periodică a perifericelor, adeseori în mod inutil, determină o scădere a vitezei de execuție prin pierdere de timp

6.2 Întreruperile

O întrerupere permite unui eveniment aleator, care apare în mod imprevizibil, să fie luat în considerație de microprocesor. Ea oferă încă alte avantaje. Să presupunem că microprocesorul trimite un şir de caractere la imprimantă, care este un periferic lent față de viteza de lucru a procesorului; acesta trebuie să aştepte un timp îndelungat până când imprimanta este disponibilă pentru un nou şir de caractere, timp în care, practic, procesorul este blocat inutil. Dacă se lucrează în întreruperi, procesorul după trimiterea şirului de caractere îşi continuă activitatea până când imprimanta printr-o nouă cerere de întrerupere va informa procesorul că este disponibilă pentru un nou şir de caractere. În acest mod de lucru, se câştigă timp de execuție enorm pentru programele curente şi creşte viteza medie de execuție.

Procedura de schimb de date în întreruperi utilizează linii speciale pentru dialogul procesor - periferic (fig. 2).

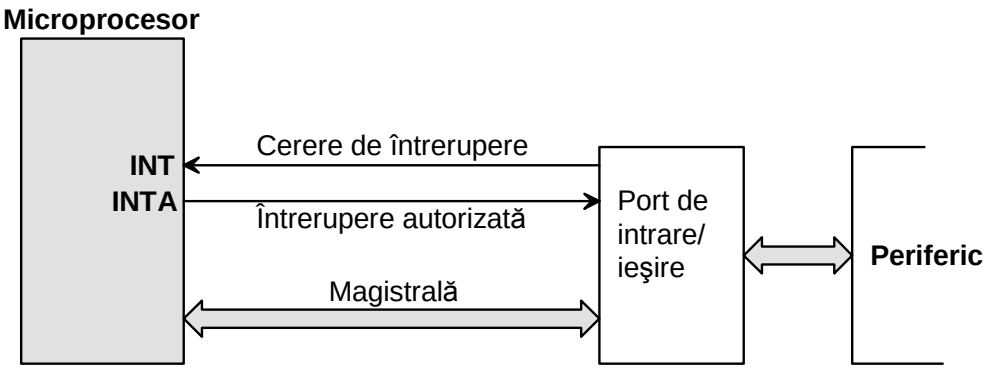


Fig. 2 Liniile necesare în întreruperi

Procedura se desfăşoară astfel:

1. Un periferic alertează circuitul său de interfață (I/O - intrare / ieşire) în vederea schimbului de informație cu microprocesorul;

2. Circuitul I/O trimite o **cerere de întrerupere** la procesor pe o linie specială, INT;

3. Procesorul termină instrucțiunea curentă şi salvează în memorie conținutul tuturor registrelor utilizate în programul în curs de execuție (indicatori de stare, acumulator, contorul de program, etc.)

Salvarea registrelor se face într-o zonă de memorie rezervată, numită **memorie stivă** (*Stack*).

4. Microprocesorul trimite către circuitul de interfață un semnal de acceptare cerere de întrerupere (INTA - *Interrupt Acknowledge*) printr-o linie specială;

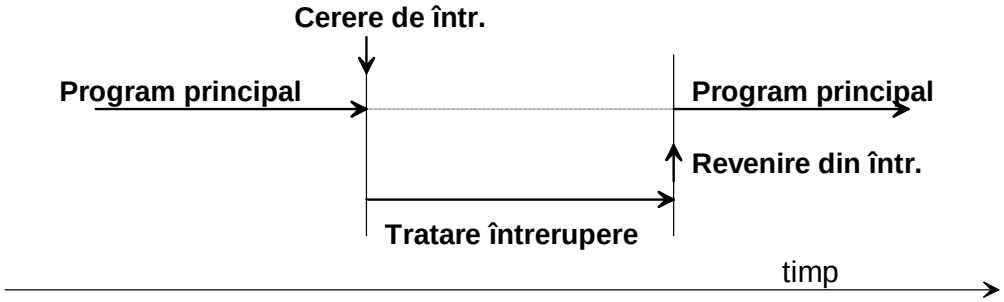


Fig.3 Principiul tratării unei întreruperi

5. Microprocesorul apelează un subprogram specific de tratare a cererii de întrerupere, care depinde de sursa de întrerupere: un anumit program pentru tastatură, un altul pentru modem, etc.

6. După execuția programului specific, procesorul poate relua execuția programului întrerupt; pentru aceasta, el transferă din memoria stivă toate informațiile în registrele corespunzătoare şi continuă programul exact din punctul în care l-a abandonat.

Programul de tratare a întreruperii, numit adesea "subrutină de tratare" se termină în mod obligatoriu cu o instrucțiune de revenire din subrutină (*Return*), care poate avea mai multe forme.

Instrucțiunea de revenire determină comutarea execuției la programul principal, cel care fusese întrerupt.

6.2.1. Memoria stivă

Ansablul registrelor interne ale microprocesorului determină complet starea programului la un moment dat. Dacă ele nu ar fi salvate în timpul tratării unei întreruperi, conținutul lor s-ar pierde, deoarece aceleaşi registre sunt utilizate de subrutina de tratare. De aceea, în mod imperativ, ele trebuie salvate înainte de începerea execuției subrutinei de tratare.

Se utilizează o zonă de memorie RAM, numită stivă (*Stack*), care funcționează ca o listă de tip LIFO (*Last In First Out*). Ea se comportă ca o stivă de farfurii: ultima farfurie aşezată în stivă este totdeauna prima care este luată când este nevoie de una.

În practică, programele se plasează la o extremitate a memoriei iar stiva sau stivele, la cealaltă extremitate. Programele se desfăşoară în ordine crescătoare a adreselor de memorie, iar stiva este utilizată în ordine descrescătoare a adreselor (fig. 4).

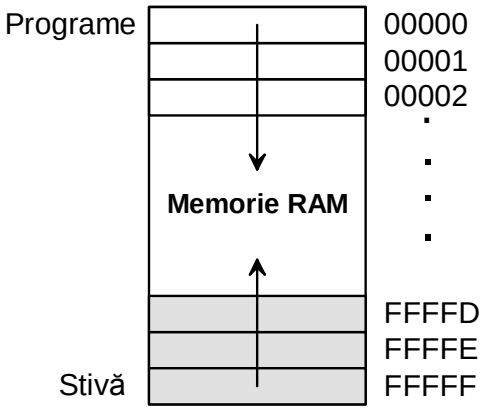


Fig.4 Programele şi stiva sunt plasate în RAM la capete diferite

Există riscul ca datele din stivă să se intersecteze cu codurile din programe, ceea ce duce la blocarea sistemului. La procesoarele evoluate acest risc este foarte redus datorită algoritmilor de adresare a memoriei.

Indicatorul de stivă. Pentru utilizarea stivei sunt necesare, în general, două adrese: adresa de început (FFFFF, în figura de mai sus) şi adresa vârfului stivei, pentru a şti care este prima locație liberă, deci nivelul de umplere al stivei. Indicatorul de stivă (*Stack Pointer*) este un registru care conține de regulă adresa primei locații libere şi se comportă

ca un flotor care arată pe o scală gradată nivelul unui lichid într-un recipient.

În funcție de convenții şi de fabricanții de microprocesoare, registrul indicator de stivă conține fie adresa ultimei locații de memorie ocupate, fie adresa primei locații libere. Vom considera în continuare că registrul indicator de stivă conține adresa primei locații libere din stivă.

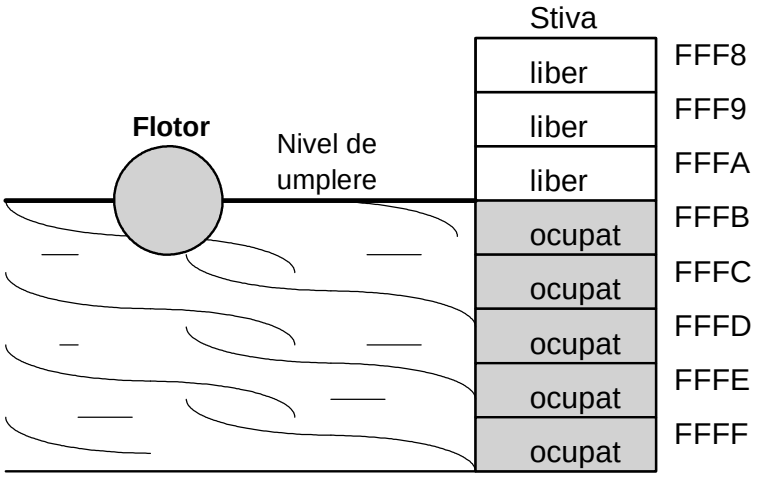


Fig. 5 Indicatorul de stivă se comportă ca un flotor şi arată nivelul de umplere al stivei

6.2.2. Tehnica de utilizare a stivei

La începerea unui program, se inițializează stiva prin încărcarea indicatorului de stivă (SP - *Stack Pointer*) cu adresa de bază a stivei (de exemplu cu FFFF), adică prima locație liberă din stivă este chiar locația la care începe stiva. Să presupunem că pentru tratarea unei întreruperi trebuie salvat în stivă contorul de program (*PC - Program Counter*), de 16 biți. Stiva este formată din locații de 8 biți, deci salvarea registrului PC în stivă se face octet cu octet.

Procedura de salvare în stivă va fi:

- Procesorul expediază primul octet, cel mai semnificativ, în locația dată de indicatorul de stivă, SP = FFFF;
- Procesorul decrementează indicatorul de stivă SP = FFFF-1=FFFE;
- Se expediază al doilea octet, cel mai puțin semnificativ, la adresa dată de SP = FFFE;
- Se decrementează din nou indicatorul SP = FFFE-1=FFFD, adresa primei locații libere.

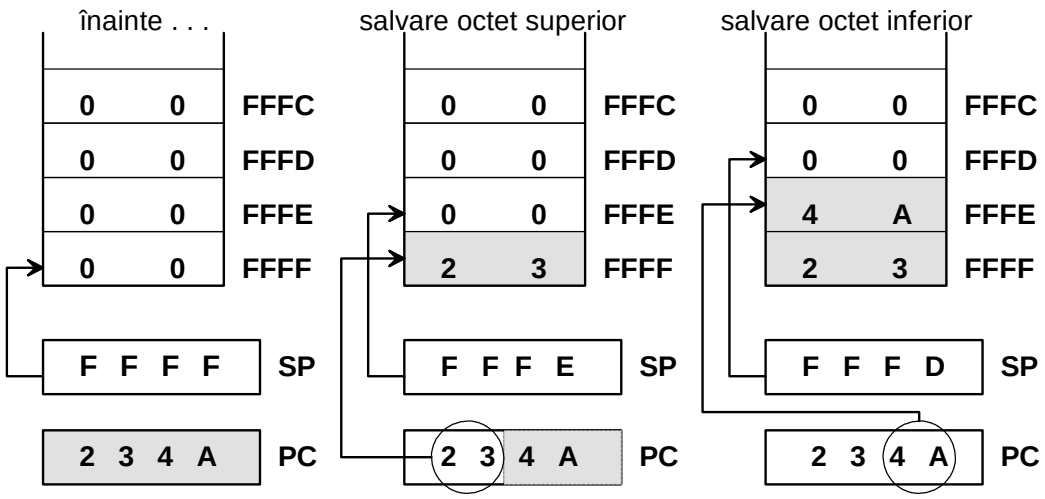


Fig. 6 Etapele salvării unui registru de 16 biți în stivă

Operațiile sunt ilustrate în fig.6. Salvarea în stivă se face prin copiere nedistructivă, adică octetul se transferă în stivă, dar registrul sursă, PC nu se modifică.

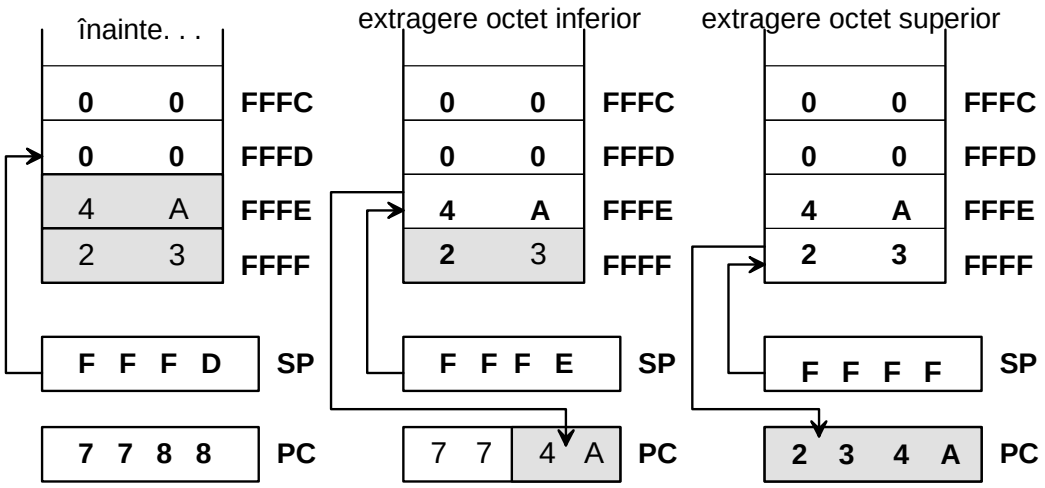


Fig.7 Etapele refacerii unui registru de 16 biți (PC) din stivă

După savare în stivă, microprocesorul execută subrutina de tratare a întreruperii. Aceasta se termină obligatoriu cu o instrucțiune de revenire în programul principal, notată RET (*Return*) sau IRET (*Interrupt Return*) sau EOI (*End Of Interrupt*).

Când întâlneşte această instrucțiune, microprocesorul trebuie să recupereze adresa (fig.7) pe care a salvat-o în stivă (în exemplu, 234A H) şi s-o plaseze în registrul PC; ordinea operațiile va fi:

- Indicatorul SP conține adresa FFFD a primei locații libere;
- Microprocesorul începe prin incrementarea indicatorului de stivă pentru a obține adresa primei locații ocupate: SP=FFFD+1=FFFE;

- El citeşte octetul şi îl plasează în PC (octet inferior);
- Este din nou incrementat indicatorul SP=FFFE+1=FFFF;
- Octetul citit este plasat în PC (octet superior);
- Programul poate fi reluat de la adresa la care a fost întrerupt, care acum se află din nou în PC.

Citirea memoriei nu este distructivă; aceleaşi informaţii rămân în stivă, dar ele nu mai sunt de interes, deoarece indicatorul nu le mai selectează. Pentru el, stiva este vidă.

În realitate, acceptarea unei întreruperi impune salvarea tuturor registrelor interne folosite în program, din care cele mai importante sunt: acumulatorul, registrul indicatorilor de condiţii, registrul de adrese PC şi registrele index. Salvarea în stivă se face exact ca în exemplul de mai sus, pentru fiecare registru, respectând o *anumită ordine* şi în mod riguros *în ordine inversă,* se realizează refacerea registrelor din stivă.

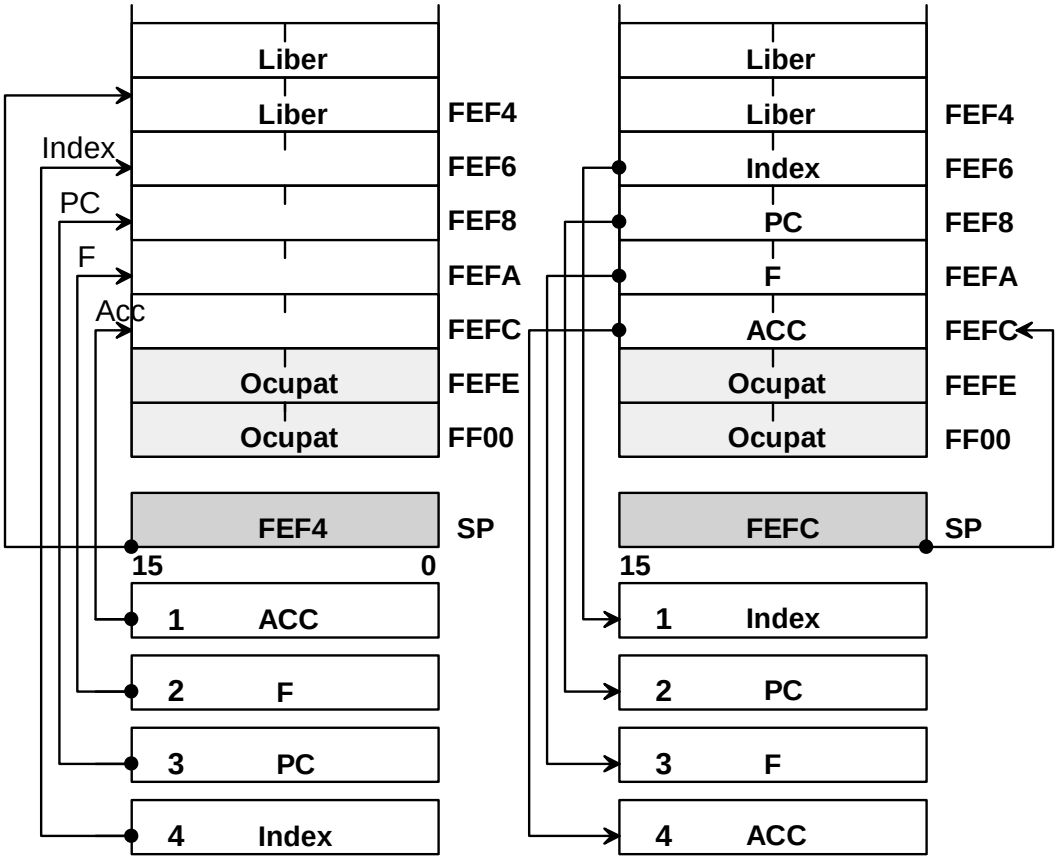


Fig. 8 Salvarea şi refacerea unui set de 4 registre de 16 biţi

La unele procesoare ordinea de introducere în stivă nu este impusă dar cea de refacere trebuie să corespundă (în sens invers) cu cea de la introducere.

La alte procesoare, salvarea în stivă se face cu o singură instrucţiune, care introduce toate registrele interne în ordinea stabilită de

proiectantul procesorului iar refacerea din stivă se realizează de asemenea cu o singură instrucțiune, care în mod automat scoate registrele în ordinea inversă introducerii lor.

În fig.8, este prezentată o stivă cu locații de 16 biți, cu octetul inferior în dreapta. Se salvează în stivă registre de 16 biți: Acc., F, PC şi un registru Index, în această ordine. Refacerea din stivă se face în ordine inversă: Index, PC, F, Acc. (ultimul intrat este primul ieşit).

Instrucțiunea de salvare în stivă este numită adesea PUSH iar cea de recuperare din stivă, POP sau PULL.

6.2.3. Subprogramele de tratare întreruperi

Se poate imagina că, atunci când un periferic solicită o întrerupere microprocesorului, el trimite simultan şi adresa subrutinei de tratare a cererii de întrerupere. Aşa ar fi totul mai simplu. În practică însă, toate "programele de gestionare a perifericelor" sunt încărcate de fişierele sistem, CONFIG.SYS şi AUTOEXEC.BAT. Software -ul de sistem este deci cel care "ştie" la ce adrese încep aceste programe.

Într-un PC există o tabelă care conține adresele de început ale subrutinelor de tratare, numită **tabela vectorilor de întrerupere**. Ea este sistematic şi obligatoriu memorată începând cu adresa 00000 H, a memoriei principale şi ocupă 1kB (1024 octeți), deci zona de memorie 00000 - 003FF H. Pentru fiecare adresă de întrerupere se rezervă 4 octeți, deci în total capacitatea este de 256 de întreruperi.

Un PC clasic dispune de 16 linii de întreruperi declanşate de periferice şi numerotate 0, 1, 2, . . . ,15, aşadar primele 16 adrese de 4 octeți le sunt rezervate.

Iată care sunt (în mod detaliat) etapele ce se succed la apariția unei cereri de întrerupere:

- Un periferic solicită întrerupere;
- Circuitul de interfață dintre periferic şi sistem transmite cererea la un circuit specializat în gestionarea întreruperilor (controler);
- Controlerul de întreruperi (programabil) transmite cererea la microprocesor, însoțind-o de numărul său caracteristic;
- **Microprocesorul** termină instrucțiunea aflată în execuție şi incrementează contorul de program (PC), acesta fiind astfel pregătit pentru extragerea instrucțiunii următoare, care nu va fi însă extrasă imediat;
- Salvează registrele de uz general şi registrul de stare, în stivă;
- Trimite către periferic un semnal special de "acceptare a cererii de întrerupere";

- Multiplică prin 4 numărul întreruperii pentru a calcula o adresă de memorie (se aplică numai la PC - uri, pentru că, în general, sistemele de calcul au propriile reguli de formare a adreselor);
- Începând cu adresa calculată, citeşte din memorie 4 octeți care vor fi utilizați pentru reperarea subrutinei de tratare a întreruperii, în memorie;
- Execută programul de tratare a întreruperii, începând de la adresa determinată cu cei 4 octeți;
- Subrutina de tratare se termină cu o instrucțiune de tip *Return*, adică revenire în programul care a fost întrerupt temporar;
- Procesorul recuperează din stivă conținutul registrelor generale şi de stare şi continuă execuția programului care a fost întrerupt.

În acest proces, este important modul de adresare a subrutinei de tratare a întreruperii (fig. 9). Acesta este un mod indirect, cu două etape de adresare: în prima etapă se utilizează numărul întreruperii pentru citirea din memorie a unei adrese pe 4 octeți iar în etapa a doua se adresează efectiv subrutina de tratare.

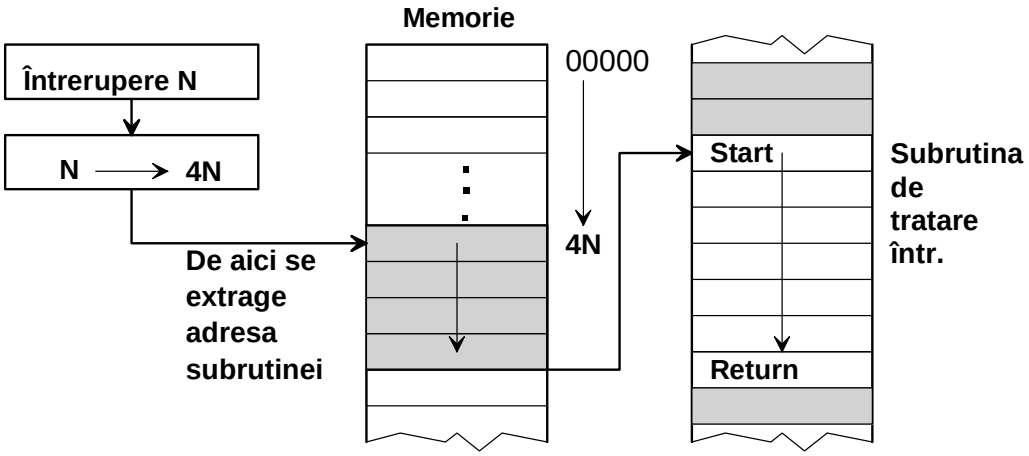


Fig. 9 Modul de adresare a subrutinei de tratare întrerupere

6.2.4. Gestionarea întreruperilor

Este posibil ca două sau mai multe periferice să solicite întreruperi simultan. Pentru a rezolva asemenea situații, proiectanții de sisteme numerice au acordat fiecărei întreruperi un anumit grad de prioritate, care permite unei întreruperi prioritare să fie luată în considerație înaintea alteia.

Să presupunem că o primă cerere de întrerupere a fost acceptată şi se află în curs de tratare; în acest timp, se manifestă o a doua cerere de întrerupere . Sunt posibile două cazuri:

1. A doua are grad redus de prioritate: ea va aştepta ca prima să fie terminată, pentru a fi luată în considerație.

2. A doua este prioritară față de prima: în acest caz ea întrerupe tratarea primei întreruperi, aşa cum prima a întrerupt programul pricipal. Microprocesorul salvează în stivă registrele al căror conținut depinde de tratarea primei întreruperi, tratează a doua întrerupere şi revine la tratarea primei întreruperi după ce recuperează din stivă conținutul registrelor.

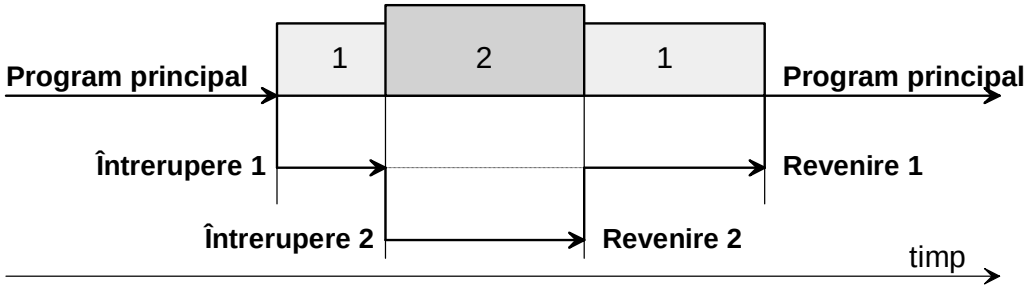


Fig. 10 Principiul tratării întreruperii prioritare (2) în timpul servirii întreruperii 1

Numărul de întreruperi ce pot fi incluse unele în altele (imbricate) nu este limitat la două. De fiecare dată, conținutul unui nou set de registre este introdus în stivă. Unul din inconveniente este desigur ocuparea stivei, care nu trebuie să depăşească zona alocată. Capacitatea stivei trebuie determinată pentru cazurile extreme.

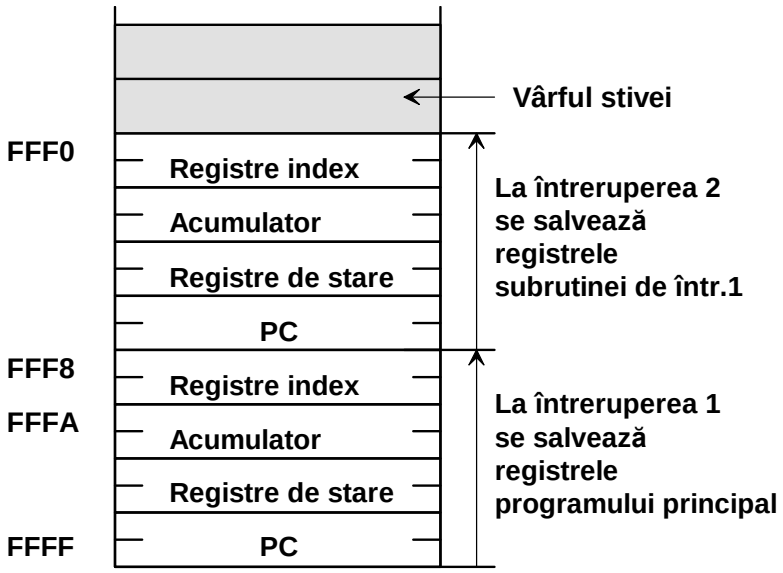


Fig.11 Aspectul stivei în situația apariției întreruperii 2, prioritară față de întreruperea 1.

Se poate înțelege (din fig. 11) de ce este avantajos ca stiva să funcționeze pe principiul LIFO (*Last In First Out*). Nu pot să apară confuzii şi nici amestec de date, deoarece datele salvate în stivă sunt necesare exact în ordinea în care pot fi extrase.

Practic, gestionarea priorităților este asigurată prin două metode:

- Prin cascadarea intrărilor de întrerupere în ordinea priorității;
- Prin utilizarea unui circuit specializat (controler de întreruperi).

Prima metodă are avantajul simplității iar cea de-a doua este mai eficientă.

În fig.12 este prezentat principiul cascadării. Fiecare circuit I/O are o intrare de validare întrerupere şi o ieşire către circuitul următor; dacă cererea de întrerupere proprie (prioritară) nu este activă, el transferă validarea către circuitul următor, şi aşa mai departe.

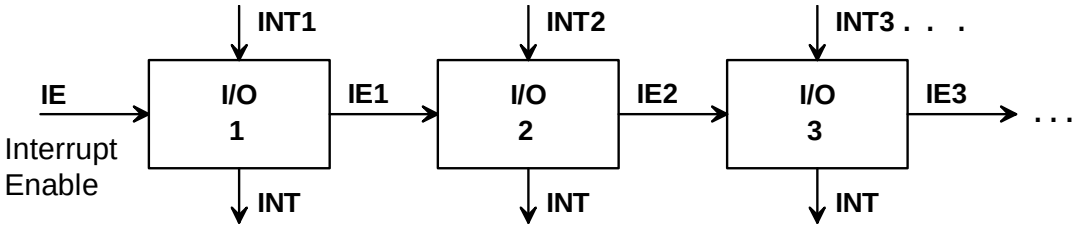


Fig. 12 Conectarea în cascadă a circuitelor I/O

Ordinea de conectare în cascadă a circuitelor I/O determină gradele de prioritate: întreruperea conectată la primul circuit din şir este cu prioritate maximă. Numărul de circuite din şir rămâne limitat la 7 - 8 din cauza acumulării timpilor de întârziere la transmisie. Este unul din dezavantajele majore.

În fig.13 este prezentat principiul gestionării întreruperilor prin controler specializat, PIC (*Programmable Interrupt Controler*).
În schemă au fost reprezentate 4 intrări, pentru simplificare, căci adesea un controler are 8 intrări. Dacă apare o cerere de întrerupere, INT="1", ieşirea circuitului NAND trece în "1" şi se transmite către procesor; după validarea cererii (IE - *Interrupt Enable*) sunt deblocate cele 4 circuite AND, dintre care unul singur va avea la ieşire "0" şi anume cel care corespunde cererii active. Codul de ieşire poate fi unul din: 0111, 1011, 1101, 1110, în funcție de linia de întrerupere activă. Acest cod va fi transmis la microprocesor pe magistrala de date. Intrarea INT_0 va fi cu prioritate maximă iar INT_3 cu prioritate minimă.

Circuitul cel mai utilizat este PIC Intel 8259. Acesta atribuie fiecărei cereri, prioritatea standard sau cea stabilită prin programare şi poate deservi 8 cereri.

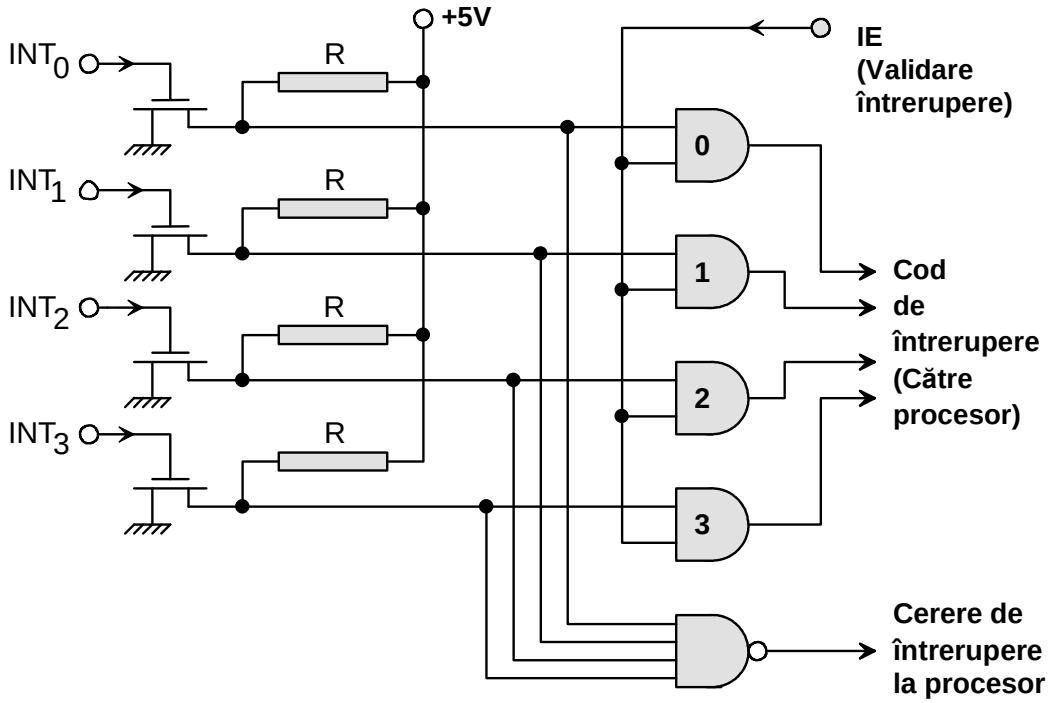


Fig. 13 Principiul controlerului de întreruperi cu 4 intrări

6.2.5. Mascarea întreruperilor

Când în programul principal se execută o secvență ce nu permite întrerupere, acestea pot fi interzise temporar, pentru a nu se produce perturbații în programul principal.

Interdicția temporară a întreruperilor este cunoscută sub numele de "mascare a întreruperilor".

Metoda cea mai utilizată de proiectanții de microprocesoare este poziționarea unui indicator I (*Interrupt*), în cuvântul de stare. Dacă acesta are valoarea "0", de exemplu, întreruperile de tip INT sunt mascate. Mascarea şi demascarea întreruperilor se face prin instrucțiuni inserate în program, de tip DI (*Disable Interrupt*), respectiv EI (*Enable Interrupt*). Etapele de mascării temporare sunt:

- Instrucțiunea DI în program (mascare întreruperi);
- Secvență de instrucțiuni ce nu poate fi întreruptă;
- Instrucțiunea EI în program (validare întreruperi);

Când se utilizează controler de întreruperi, PIC, acesta permite în plus, prin programare, mascarea anumitor întreruperi sau schimbarea temporară a priorităților.

6.2.6. Tipuri de întreruperi

1. Întreruperi de tip *hardware*, sau întreruperile materiale, sunt cele declanşate de un periferic.

2. Întreruperi de tip *software*, introduse în program sub forma unor instrucțiuni; programatorul poate defini propriile sale întreruperi.

Într-un calculator PC, tabela vectorilor de întrerupere oferă 256 de adrese pentru tot atâtea cereri distincte. Dintre acestea, sunt efectiv utilizate ca întreruperi hard, doar 16; celelalte sunt disponibile ca întreruperi de tip soft.

Numărul întreruperii	Originea întreruperii
0	Impulsurile de tact
1	Tastatura
2	Cascadarea unui PIC
3	Porturi seriale COM2 şi COM4
4	Porturi seriale COM1 şi COM3
5	Port paralel LPT2
6	Unitate Floppy Disk
7	Port paralel LPT1 (imprimantă)
8	Ceas-calendar în timp real
9	Redirecționare IRQ2 - ecran VGA
10	Rezervat sau liber
11	Rezervat sau liber
12	Rezervat sau liber
13	Coprocesor aritmetic
14	Hard Disk
15	Rezervat sau liber

Într-un microcalculator PC, sistemul de operare face apel la întreruperile soft pentru activarea unor funcții de sistem. De exemplu, INT 21H dă acces la numeroase funcții.

La rândul lor, întreruperile de tip hard sunt de mai multe categorii:

- Întreruperile INT, sunt cele prezentate până acum;
- Întreruperile de înaltă prioritate, NMI (*Non Mascable Interrupt*), care sunt deci nemascabile prin program; acestea se aplică la o intrare specială a procesorului şi reprezintă intervenția operatorului sau un avertisment de urgență;
- O întrerupere suverană, RESET, care produce în mod inevitabil reinițializarea procesorului şi implicit a sistemului de calcul.

7 Acces direct la memorie

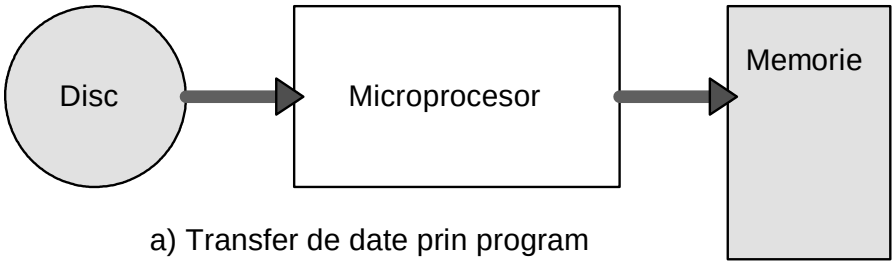
Funcția de bază a unui microprocesor este aceea de a executa instrucțiunile programului, utilizând când este cazul memoria externă, porturile, magistralele, deci toate resursele sistemului. Anumite operații de rutină, ca de exemplu transferul unor fişiere masive între memoria centrală şi disc, necesită intervale importante de timp, ceea ce duce la scăderea vitezei medii de execuție a programelor.

Se realizează o eficiență ridicată a procesului de transfer dacă acesta se face fără implicarea directă a microprocesorului. Soluția este accesul direct la memorie, ***DMA (Direct Memory Access****), realizat de un dispozitiv specializat sub controlul unității centrale.*

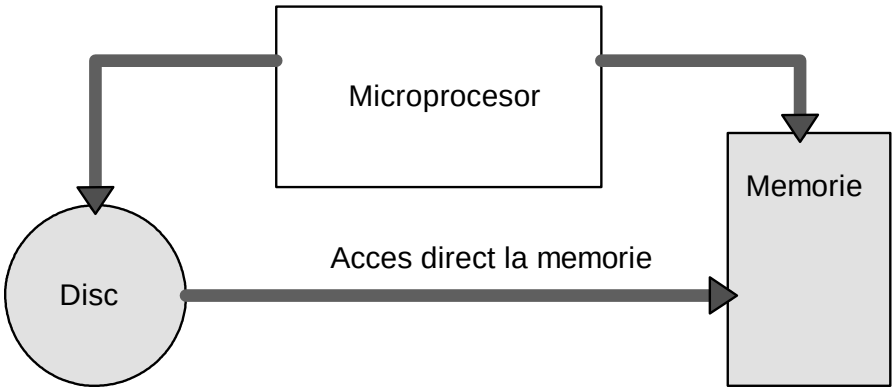
7.1 Principiul tehnicii DMA

Transferul de date prin tehnica DMA este inițiat de microprocesor dar după lansarea în execuție a procesului, acesta nu se mai implică în mod direct.

Microprocesorul se deconectează de magistralele sistemului (de date şi de adrese), pe durata transferului, acestea fiind utilizate de un circuit specializat pentru transfer.



a) Transfer de date prin program



b) Transfer de date prin tehnica DMA

Fig.1 Principiul transferului DMA (b), comparativ cu transferul prin program (a).

Implementarea tehnicii DMA necesită introducerea a două linii suplimentare la microprocesor, una pentru cerere DMA şi alta pentru autorizare DMA. De asemenea, trebuie introdus în sistem un circuit specializat care să coordoneze transferul şi să susţină dialogul cu microprocesorul (controler DMA).

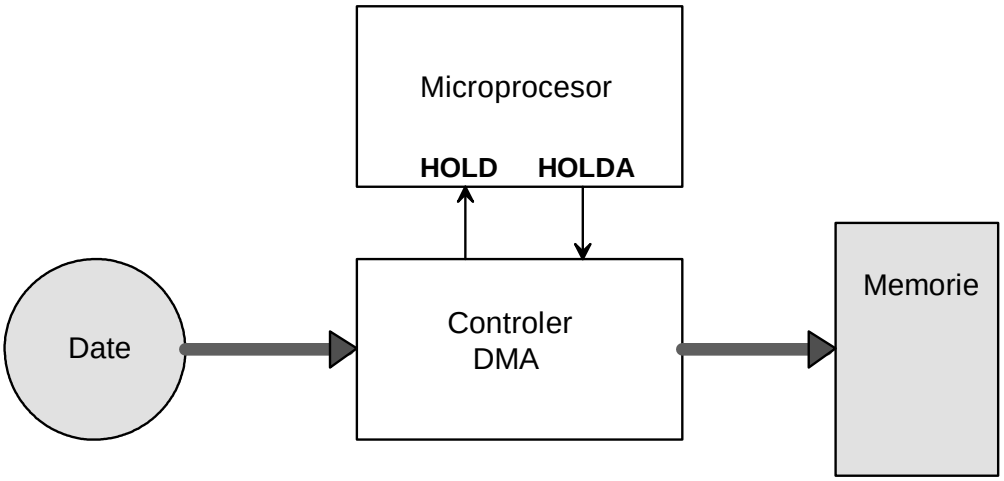


Fig. 2 Conectarea unui controler DMA în sistem

HOLD (suspendare, deconectare). Cerere DMA către microprocesor pentru cedarea magistralelor.

HOLDA - Hold Acknowledge. Răspunsul microprocesorului prin care acceptă cererea DMA; semnalul este generat după ce magistralele au fost comutate în starea de înaltă impedanţă. Procesorul rămâne deconectat de magistrale cât timp semnalul HOLD este activ şi execută numai operaţii interne care nu necesită accesul la memorie şi porturi.

Tehnica DMA dispune de trei strategii de operare care, în ordinea crescătoare a complexităţii, sunt:

- Oprirea procesorului;
- Deturnarea semnalului de tact;
- Multiplexare în timp.

7.2 DMA prin oprirea procesorului

Este metoda cea mai simplă dar nu şi cea mai eficientă. Ea necesită existenţa unui circuit specializat, numit **controler DMA.** Acesta preia controlul asupra magistralelor şi guvernează schimbul de date. Controlerul DMA este un circuit programabil. Modelul de referinţă este Intel 8237.

Pentru transferul unui bloc de date, controlerul trebuie să dispună de două adrese (adresa sursă şi adresa destinaţie) şi de două căi de transmisie (*canal 0* pentru adresa sursă şi *canal 1* pentru cea destinaţie).

Funcționarea este în principiu următoarea:
- adresa sursă este folosită pentru citirea unui octet pe canal_0;
- octetul este stocat temporar într-un registru intern;
- octetul este expediat la adresa de destinație pe canal_1.

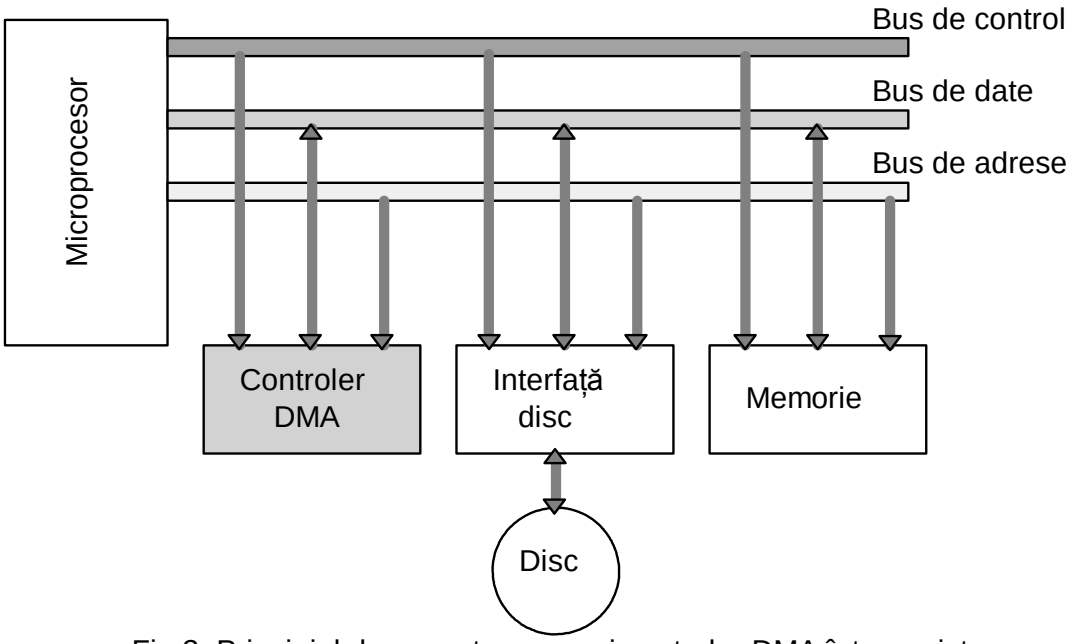


Fig.3 Principiul de conectare a unui controler DMA într-un sistem

7.2.1. Organizarea schimbului

Programul încarcă într-un registru contor numărul total de octeți ce se transferă prin tehnica DMA şi în registrele de adresă cele două adrese inițiale, sursă şi destinație. Principiul procedurii de transfer este dat de următoarele operații:

- Controlerul DMA recepționează de la un periferic o cerere DMA;
- El solicită procesorului cedarea magistralelor;
- Procesorul termină instrucțiunea curentă, cedează magistralele (trece toate ieşirile în starea de înaltă impedanță) şi generează un semnal de acceptare DMA către controler;
- Controlerul DMA inițiază transferul octet cu octet; pentru fiecare transfer se execută următoarele operații:
  - plasarea adresei sursă pe magistrala de adrese;
  - transferul octetului într-un registru temporar;
  - plasarea adresei destinație pe magistrala de adrese;
  - transferul octetului la destinație;
  - decrementarea (sau incrementarea) adreselor;
  - decrementarea numărului de octeți de transferat;
- Când contorul de octeți ajunge la "0", transferul se încheie;

- Controlerul trece în stare inactivă cererea DMA la microprocesor;
- Acesta anulează autorizarea pentru DMA către controler şi preia controlul asupra magistralelor;
- Microprocesorul reia execuția programului curent.

7.2.2. Avantaje şi dezavantaje

Procesorul se deconectează şi apoi se reconectează la magistrale fără a salva în stivă registrele interne, ceea ce constituie o importantă economie de timp în comparație cu transferul prin întrerupere.

În concluzie, transferul prin tehnică DMA este mai rapid decât prin prin întrerupere.

Pe de altă parte, transferul prin DMA suspendă temporar activitatea procesorului, ceea ce stopează execuția programului principal pe durata transferului. Dacă transferurile DMA sunt frecvente, programul principal nu poate fi executat sau viteza de execuție scade foarte mult; dacă în plus, apar şi cereri de întrerupere, microprocesorul este paralizat.

Pentru activitatea DMA, viteza de transfer este maximă. Nimic nu produce întârzierea sau întreruperea temporară a transferului din momentul în care microprocesorul cedează magistralele.

Condiție. Pentru realizarea transferului de date prin DMA cu oprirea temporară a procesorului este necesar ca toate registrele interne ale procesorului să fie de tip RAM static (realizate cu circuite basculante bistabile); în cazul în care registrele interne sunt de tip RAM dinamic, nu este posibilă aplicarea tehnicii DMA cu oprirea procesorului, deoarece există riscul pierderii totale a informației din registre.

7.3 DMA prin deturnarea semnalului de tact

Metoda elimină unele inconveniente ale primei metode, în principal obligația ca un controler DMA să aştepte terminarea instrucțiunii curente de către microprocesor pentru a primi controlul asupra magistralelor. Un alt avantaj este acela că nu stopează total microprocesorul; acesta poate dispune de registrele sale interne chiar dacă sunt de tip RAM dinamic. Aşa se explică, de altfel, succesul metodei, deoarece multe tipuri de microprocesoare din primele generații erau dotate cu registre interne de tip RAM dinamic care ocupă o suprafață mai mică pe pastila de siliciu.

Principiul metodei constă în deturnarea unui impuls de tact de la intrarea de ceas a microprocesorului şi utilizarea lui pentru DMA.

În mod normal, toate impulsurile de tact se aplică atât sistemului cât şi microprocesorului, care funcționează cu viteza ce rezultă din frecvența de tact a generatorului. Circuite specializate deturnează (fură!) un impuls

de tact (unul singur) care nu se mai aplică la intrarea de tact a procesorului. Un transfer DMA are loc pe durata acestui unic impuls.

Impulsurile următoare se aplică normal la procesor ca şi cum nimic nu s-ar fi întâmplat. Figura 4 arată tocmai această dispariție a impulsului de tact şi validarea DMA pentru o perioadă de tact.

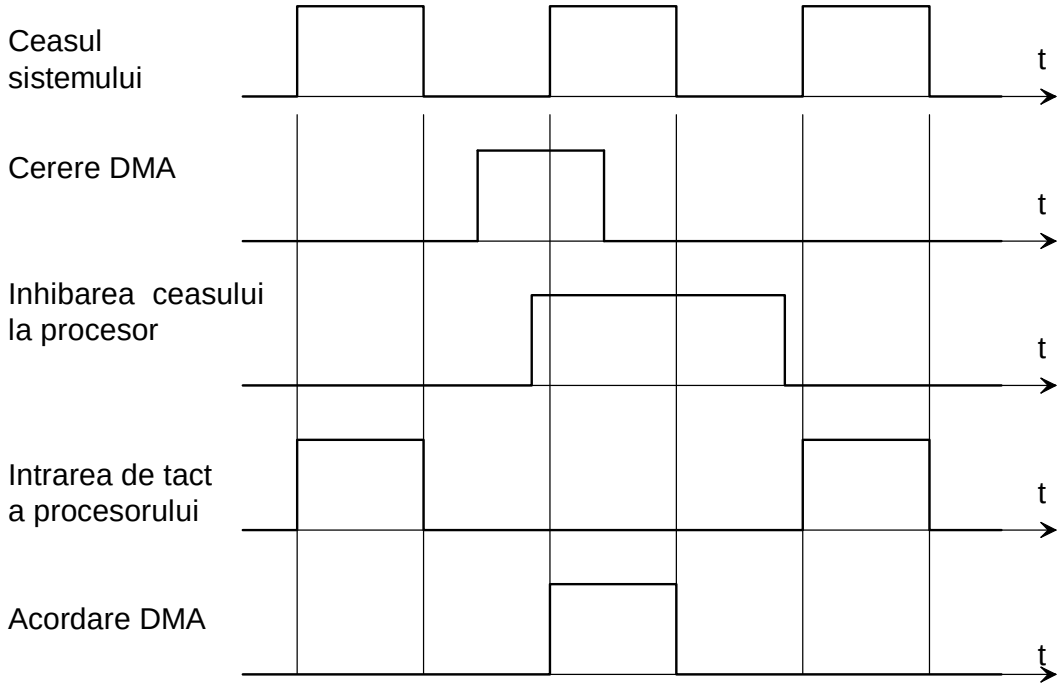


Fig.4 DMA prin deturnarea unui impuls de tact de la procesor

Deturnarea unui impuls se poate repeta cu o anumită cadență. În orice caz, microprocesorul nu este oprit din activitate ci numai încetinit (funcționează la "relanti"). Pentru a se realiza transfer DMA nu este necesar ca microprocesorul să termine instrucțiunea curentă şi să treacă magistralele în starea de înaltă impedanță; lui nu i se cere absolut nimic!.

Concluzie:

- Procesorul nu este oprit complet ci funcționează la "relanti";
- Nu este necesar să se încheie instrucțiunea curentă;
- Cererea DMA este servită imediat, în perioada de tact următoare;
- Transferul DMA este mai lent, nu se poate efectua "în salvă";
- Procesul necesită logică suplimentară pentru deturnarea impulsului de tact.

7.4 DMA prin multiplexarea magistralelor

Este metoda cea mai complexă. Se utilizează intervalele de timp în care magistralele sistemului sunt libere. Controlerul DMA detectează disponibilitatea magistralelor, când microprocesorul execută operații

interne (aritmetice, logice), preia controlul magistralelor şi execută rapid un transfer DMA.

Se produce o multiplexare în timp a ocupării magistralelor de către microprocesor şi procesul DMA; de aici rezultă numele metodei.

Principiul multiplexării magistralelor impune existența a două seturi de magistrale: un set pentru procesor, unul pentru deservirea DMA şi o logică de comutare complexă.

Cele două seturi de magistrale sunt separate de circuite tampon (*buffers*) cu ieşiri cu 3 stări (fig.5).

Se realizează cel mai bun compromis privind viteza globală de lucru între microprocesor şi DMA.

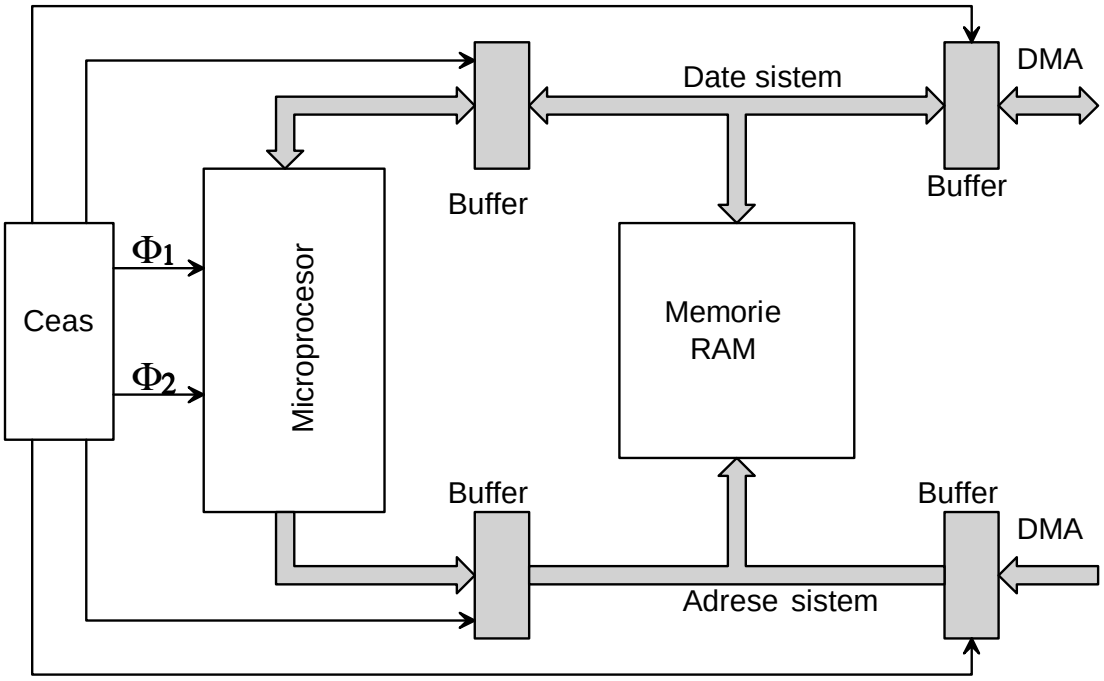


Fig.5 DMA prin multiplexarea magistralelor

Procesorul lucrează normal, controlerul DMA utlizează magistralele când sunt libere. Structura sistemului se complică prin introducerea circuitelor tampon de magistrală care produc comutarea accesului între procesor şi controlerul DMA. În cazul microprocesoarelor avansate, care utilizează foarte intens magistralele, metoda devine ineficientă pentru DMA.