

1 CONSIDERAȚII INTRODUCTIVE

1.1. Notiuni fundamentale

Calculatorul este o **mașină** destinată procesării de date, înzestrată, în acest sens, cu capabilități de a **accepta, reține, înțelege și executa ordine** exprimate de utilizator printr-o succesiune de **instrucții** conformă scopului urmărit și care constituie ceea ce se cheamă **program**.

Prin instrucții se specifică operațiile solicitate și, când este cazul, se desemnează, într-un anumit mod dintre mai multe posibile, datele de tratat și destinația rezultatelor.

Instrucțiile și datele vizate de ele sunt **construcții literal-cifrice** sugestive, de tip lingvistic, deci cu o anumită sintaxă și o anumită semantică.

Pentru ca instrucțiile și datele să poată fi acceptate, reținute, înțelese și tratate, forma lor **literal-cifrică** se transpune într-o formă **fizică**. Sub această formă, instrucțiile și datele se prezintă ca seturi de tensiuni electrice, constituite potrivit unor convenții de codificare.

În unul dintre standardele consacrate, s-a instituit convenția ca aceste tensiuni să poată lua fie o valoare în intervalul 0-0.4 V, fie o valoare în intervalul 2.4-5 V, în timp ce orice alte valori sunt interzise.

O astfel de tensiune, cu două valori posibile, se **asociază** cu ceea ce se cheamă **bit**.

Un **bit** este informația elementară, ireductibilă, caracterizată prin aceea că surprinde **una din două posibilități** care sunt, una în raport cu cealaltă, **contradictorii și complementare**.

Corespunzător celor două posibilități, bitul ia valorile logice 0 sau 1, după caz.

O informație oarecare, ce surprinde una din m posibilități se poate reprezenta printr-un set de biți de cardinal cel puțin egal cu $\log_2 m$.

Un set de n biți se cheamă cod pe n biți.

Codurile au, în contexte diferite, semnificații diferite. De exemplu, 0100 0001 reprezintă, uneori, litera “A”, alteori, numărul 65 în baza 10, iar alteori, cu totul altceva.

Un set de 8 biți se numește **byte** sau **octet**.

De obicei, codurile au lungimi care sunt puteri ale lui 2. Sunt uzuale codurile pe 8, 16, 32, 64, 128, 256 biți.

Multiplii bitului sunt:

$$\begin{aligned}1 \text{ KiloBit} &= 2^{10} \text{ Bits} \\1 \text{ MegaBit} &= 2^{20} \text{ Bits} \\1 \text{ GigaBit} &= 2^{30} \text{ Bits} \\1 \text{ TeraBit} &= 2^{40} \text{ Bits}\end{aligned}$$

$$\begin{aligned}1 \text{ PetaBit} &= 2^{50} \text{ Bits} \\1 \text{ ExaBit} &= 2^{60} \text{ Bits} \\1 \text{ ZettaBit} &= 2^{70} \text{ Bits} \\1 \text{ YottaBit} &= 2^{80} \text{ Bits}\end{aligned}$$

Multiplii *byte*-ului sunt:

$$\begin{aligned}1 \text{ KiloByte} &= 2^{10} \text{ Bytes} \\1 \text{ MegaByte} &= 2^{20} \text{ Bytes} \\1 \text{ GigaByte} &= 2^{30} \text{ Bytes} \\1 \text{ TeraByte} &= 2^{40} \text{ Bytes}\end{aligned}$$

$$\begin{aligned}1 \text{ PetaByte} &= 2^{50} \text{ Bytes} \\1 \text{ ExaByte} &= 2^{60} \text{ Bytes} \\1 \text{ ZettaByte} &= 2^{70} \text{ Bytes} \\1 \text{ YottaByte} &= 2^{80} \text{ Bytes}\end{aligned}$$

Reprezentarea fizică a bitului printr-o tensiune ce poate lua doar două valori se bazează pe fenomenele de comutație -saturare / blocare- specifice tranzistoarelor de toate tipurile. Exploatând aceste fenomene, s-au conceput și realizat circuite în comutație -numite și **circuite digitale** sau **circuite numerice** sau **circuite logice**-, ajunse actualmente la o diversitate extrem de largă.

Cu ajutorul circuitelor digitale, se pot memora informații și se pot efectua operații logice și matematice elementare sau complexe, precum și operații de diverse alte naturi, reductibile la primele.

Toate acestea au ca suport teoretic primar algebra booleană și aritmetica sistemului de numerație binar.

Un calculator constă, din punct de vedere arhitectural, în:

- procesor
- memorie
- interfețe
- dispozitive de introducere a datelor
- dispozitive de extragere a datelor
- memorii externe

Procesorul este partea unui calculator care are rolul de a implementa sarcinile de comandă pe care le implică acceptarea, reținerea, înțelegerea și executarea ordinelor –adică: execuția de programe- și de a efectua prelucrările propriu-zise. Blocurile funcționale ale unui procesor sunt:

- unitatea aritmetico-logică
- unitatea de comandă
- unitatea de registre

Unitatea aritmetico-logică este partea unui procesor care, aşa cum numele său o sugerează, are în sarcină efectuarea operațiilor aritmetice și logice. Alături de unitatea aritmetico-logică, în structura procesoarelor destinate utilizărilor în care e necesară putere de calcul ridicată există un bloc funcțional care implementează operații matematice complexe -practic toate funcțiile matematice uzuale- și care operează la rezoluții și precizii mai înalte, numit *coprocesor matematic*.

Unitatea de comandă este partea unui procesor care are în sarcină generarea comenzilor interne și externe necesare funcționării calculatorului în ansamblul său.

Unitatea de registre este partea unui procesor care reprezintă o memorie de manevră, realizată pe același cip și în aceeași tehnologie cu *unitatea aritmetico-logică* și cu *unitatea de comandă* și, în consecință, la fel de rapidă ca și acestea, la care accesele se fac mult mai simplu și eficient decât la memoria propriu-zisă.

Memoria este partea unui calculator care are rolul de a păstra **informațiile operative** la un moment dat, adică: codurile programelor aflate în rulare și datele asupra cărora ele acționează, respectiv pe care le produc.

Interfețele sunt entități structural-funcționale care au rolul de a asigura schimbul de informații între procesor sau memorie, pe de o parte, și dispozitivele de introducere a datelor, dispozitivele de extragere a datelor, respectiv memoriile externe, pe de altă parte.

Dispozitivele de introducere a datelor sunt entități structural-funcționale care au rolul de a intermedia transferul de informații de la utilizator către calculator. Dispozitivele de introducere a datelor cele mai uzuale sunt **tastatura, mouse-ul, joy-stick-ul și ecranul senzitiv**.

Dispozitivele de extragere a datelor sunt entități structural-funcționale care au rolul de a intermedia transferul de informații de la calculator către utilizator. Dispozitivele de extragere a datelor cele mai uzuale sunt **monitorul sau display-ul, imprimanta și ploterul**.

Memoriile externe sunt entități structural-funcționale cu rolul de a păstra **informațiile neoperative** la un moment dat (a se vedea, mai sus, ce se înțelege prin “**informații operative**”). Memoriile externe cele mai uzuale sunt: **unitatea de hard-disk, unitatea de CD / DVD, stick-ul**.

Dispozitivele de introducere a datelor, dispozitivele de extragere a datelor și memoriile externe sunt referite, generic, prin noțiunea de **echipamente periferice**.

Într-o reprezentare arhitecturală, într-un *overview*, un calculator poate fi schematizat aşa cum se arată în figura 1_1:

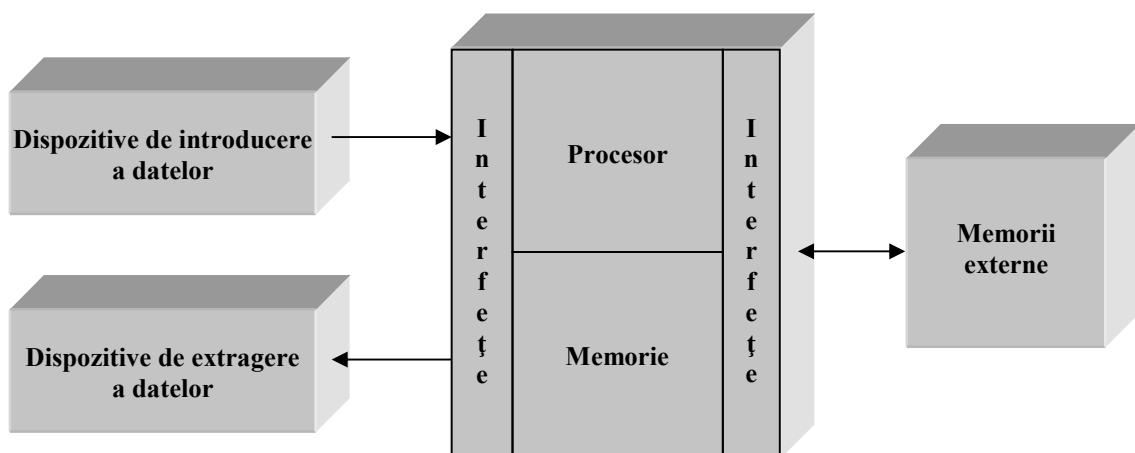


Fig. 1_1. Reprezentare arhitecturală a unui calculator.

Detaliind, tot în reprezentare arhitecturală, interiorul procesorului și evidențiind infrastructura de comunicare între diversele entități ce intervin, infrastructură formată din așa-numitele magistrala de date, magistrala de adrese și liniile de comandă, despre care vom vorbi mai târziu, obținem schema din figura 1_2:

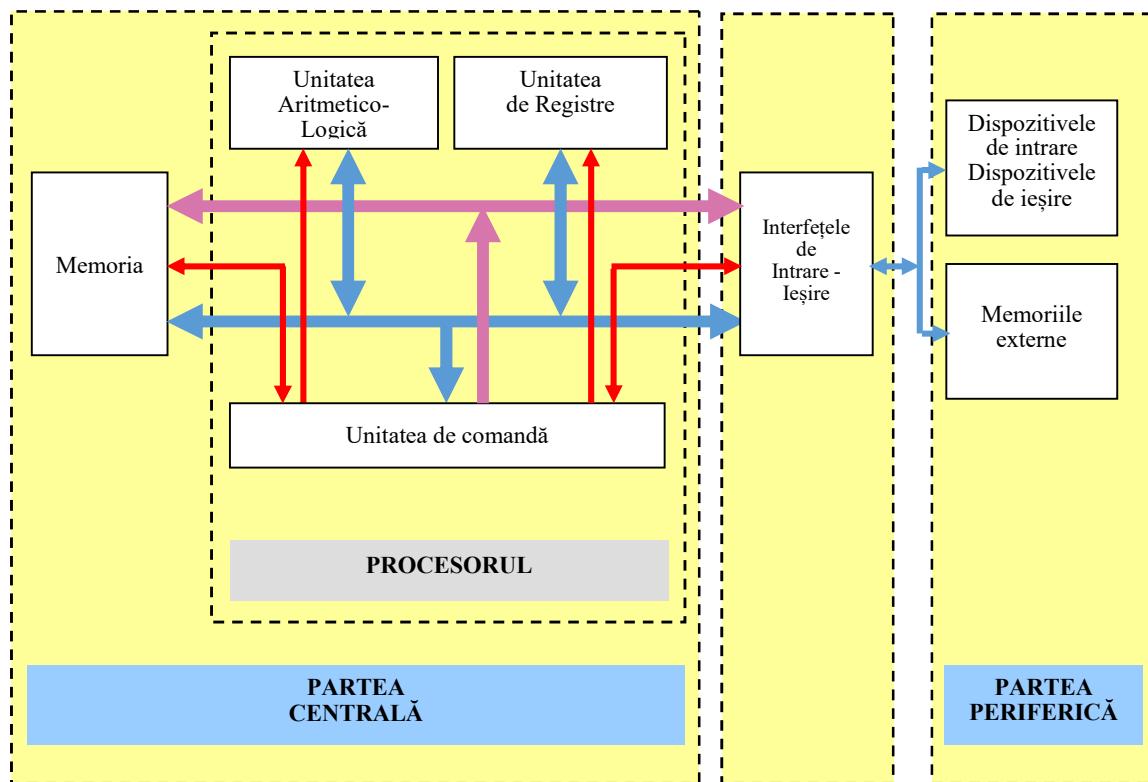


Fig. 1_2. Vedere arhitecturală asupra unui calculator,
cu detalii privind structura internă a procesorului

Legenda:

- magistrală de date
- magistrală de adrese
- linii de comandă

Se precizează că, până nu demult, noțiunea de calculator subsuma doar procesorul, memoria și interfețele, unii specialiști rămânând și astăzi cu această accepțiune asupra termenului. Noi opinăm că o asemenea abordare nu mai este în acord cu starea de fapt la care s-a ajuns în domeniul sub aspect constructiv, stare caracterizată prin aceea că cel puțin procesorul, memoria, interfețele și memoriile externe sunt înglobate în aceeași carcăsa, în timp ce unele modele de calculatoare –ne referim la așa-numitele *laptop-uri* sau *note-book-uri* – încorporează nu numai aceste entități, ci și o tastatură, un *mouse* (eventual și un *joy-stick*) și un *display*.

Trebuie spus că mașina aceasta numită calculator, deși deloc simplă, nu ar impresiona aşa cum o face, dacă nu ar funcționa într-un sistem mai complex, numit **sistem de calcul**, care cuprinde, alături de entitățile referite mai sus –care, toate, au o natură materială, fizică- și resurse de natură pur logică, nepalpabile, reprezentând aşa-numitele **programe de sistem**.

Menirea programelor de sistem este, pe de o parte, aceea de a face ca exploatarea părții fizice, precum și a lor însese să decurgă cât mai judicios și cât mai comod, iar pe de altă parte aceea de a conferi sistemului capabilități în plus față de cele prezentate de partea fizică.

Cele mai importante programe de sistem sunt:

- editoarele de texte^{*)}
- translatoarele de limbaj^{*)}
- programele de bibliotecă^{*)}
- editoarele de legături^{*)}
- depanatoarele^{*)}
- utilitarele
- *driver-ele* de intrare-iesire
- sistemul de operare

Editoarele de texte sunt programe cu ajutorul cărora se scriu textele programelor, dar, posibil, și texte oarecare. Prin acțiunea editoarelor de texte se generează aşa-numitele **fișiere-sursă**, care conțin textele programelor și a căror denumire este, în partea sa numită "extensie", specifică limbajului de programare utilizat (de exemplu: ".c" -pentru C, ".cpp" -pentru C++, etc.).

Translatoarele de limbaj sunt programe cu ajutorul cărora se efectuează traducerea programelor utilizatorilor din limbajul de programare în care au fost scrise, în limbajul mașinii numită calculator sau, altfel spus: în **cod-mașină**. Există trei tipuri de translatoare de limbaj, denumite:

- compilatoare
- asamblare
- interpretoare

Compilatoarele și asambloarele sunt programe care la un apel al lor traduc un întreg program sau modul de program și, în final, generează codul-mășină corespunzător, într-un fișier numit "**fișier-obiect**", al cărui nume are extensia ".obj". Diferența dintre compilatoare și asambloare rezidă în faptul că primele traduc programe scrise în limbaje de nivel înalt, iar cele din urmă – programe scrise în limbaj de asamblare.

Interprotoarele sunt programe care traduc o linie a unui program și introduc imediat în rulare codul-mășină rezultat, apoi trec la linia următoare și.a.m.d., fără ca în final să ofere codul-mășină al întregului program sau modul de program, cum am văzut că fac compilatoarele și asambloarele. Rezultă, aşadar, că în cazul folosirii interprotoarelor, de fiecare dată când se dorește rularea unui program, se impune o nouă traducere și că, inevitabil, timpul necesar rulării programului este prelungit cu timpul de traducere.

Programele de bibliotecă sunt programe care pun la dispoziția utilizatorilor o serie de facilități, ce se adaugă celor pe care le prezintă limbajele de programare însăși.

Editoarele de legături sunt programe care au rolul de a interconecta codul-obiect al programului utilizatorului, respectiv codurile-obiect ale modulelor în care acesta consistă, cu coduri-obiect de programe de bibliotecă, de *driver*-e de intrare-iesire și-sau de programe ținând de sistemul de operare, după caz, astfel încât ceea ce se obține să reprezinte un program executabil; acesta face obiectul unui fișier numit "**fișier-executabil**", caracterizat prin aceea că poartă un nume cu extensia ".exe" sau, uneori, ".com".

Depanatoarele sunt programe cu ajutorul cărora se pun la dispoziția utilizatorilor o seamă de facilități de depanare a programelor: execuție pas cu pas, oprire la o anumită linie, vizualizarea/modificarea valorilor unor variabile, etc.

Toate aceste programe -marcate cu *) în enumerarea de mai sus- sunt integrate în așa-numitele **medii de dezvoltare a aplicațiilor**, fără, însă, ca, în cadrul acestora, să-și piardă identitatea funcțională.

Utilitarele sunt programe cu ajutorul cărora se efectuează operații cu un grad ridicat de uzualitate, cum sunt: crearea, copierea, mutarea fișierelor, etc.

Driver-ele de intrare-iesire sunt programe cu ajutorul cărora se realizează introducerea, respectiv extragerea datelor, evident, într-o strânsă conlucrare cu interfețele de intrare-iesire.

Sistemul de operare este un conglomerat de programe care au rolul generic de a gestiona resursele sistemului, interpunându-se între acestea și utilizator. Metaoric, se poate spune că un sistem de operare joacă rol de dirijor, orchestrând doleanțele utilizatorilor prin atribuirea de partituri adecvate către diversele resurse și vechind la interpretarea armonizată a respectivelor partituri, astfel încât scopurile urmărite să fie atinse.

1.2. Acțiuni ce au loc la pornirea unui calculator

În general, la pornirea unui calculator, intră în rulare, în mod automat, un program păstrat în partea *nevولاتیلă* (a se citi: care își păstrează conținutul intact și în lipsa tensiunii de alimentare) a memoriei calculatorului, instantiată, de obicei, în varianta cunoscută sub numele **“EPROM”**, program care are ca misiune **identificarea configurației** în care se află calculatorul, **efectuarea de teste** de bună funcționare a principalelor entități componente, **încărcarea** în memorie, de pe *hard-disk*, a componentelor **sistemului de operare și lansarea** acestuia **în execuție**. O dată intrat în execuție, sistemul de operare se va îngriji ca tot ce se va întâmpla mai departe pe calculator să fie sub controlul sau, cel puțin, cu consimțământul său. Astfel, el va asigura afișarea meniurilor cu alternativele ce stau în fiecare moment când el este în execuție la dispoziția utilizatorului, preluarea și introducerea în fluxul de procesare a informațiilor specificate de utilizator cu ajutorul tastaturii, *mouse-ului*, etc., lansarea în execuție, după caz, a altor programe de sistem -medii de programare, utilitare- sau a unor programe ale utilizatorului, etc.

2 DESPRE MEMORIE

În acest capitol, ne propunem să vorbim despre memoria calculatoarelor doar atât cât este necesar pentru înțelegerea a ceea ce ea reprezintă, a principiilor organizării interne și a modului în care ea funcționează la nivel bloc. Și toate acestea, pentru ca mai târziu, pe parcursul capitolului 6, să existe premizele unei facile și corecte înțelegeri a conlucrării dintre procesor și memorie, a funcționalității ansamblului procesor-memorie și, în definitiv, a însăși fenomenologiei computației.

Memoria unui calculator trebuie percepută ca *tabelă de locații* asimilabile, fiecare, câte unui registru pe 8 biți și având asociate, în mod injectiv, informații de reperare cunoscute sub numele de *adrese*.

Deși în calculatoarele actuale este obișnuit ca, în mod dinamic, după necesități, memoriile să fie accesate fie la nivel de octet, fie la nivel de dublu-octet, numit *cuvânt*, fie la nivel de cuadruplu-octet, numit *cuvânt lung*, și chiar la nivel mult mai extins, noi vom considera, simplificator, în continuare, în acest capitol, cazul memoriilor capabile să opereze doar la nivel de octet. În această ipoteză, o memorie se prezintă, la nivel bloc, ca entitate prevăzută cu:

- 8 borne de intrare-iesire –deci: bidirectionale-, dedicate vehiculării informațiilor de scris în ea, respectiv citite din ea
- 1 bornă de intrare de specificare a operației ce se execută la un moment: scriere, respectiv citire
- $\log_2 N$ borne de intrare de adresare, unde N reprezintă capacitatea memoriei, adică: numărul de locații cu care memoria este înzestrată
- 1 bornă de intrare de validare / invalidare
- 1 bornă de ieșire de confirmare a efectuării complete a operației ce a făcut obiectul unei sesiuni de lucru la un moment dat

Cele 8 borne de intrare-iesire, numite **borne de date**, se notează, de obicei, cu **D0-D7**. Borna de specificare a operației ce se execută la un moment dat, numită **bornă de citire-scriere**, se notează cu **R/W** (abreviere de la “**read / write**”). **Borna de validare / invalidare**, numită și **bornă de selecție** se notează cu **MRQ** (abreviere de la “**memory request**”), iar bornele de adresare, numite și **borne de adrese** -cu **A0-AX**, unde $X = \log_2 N - 1$. Borna de **confirmare** se notează cu **ACK** (abreviere de la “**acknowledge**”). Cu aceste notații –precizăm: regăsibile mai mult sau mai puțin ca atare în documentații, întrucât ele nu corespund decât întâmplător unor procesoare concrete-, vom putea reprezenta o memorie aşa cum se arată în figura 2_1.

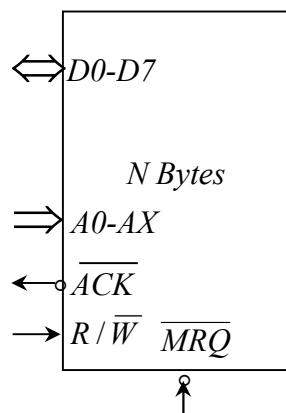


Fig. 2_1. Reprezentarea bloc a unei memorii.

Memoria efectuează o operație de citire sau de scriere, după caz, doar atunci când semnalul de la intrarea **MRQ** are valoarea logică “0”; altfel, ea își conservă conținutul și își ține bornele **D0-D7** în starea de înaltă impedanță.

Când semnalul de la intrarea **MRQ** este “0”, dacă semnalul de la borna **R/W** este “1”, atunci se execută o operație de citire din locația a cărei adresă este stabilită la bornele **A0-AX**. Informația citită este oferită la bornele **D0-D7** după un interval de timp numit “**temp de acces**”, măsurat începând din momentul în care ultimul dintre semnalelele **A0-AX**, **MRQ** și **R/W** ajunge să fie stabil.

Când semnalul de la intrarea **MRQ** este “0”, dacă semnalul de la borna **R/W** este “0”, atunci se execută o operație de scriere în locația a cărei adresă este stabilită la bornele **A0-AX**, a informației prezente la bornele **D0-D7**. Scrierea se încheie la un anumit interval de timp după momentul în care ultimul dintre semnalelele **A0-AX**, **MRQ** și **R/W** ajunge să fie stabil.

Atât în cazul citirii, cât și în cazul scrierii, în final, se activează semnalul \overline{ACK} , ca semn, la citire, că datele citite au ajuns să fie disponibile la bornele $D0-D7$, iar la scriere, că datele de scris sunt deja scrise. Semnalul \overline{ACK} este destinat procesorului (mai exact: unității de comandă a procesorului), care se informează cu ajutorul lui cât trebuie să țină activ semnalul \overline{MRQ} . La trecerea semnalului \overline{ACK} la “0”, procesorul înțelege că memoria și-a încheiat ciclul ce i-a fost cerut și o dezleagă temporar de sarcini, trecând semnalul care a ținut-o activă –este vorba despre \overline{MRQ} - la “1”.

În figura 2_2, se prezintă schema unei memorii organizată și capabilă să opereze la nivel de octet -și numai la nivel de octet- și având capacitatea de 64 KBytes. S-a presupus că această memorie este realizată cu ajutorul a 4 capsule de câte 16 KBytes fiecare, prevăzute –evident: pe lângă bornele de alimentare- cu 14 borne de adrese ($14 = \log_2(16K) = \log_2(16*2^{10}) = \log_2(2^4*2^{10}) = \log_2 2^{14}$), notate $A0-A13$, 8 borne de date, notate $D0-D7$, 1 bornă de citire-scriere, notată R/\overline{W} și 1 bornă de selecție, notată \overline{CS} (abreviere de la “chip select”). Pentru ca cele 4 capsule să poată constitui o unitate de memorie, a fost necesar ca ele să fie conectate bornă la bornă, excepție de la această regulă făcând bornele \overline{CS} , care se conectează separat, fiecare la câte o ieșire a circuisticii externe de decodificare a adreselor.

Circuistica externă de decodificare a adreselor va fi validă –adică: va putea efectua decodificarea combinației de valori logice prezente pe liniile $A15$ și $A14$ - doar în timp ce semnalul \overline{MRQ} este “0”; valoarea “1” a semnalului \overline{MRQ} face ca toate ieșirile circuisticii de decodificare să fie ținute la valoarea lor inactivă, adică: la ”1”. Când semnalul \overline{MRQ} ia valoarea “0”, una dintre ieșirile circuisticii de decodificare va comuta la “0”. Prin consecință, una dintre cele patru capsule de memorie din figura 2_2, va ajunge să fie selectată, procedând, după caz, la efectuarea unei operații de citire sau a unei operații de scriere.

Câteva remarci se impun în privința decodificării adreselor. Să presupunem că dispunem de un procesor cu capacitatea de a adresa o memorie de capacitate de până la N locații. Asta înseamnă că el dispune de un număr de liniile de adresare $X = \log_2 N$. Dacă memoria este realizată cu capsule cu capacitatea de M locații, atunci, dintre cele X liniile de adrese ale procesorului, $Y = \log_2 M$ și anume, cele de indici $0...Y-1$ vor fi decodificate în interiorul capsulelor, iar diferența de $X - Y$, cele mai semnificative, începând cu cea de indice Y , vor fi decodificate în exteriorul capsulelor, printr-o circuistică externă de decodificare cu $X-Y$ intrări și 2^{X-Y} ieșiri.

Fiecare dintre aceste ieșiri va selecta câte o capsulă de M locații. În exemplul din figura 2_2, avem:

- capacitatea de adresare a procesorului: 64 Klocătii, deci $X = \log_2(64K) = \log_2(64*2^{10}) = \log_2(2^6*2^{10}) = \log_22^{16} = 16$
- capacitatea unei capsule: 16 Klocătii, deci $Y = \log_2(16K) = \log_2(16*2^{10}) = \log_2(2^4*2^{10}) = \log_22^{14} = 14$
- numărul liniilor de adresare decodificate în exteriorul capsulelor: $X - Y = 16 - 14 = 2$
- cele 14 linii de adrese decodificate în interiorul capsulelor sunt $A_0 - A_{13}$, cele 2 linii de adrese decodificate în exteriorul capsulelor sunt $A_{14} - A_{15}$

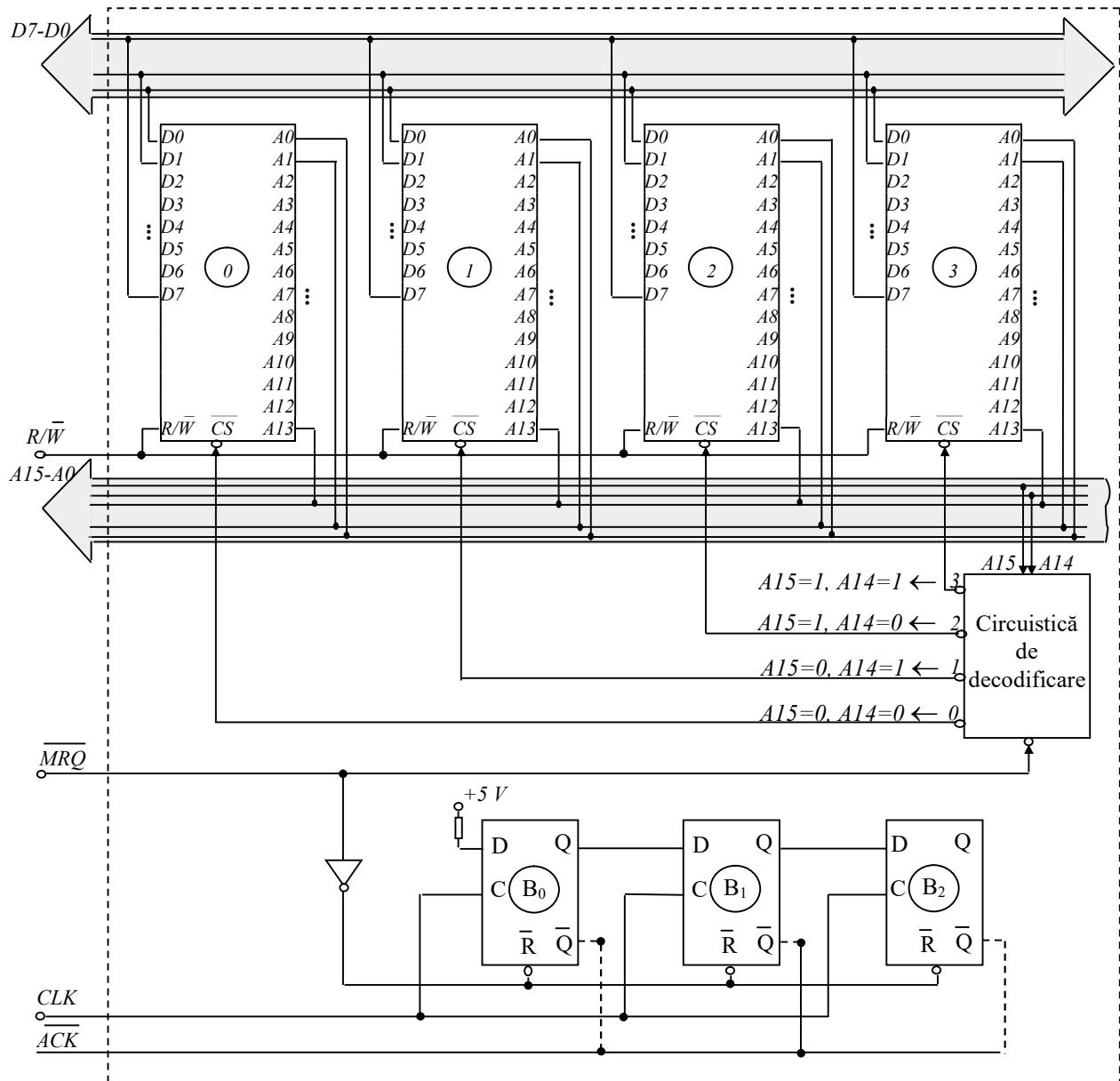


Fig. 2_2. Un exemplu de memorie.

Observatii:

În figura 2_2:

[capitol - adr. 1 capsula]

Capsula 0 conține locațiile de memorie cu adresele hexazecimale 0000-3FFF.

Capsula 1 conține locațiile de memorie cu adresele hexazecimale 4000-7FFF.

Capsula 2 conține locațiile de memorie cu adresele hexazecimale 8000-BFFF.

Capsula 3 conține locațiile de memorie cu adresele hexazecimale C000-FFFF.

Acum lucru devine evident dacă exprimăm adresele în binar: $Y = 14$ (linii)

Capsula	Adresele															
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0				0				0				0			
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	⋮				⋮				⋮				⋮			
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	3				F				F				F			
	4				0				0				0			
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	⋮				⋮				⋮				⋮			
2	7				F				F				F			
	8				0				0				0			
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	⋮				⋮				⋮				⋮			
3	B				F				F				F			
	C				0				0				0			
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	⋮				⋮				⋮				⋮			
	F				F				F				F			

În partea inferioară a schemei din figura 2_2, este prezentă circuitistica de generare a semnalului \overline{ACK} , concepută în ipoteza că atât la citire, cât și la scriere, memoria răspunde cererii procesorului în cel mult $2 \frac{1}{2}$ perioade ale semnalului de tact, de la trecerea semnalului \overline{MRQ} la "0". Evident, dacă durata acestor operații ar fi mai mare de $2 \frac{1}{2}$ perioade ale semnalului de tact, atunci celor trei bistabile ale schemei ar trebui să li se adauge și altele,

conectate după aceeași logică. În figurile 2_3.a - 2_3.c, sunt redate cronogramele semnalelor \overline{MRQ} și \overline{ACK} , pentru cazurile $ta \in [0 \dots 0.5pt]$, $ta \in [0.5pt \dots 1.5pt]$, $ta \in [1.5 \dots 2.5pt]$, unde cu ta s-a notat *timpul de acces la memorie*, iar cu pt – perioada de tact.

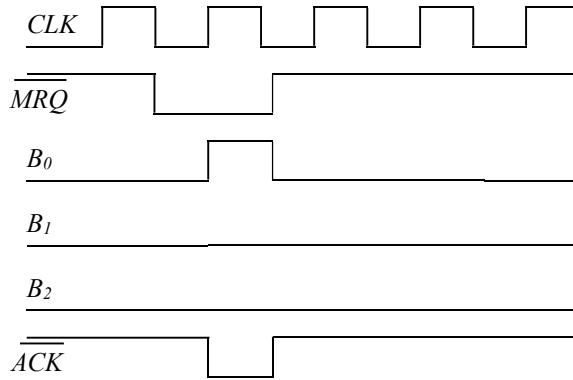


Fig. 2_3.a. Cronograma semnalelor \overline{MRQ} și \overline{ACK}
-cazul unui timp de acces de 0...0.5 perioade de tact-

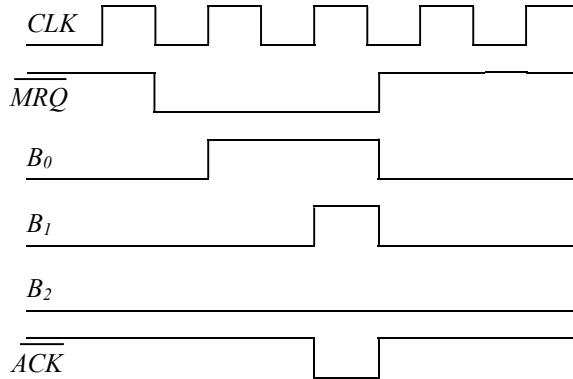


Fig. 2_3.b. Cronograma semnalelor \overline{MRQ} și \overline{ACK}
-cazul unui timp de acces de 0.5...1.5 perioade de tact-

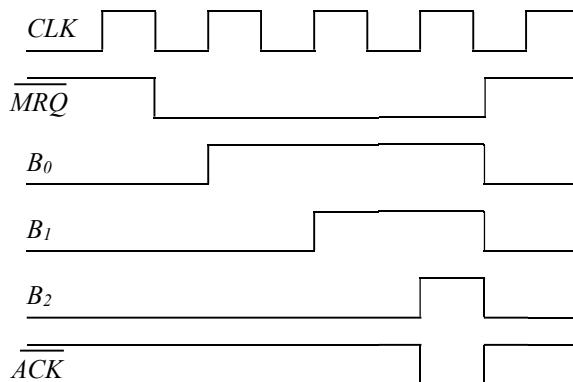


Fig. 2_3.c. Cronograma semnalelor \overline{MRQ} și \overline{ACK}
-cazul unui timp de acces de 1.5...2.5 perioade de tact-

Până acum, s-a presupus că fiecare capsulă este organizată pe locații de 1 octet (8 biți / locație). Este, însă, posibil, ca organizarea internă a capsulelor să fie pe locații de câte 1 bit, 2 biți sau 4 biți. Într-o asemenea situație, un număr de **8 / K capsule**, unde **K = 1, 2 sau 4** vor fi conectate între ele bornă la bornă, mai puțin bornele de date, care vor fi relaționate fiecare cu câte o linie a magistralei de date. Putem afirma, deci, că, procedând aşa cum s-a arătat, în cele trei cazuri, cu 8, 4, respectiv 2 capsule, organizate pe 1 bit, 2 biți, respectiv 4 biți per locație se va constitui capsula echivalentă pe 8 biți. Figurile 2_3, 2_4, respectiv 2_5 surprind acest lucru.

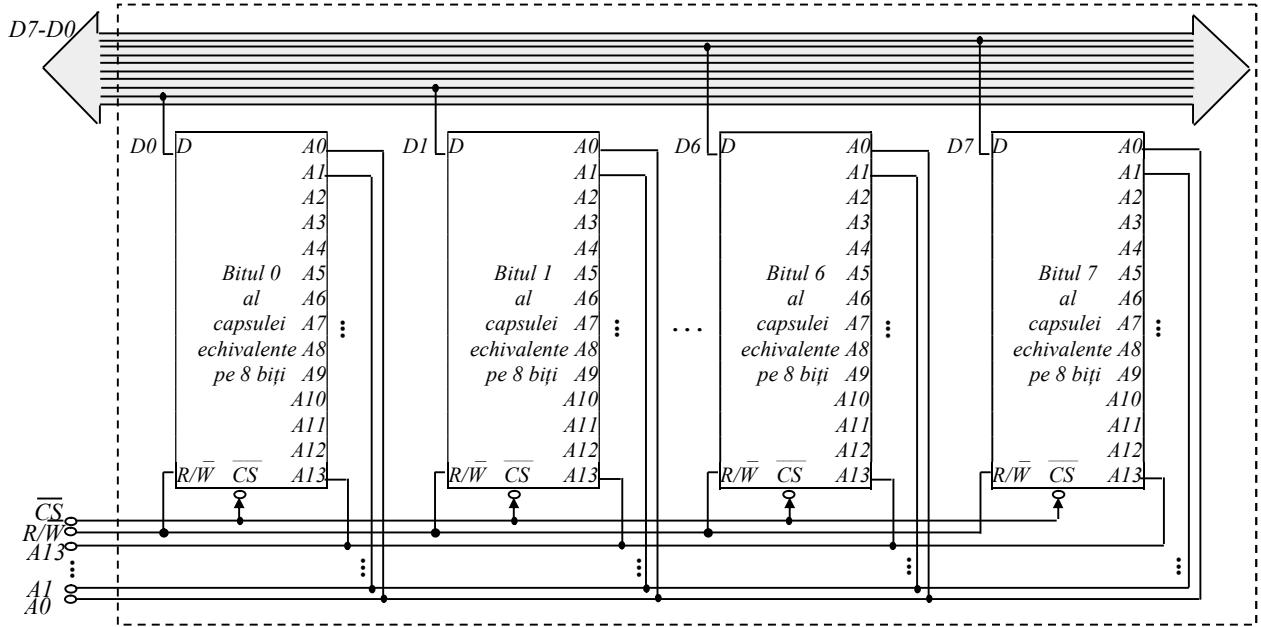


Fig. 2_3. Capsula echivalentă de 16 Kbytes, realizată cu 8 capsule de 16 Klocații de 1 bit per locație.

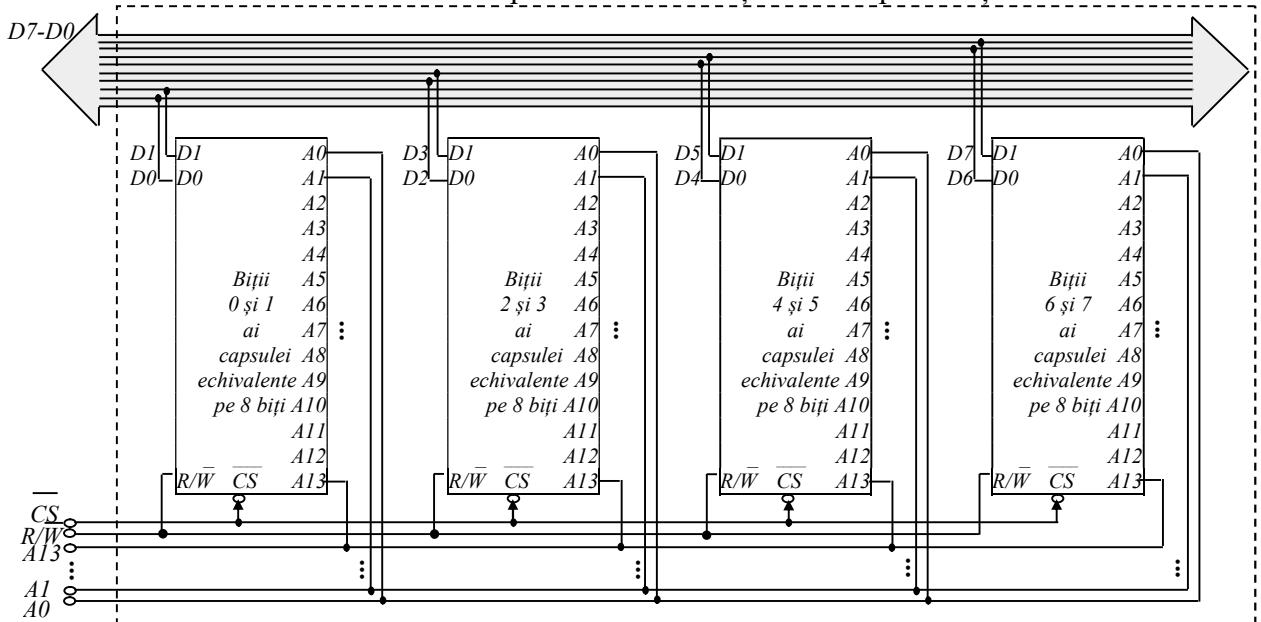


Fig. 2_5. Capsula echivalentă de 16 Kbytes, realizată cu 4 capsule de 16 Klocații de 2 biți per locație.

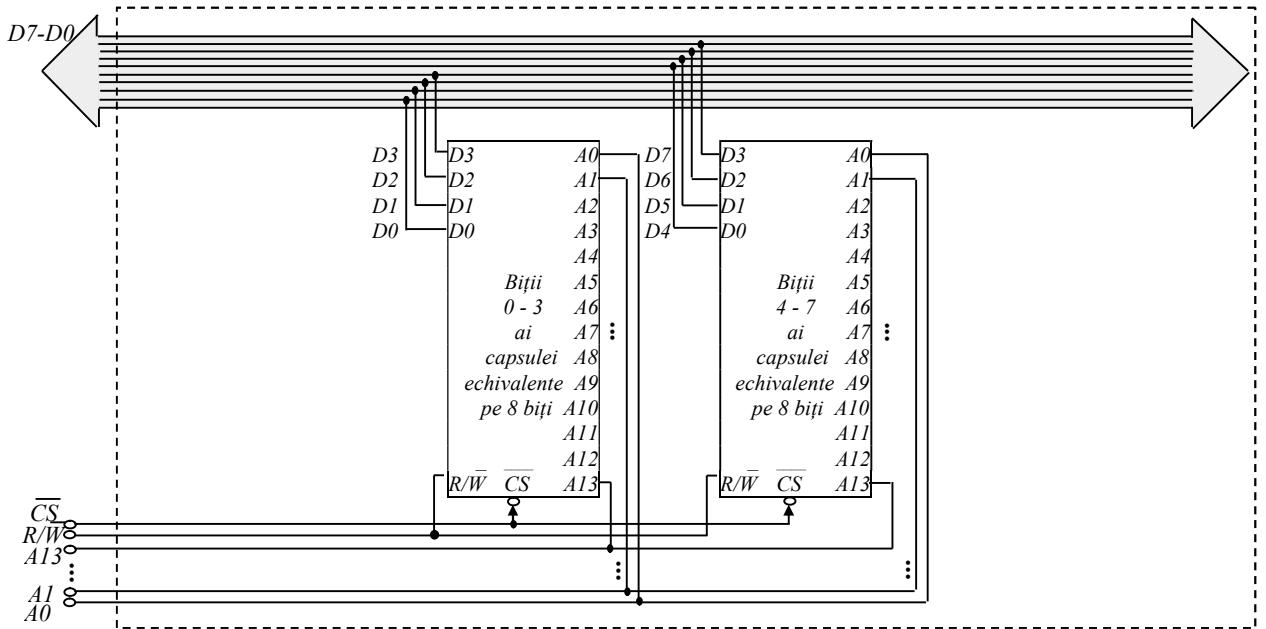


Fig. 2_5. Capsula echivalentă de 16 Kbytes, realizată cu 2 capsule de 16 Kbytes de la 4 biți per locație.

Memoria oricărui calculator cuprinde o parte exploataată atât în regim de citire cât și în regim de scriere, numită **RAM** –de la Random Access Memory-, și o parte exploataată doar în regim de citire, numită **ROM** –de la Read Only Memory-. Este important de reținut că memoria **RAM** păstrează informațiile memorate în ea intacte doar câtă vreme este alimentată corespunzător, în timp ce memoria **ROM** își păstrează conținutul intact și în lipsa tensiunii de alimentare. Se spune că memoria **RAM** este *volatile* și că memoria **ROM** este *nevolatile*.

În cazul calculatoarelor universale –a se citi: de uz nespecific-, partea de memorie **RAM** este de departe majoritară. La aceste calculatoare, memoria de tipul **ROM** conține doar programul de *boot*-are, adică: un program introdus implicit în rulare la pornire și care are ca misiune **identificarea configurației** în care se află calculatorul, **efectuarea de teste de bună funcționare a principalelor entități componente, încărcarea**, dintr-o memorie externă –de regulă: de pe *hard-disk*-, a componentelor **sistemului de operare și lansarea în execuție a acestuia**. De asemenea, tot în memoria **ROM**, se obișnuiește să se păstreze și *driver*-ele sau o parte a *driver*-elor de intrare ieșire. Celealte programe –de sistem sau de aplicație- se păstrează pe suporturile externe de memorare. De aici, ele se încarcă doar temporar în memoria **RAM**, în vederea rulării. Evident, când spunem că un program se află încărcat pentru rulare, înțelegem că partea lui de cod se află în memorie și că o anumită cantitate din memorie este la dispoziția sa pentru implementarea diverselor variabile și pentru anumite manevre.

În cazul calculatoarelor dedicate unor aplicații –se au, aici, în vedere, în primul rând, așa-numitele “*embedded systems*”-, memoria este preponderent de tip *ROM*. În ea se găsește programul aferent aplicației, programe conexe acestuia și, eventual, un minim de programe de sistem, iar memoria *RAM* servește doar implementării variabilelor și operațiilor de manevră.

Există mai multe **tipuri de memorie ROM**:

- memoria *ROM* propriu-zisă
- memoria *PROM*
- memoria *EPROM*
- memoria *EEPROM*
- memoria *FLASH*

Memoria *ROM* propriu-zisă se caracterizează prin aceea că este înscrisă prin însuși procesul său de fabricație, în mod imuabil.

Memoria *PROM* se caracterizează prin aceea că este înscrisă de utilizator, printr-un proces specific, cu ajutorul unui dispozitiv dedicat, numit “programator de *PROM*-uri” sau “arzător de *PROM*-uri”. O dată înscrisă, o asemenea memorie nu mai acceptă nici un fel de modificare a conținutului său, putând, în continuare, fi doar citită. *PROM* vine de la ***Programmable Read Only Memory***.

Memoria *EPROM* se caracterizează prin aceea că este înscrisă de utilizator, printr-un proces specific, cu ajutorul unui dispozitiv dedicat, numit “programator de *EPROM*-uri”, după ce a fost pregătită pentru aceasta prin iradiere cu raze ultraviolete. Iradierea are ca efect ștergerea memoriei, adică: aducerea tuturor celulelor sale la “1”. O memorie *EPROM* concretă poate fi ștearsă și reprogramată de un număr de ori foarte ridicat, reprezentând o dată de catalog a circuitului respectiv. *EPROM* vine de la ***Erasable Programmable Read Only Memory***.

Memoria *EEPROM* se caracterizează prin aceea că este înscrisă de utilizator, printr-un proces specific, după ce a fost pregătită pentru aceasta prin ștergere electrică; atât ștergerea, cât și înscrierea memoriilor *EEPROM* se pot face chiar în contextul circuistic în care ele sunt exploataate. O memorie *EEPROM* concretă poate fi ștearsă și reprogramată de un număr de ori foarte ridicat, reprezentând o dată de catalog a circuitului respectiv. *EEPROM* vine de la ***Electrical Erasable Programmable Read Only Memory***.

Memoria *FLASH* este o variantă de memorie *EEPROM*, la care ștergerea și înscrierea se fac tot în contextul circuistic în care ea este exploataată, dar ștergerea necesită mult mai puțin timp decât la *EEPROM*. Memoriile *flash* au ajuns, astăzi, la capacitateuri uriașe, de-a dreptul. Le întâlnim pe de o parte în microcontrolere, în diverse dipozitive specializate, cum ar fi cele dedicate jocurilor, dar și în componența multor calculatoare, dedicate sau universale, realizate cu microprocesoare, respectiv, cel mai la vedere, în atât de folositele de către noi *stick*-uri de memorie externă.

3 DESPRE INTERFEȚE

Așa cum s-a precizat în capitolul 1, interfețele din structura unui calculator au rolul de a asigura schimbul de informații între procesor sau memorie, pe de o parte și dispozitivele de introducere a datelor, dispozitivele de extragere a datelor, respectiv memoriile externe, pe de altă parte.

O interfață constă în unul sau mai multe *porturi* și, eventual, o circuistică conexă, ce îndeplinește funcții de comandă locală.

Prin termenul de *port* se desemnează ansamblul format dintr-un regiszru - dimensionat, de regulă, la 8 biți- și o circuistică de comandă aferentă lui, având rolul de a prelua date de la echipamentele periferice, respectiv de a scoate date spre echipamentele periferice, precum și de a materializa diversele sarcini de comandă aferente transferului datelor.

Porturile dedicate preluării datelor de la echipamentele periferice se numesc “*porturi de intrare*”.

Porturile dedicate scoaterii datelor spre echipamentele periferice se numesc “*porturi de ieșire*” (evident, porturile folosite pentru materializarea de sarcini de comandă sunt, inevitabil, fie de intrare, fie de ieșire).

În principiu, un *port de intrare* –vezi figura 3_1- este prevăzut cu:

- 8 borne de intrare dedicate preluării datelor de la echipamentul periferic, numite “*borne de intrare date*” și notate *I0...I7*;
- 1 bornă de intrare de comandă a scrierii în port a datelor prezente pe liniile *I0...I7*, numită “*bornă de scriere*” și notată *EXTWR* (abreviere de la “*external write*”);
- 8 borne de ieșire dedicate transmiterii datelor spre procesor, numite “*borne de ieșire date*” și notate *O0...O7*;

- 1 bornă de intrare de comandă a disponibilizării către procesor a datelor înscrise în port, numită “*bornă de citire*” și notată \overline{RD} (abreviere de la “*read*”);
- 1 bornă de ieșire de semnalare a faptului că în port există o dată înscrisă și încă necitită, numită “*bornă de semnalare*” și notată IBF (abreviere de la “*input buffer full*”);
- 1 bornă de intrare de validare/invalidare, numită “*bornă de selecție*” și notată \overline{CE} sau CS (abrevieri de la “*chip enable*”, respectiv “*chip select*”).

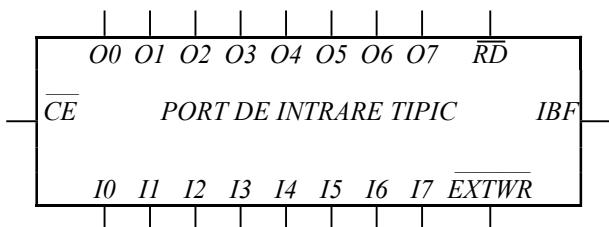


Fig. 3_1. Bornele clasice ale unui port de intrare.

Introducerea datelor în calculator cu ajutorul unui port de intrare tipic este ilustrată prin cronograma din figura 3_2.

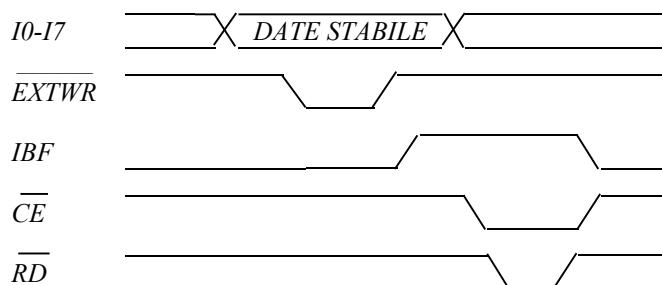


Fig. 3_2. Cronograma semnalelor ce intervin la introducerea datelor în calculator cu ajutorul unui port de intrare tipic.

Cronograma din figura 3_2 vrea să spună următoarele lucruri:

Când datele pe care perifericul trebuie să le introducă în calculator sunt stabilă, are loc activarea de către periferic a semnalului \overline{EXTWR} , prin trecerea sa la nivel “0”. Valoarea activă a semnalului \overline{EXTWR} determină preluarea în port a datei prezente la bornele sale $I0-I7$. Buffer-ul portului ajungând astfel plin, semnalul fanion IBF trece la “1”. Prin testarea acestui semnal, procesorul află că o dată este pregătită pentru el în port și execută o operație de citire cu referire la el, preluând data prin bornele $O0-O7$, sub

comanda semnalului \overline{RD} . Evident, în cadrul operației de citire, procesorul plasează pe magistrala de adrese adresa portului în cauză, adresă a cărei decodificare conduce la activarea semnalului \overline{CE} . La încheierea operației de citire, portul își va trece din nou semnalul IBF la “0”, făcând, astfel, cunoscut, atât pentru procesor, cât și pentru periferic, faptul că *buffer*-ul de intrare este gol. Pentru a se evita pierderea de date prin suprascrivere – suprascrierea înseamnă preluarea în port a unei noi date înainte ca precedenta să fi fost citită-, perifericul trebuie să nu activeze semnalul \overline{EXTWR} decât în timp ce semnalul IBF este la “0”. Pe de altă parte, procesorul trebuie să nu citească portul decât în timp ce semnalul IBF este la “1”, pentru că, altfel, va accesa fie o dată fără semnificație, stabilită în port cu ocazia punerii sub tensiune, fie o dată care a fost deja citită.

În principiu, un **port de ieșire** –vezi figura 3_3- este prevăzut cu:

- 8 borne de intrare dedicate preluării datelor de la procesor, numite “*borne de intrare date*” și notate $I0...I7$;
- 1 bornă de intrare de comandă a scrierii în port a datelor prezente pe liniile $I0...I7$, numită “*bornă de scriere*” și notată \overline{WR} (abreviere de la “*write*”);
- 8 borne de ieșire dedicate transmiterii datelor spre echipamentul periferic, numite “*borne de ieșire date*” și notate $O0...O7$;
- 1 bornă de intrare de informare a portului asupra faptului că echipamentul periferic a preluat datele înscrise în el, numită “*bornă de citire*” și notată \overline{EXTRD} (abreviere de la “*external read*”);
- 1 bornă de ieșire de semnalare a faptului că în port există o dată înscrisă și încă necitită, numită “*bornă de semnalare*” și notată \overline{OBF} (abreviere de la “*output buffer full*”);
- 1 bornă de intrare de validare/invalidare, numită “*bornă de selecție*” și notată \overline{CE} sau \overline{CS} (abrevieri de la “*chip enable*”, respectiv “*chip select*”).

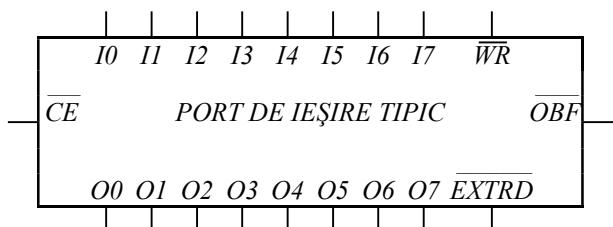


Fig. 3_3. Bornele clasice ale unui **port de ieșire**

Extragerea datelor din calculator cu ajutorul unui port de ieșire tipic este ilustrată prin cronograma din figura 3_4.

Cronograma din figura 3_4 vrea să spună următoarele lucruri:

Când procesorul vrea să scoată o dată spre periferic, execută o operație de scriere cu referire la port, ceea ce provoacă activarea semnalului \overline{WR} , prin trecerea sa la "0". Evident, în cadrul operației de scriere, procesorul va plasa pe magistrala de adrese adresa portului în cauză, adresă a cărei decodificare va conduce la activarea semnalului \overline{CE} . Valoarea activă a semnalului \overline{WR} determină preluarea în port a datei prezente la bornele sale $I0-I7$. Buffer-ul portului ajungând astfel plin, semnalul fanion \overline{OBF} trece la "0". Prin testarea acestui semnal, perifericul află că o dată este pregătită pentru el în port și execută o operație de citire, preluând data prin bornele $O0-O7$, sub comanda semnalului \overline{EXTRD} . La încheierea operației de citire, portul își va trece din nou semnalul \overline{OBF} la "1", făcând, astfel, cunoscut, atât pentru procesor, cât și pentru periferic, faptul că buffer-ul de ieșire este gol. Pentru a se evita pierderea de date prin suprascriere, procesorul trebuie să nu scrie în port decât în timp ce semnalul \overline{OBF} este la "1". Pe de altă parte, perifericul trebuie să nu citească portul decât în timp ce semnalul \overline{OBF} este la "0", pentru că, altfel, va accesa fie o dată fără semnificație, stabilită în port cu ocazia punerii sub tensiune, fie o dată care a fost deja citită.

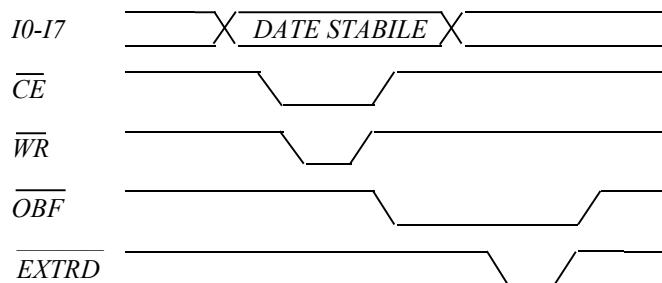


Fig. 3_4. Cronograma semnalelor ce intervin la extragerea datelor din calculator cu ajutorul unui port de ieșire tipic.

Se precizează că bornele de ieșire date ale porturilor de intrare sunt ținute, în cea mai mare parte a timpului, în starea de înaltă impedanță, fiind activate doar în timp ce un astfel de port este citit efectiv de către procesor,

adică: în timp ce atât semnalul \overline{CE} , cât și semnalul \overline{RD} sunt active. De asemenea, se menționează că semnalele \overline{RD} și \overline{WR} au efecte doar în timp ce portul este validat de semnalul \overline{CE} , în timp ce semnalele \overline{EXTRD} și \overline{EXTWR} nu sunt legate de semnalul de validare / invalidare.

Ca și locațiile de memorie, porturile au asociate adrese. Adresele de porturi pot fi parte a unui spațiu unic de adrese la nivelul calculatorului, alături de adresele de locații de memorie sau pot constitui un spațiu de adrese distinct, complementar spațiului de adrese de locații de memorie. Din considerente didactice, vom merge pe varianta din urmă, adică: vom presupune că se dispune de un spațiu de adrese de locații de memorie, având ca “martor” semnalul \overline{MRQ} , și de un spațiu de adrese de porturi, având ca “martor” un semnal numit \overline{IORQ} . O adresă de o anumită valoare se referă, la un moment, la o locație de memorie sau la un port, după cum în acel moment este activ semnalul \overline{MRQ} sau, dimpotrivă, semnalul \overline{IORQ} . Dacă nici unul dintre aceste semnale nu este activ, atunci codul prezent la bornele de adrese nu reprezintă o adresă –de fapt, nu are nicio semnificație-și, în consecință, el nu trebuie să conducă nici la selecția unei locații de memorie și nici la selecția unui port. Acest lucru se asigură decodificând adresele condiționat de semnalele \overline{MRQ} , respectiv \overline{IORQ} .

Porturile se introduc în schemele calculatoarelor conectând între ele, bornă la bornă, bornele de ieșire date ale porturilor de intrare, bornele de intrare date ale porturilor de ieșire și bornele de intrare-ieșire date ale memoriei. Conectarea nu trebuie înțeleasă obligatoriu directă, ci posibil prin intermediul unor circuite tampon.

4 DESPRE PROCESOR. UNITATEA ARITMETICO-LOGICĂ

4.1. Reprezentarea numerelor în calculator

4.1.1. Reprezentarea numerelor în virgulă fixă

4.1.1.1. Reprezentarea numerelor pozitive

Într-un sistem de numerație de bază r , un număr K se exprimă prin notația:

$$\tilde{K} = c_{n-1}c_{n-2}...c_1c_0.c_{-1}...c_{-(m-1)}c_{-m}, \quad 0 \leq c_i < r \quad (4.1.1.1-1)$$

și are valoarea:

$$K = c_{n-1} \cdot r^{n-1} + \dots + c_1 \cdot r + c_0 + c_{-1} \cdot r^{-1} + \dots + c_{-m} \cdot r^{-m} = \sum_{i=n-1}^{-m} c_i \cdot r^i \quad (4.1.1.1-2)$$

Se spune că notația (1) este pozițională, deoarece fiecare cifră intervine în valoarea numărului cu o pondere dată de poziția sa (“ r^i ” pentru cifra din poziția “ i ”, valoarea indicilor de poziție crescând de la dreapta spre stânga, de la $-m$, la $n-1$, $-m$ fiind indicele cifrei cea mai îndepărtată spre dreapta de virgula separatoare între partea întreagă și partea fracționară, iar $n-1$ indicele cifrei cea mai îndepărtată spre stânga de virgula separatoare între partea întreagă și partea fracționară).

Întrucât în calculatoare se lucrează în baza 2, în continuare, vom considera $r=2$.

Fie:

$$\tilde{K}' = c_{n-1}c_{n-2}...c_1c_0. \quad (4.1.1.1-3)$$

semnul “.” nu se mai scrie, fiind subînțeles

$$\Rightarrow K' = c_{n-1} \cdot 2^{n-1} + c_{n-2} \cdot 2^{n-2} + \dots + c_1 \cdot 2^1 + c_0 \quad (4.1.1.1-4)$$

$$K' = 2\underbrace{(c_{n-1} \cdot 2^{n-2} + c_{n-2} \cdot 2^{n-3} + \dots + c_1)}_{K'_1} + c_0 \quad (4.1.1.1-5)$$

$$K'_1 = 2\underbrace{(c_{n-1} \cdot 2^{n-3} + c_{n-2} \cdot 2^{n-4} + \dots + c_2)}_{K'_2} + c_1 \quad (4.1.1.1-6)$$

.....

$$K'_{n-3} = 2\underbrace{(c_{n-1} \cdot 2 + c_{n-2})}_{K'_{n-2}} + c_{n-3} \quad (4.1.1.1-7)$$

$$K'_{n-2} = 2\underbrace{(c_{n-1})}_{K'_{n-1}} + c_{n-2} \quad (4.1.1.1-8)$$

$$K'_{n-1} = 2(0) + c_{n-1} \quad (4.1.1.1-9)$$

Aşadar, cifrele binare ale unui număr pozitiv întreg se pot obține prin împărțiri succesive la 2 ale numărului în cauză, respectiv ale căturilor, până se ajunge la cât nul, ele -cifrele- fiind resturi în aceste împărțiri.

Exemplu:

$$K' = 42 :$$

$$\begin{array}{rcl}
 & & C_0 \\
 42:2=21 \text{ rest } 0 & \Rightarrow & 42=2\cdot 21+0 \\
 21:2=10 \text{ rest } 1 & \Rightarrow & 21=2\cdot 10+1 \\
 10:2=5 \text{ rest } 0 & \Rightarrow & 10=2\cdot 5+0 \\
 5:2=2 \text{ rest } 1 & \Rightarrow & 5=2\cdot 2+1 \\
 2:2=1 \text{ rest } 0 & \Rightarrow & 2=2\cdot 1+0 \\
 1:2=0 \text{ rest } 1 & \Rightarrow & 1=2\cdot 0+1 \\
 & & C_5
 \end{array}$$

$$\Rightarrow 42_{10}=101010_2$$

Se precizează că numărul cifrelor binare necesare reprezentării unui număr “ a ” pozitiv întreg este:

$$n = \min_{i \in N} i \mid i > \log_2 a$$

Fie:

$$\tilde{K}'' = .c_{-1} \dots c_{-(m-1)} c_{-m} \quad (4.1.1.1-10)$$

$$K'' = c_{-1} \cdot 2^{-1} + c_{-2} \cdot 2^{-2} + \dots + c_{-(m-1)} \cdot 2^{-(m-1)} + c_{-m} \cdot 2^{-m} \quad (4.1.1.1-11)$$

Prin înmulțiri succesive cu 2 ale numărului inițial și ale părților fracționare ale produselor, rezultă:

$$K'' \cdot 2 = c_{-1} + \underbrace{c_{-2} \cdot 2^{-1} + \dots + c_{-(m-1)} \cdot 2^{-(m-2)}}_{K_1''} + c_{-m} \cdot 2^{-(m-1)} \quad (4.1.1.1-12)$$

$$K_1'' \cdot 2 = c_{-2} + \underbrace{c_{-3} \cdot 2^{-1} + \dots + c_{-(m-1)} \cdot 2^{-(m-3)}}_{K_2''} + c_{-m} \cdot 2^{-(m-2)} \quad (4.1.1.1-13)$$

.....

$$K_{m-2}'' \cdot 2 = c_{-(m-1)} + \underbrace{c_{-m} \cdot 2^{-1}}_{K_{m-1}''} \quad (4.1.1.1-14)$$

$$K_{m-1}'' \cdot 2 = c_{-m} \quad (4.1.1.1-15)$$

Așadar, cifrele binare ale unui număr pozitiv fracționar se pot obține prin înmulțiri succesive cu 2 ale numărului în cauză, respectiv ale părților fracționare ale produselor, ele –cifrele- fiind părțile întregi ale produselor; înmulțirile se efectuează până la determinarea unui număr satisfăcător de cifre.

Exemplu:

1). $K'' = 0.8125$

$$\begin{array}{r} & c_{-1} \\ & \diagup \\ 0.8125 \cdot 2 & = 1 + 0.625 \\ & \diagdown \\ 0.625 \cdot 2 & = 1 + 0.25 \\ & \diagdown \\ 0.25 \cdot 2 & = 0 + 0.5 \\ & \diagdown \\ 0.5 \cdot 2 & = 1 + 0.0 \\ & \diagdown \\ & c_{-4} \end{array}$$

$$\Rightarrow 0.8125_{10} = 0.1101_2$$

2). $K'' = 0.8124$

$$\begin{array}{r} & c_{-1} \\ & \diagup \\ 0.8124 \cdot 2 & = 1 + 0.6248 \\ & \diagdown \\ 0.6248 \cdot 2 & = 1 + 0.2496 \\ & \diagdown \\ 0.2496 \cdot 2 & = 0 + 0.4992 \\ & \diagdown \\ 0.4992 \cdot 2 & = 0 + 0.9984 \\ & \diagdown \\ 0.9984 \cdot 2 & = 1 + 0.9968 \\ & \diagdown \\ & c_{-5} \end{array}$$

$$\Rightarrow 0.8124_{10} = 0.11001\dots_2$$

Cifrele binare ale unui număr pozitiv format din parte întreagă și parte fracționară se determină aplicând regulile de mai sus pentru cele două părți.

Exemplu:

$$42.8125 = 42 + 0.8125 = 101010_2 + 0.1101_2 = 101010.1101_2$$

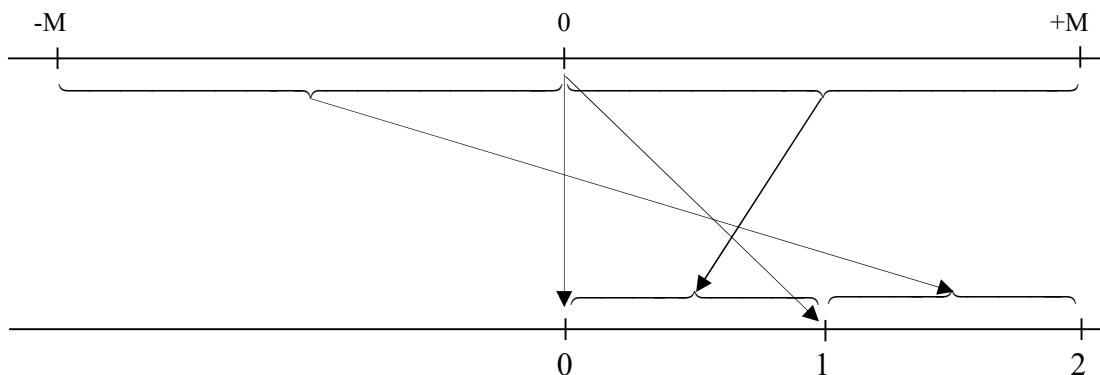
4.1.1.2. Reprezentarea numerelor de orice semn

4.1.1.2.1. Reprezentarea prin semn-mărime

În reprezentarea semn-mărime, numerele dintr-un interval oarecare, $(-M, +M)$ se reprezintă prin numere imagine din intervalul $[0, 2)$, după următoarea lege de corespondență:

$$sm(x) = \begin{cases} \frac{x}{M} \in (0;1), & \text{daca } x > 0 \\ 1 + \frac{|x|}{M} \in (1;2), & \text{daca } x < 0 \\ 0 \text{ sau } 1 \text{ daca } x = 0 \end{cases} \quad (4.1.1.2.1-1)$$

Evident, legea de mai sus ne spune că numerele pozitive se reprezintă prin numere din intervalul $(0; 1)$, iar numerele negative, prin numere din intervalul $(1; 2)$, în timp ce numărul *zero* are o dublă imagine, pe *zero*, respectiv pe *unu*:



Așadar, dacă notăm cu A un număr oarecare și a reprezentarea sa în semn-mărime, atunci, dacă:

$$\tilde{A} = \pm c_{n-1} \dots c_0 \cdot c_{-1} \dots c_{-m} \quad (4.1.1.2.1-2)$$

rezultă:

$$\tilde{a} = a_0.a_{-1}...a_{-(m+n)} \quad (4.1.1.2.1-3)$$

caracterul “.” nu se reprezintă în calculator prin nimic,
el fiind considerat în mod implicit

unde:

$$\bullet a_0 = \begin{cases} 0, & \text{daca } A \geq 0 \\ 1, & \text{daca } A < 0 \end{cases} \quad (4.1.1.2.1-4)$$

$$\bullet a_{-i} = c_{n-i}, \forall i = 1...m+n \quad (4.1.1.2.1-5)$$

Lungimea de cod a reprezentării semn-mărime este dată de relația:

$$lc_sm = \log_2 \frac{2M}{r} \quad (4.1.1.2.1-6)$$

unde:

r: rezoluția cu care se iau în considerare numerele de reprezentat în intervalul (-M; M).

Observație:

Atât M, cât și r, sunt, în practică, puteri ale lui 2.

Tabelele din figurile 4.1.1.2.1_1 și 4.1.1.2.1_2 exemplifică reprezentarea numerelor în semn mărime în ipotezele:

- $(-M; M) = (-2; 2)$
- $r = 0.25$

respectiv:

- $(-M; M) = (-8; 8)$
- $r = 1$

Numărul de reprezentat	-1.75	-1.50	-1.25	-1.00	-0.75	-0.50	-0.25	0	0.25	0.50	0.75	1.00	1.25	1.50	1.75
Codul semn-mărime	1111	1110	1101	1100	1011	1010	1001	0000 1000	0001	0010	0011	0100	0101	0110	0111
Valoarea codului semn-mărime	1.875	1.750	1.625	1.500	1.375	1.250	1.125	0.000 1.000	0.125	0.250	0.375	0.500	0.625	0.750	0.875

Fig. 4.1.1.2.1_1. Reprezentarea în semn-mărime a numerelor din intervalul (-2; 2), considerate cu rezoluția 0.25.

Numărul de reprezentat	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
Codul semn-mărime	1111	1110	1101	1100	1011	1010	1001	0000 1000	0001	0010	0011	0100	0101	0110	0111
Valoarea codului semn-mărime	1.875	1.750	1.625	1.500	1.375	1.250	1.125	0.000 1.000	0.125	0.250	0.375	0.500	0.625	0.750	0.875

Fig. 4.1.1.2.1_2. Reprezentarea în semn-mărime a numerelor din intervalul $(-8; 8)$, considerate cu rezoluția 1.

Evident, valoarea unui număr x al cărui cod în semn-mărime este $sm(x)$ va fi dată de expresiile:

$$x = \begin{cases} M \cdot sm(x), & \text{daca } sm(x) \in (0;1) \\ -M \cdot (sm(x)-1), & \text{daca } sm(x) \in (1;2) \\ 0, & \text{daca } sm(x) \in \{0;1\} \end{cases} \quad (4.1.1.2.1-7)$$

Avantajele reprezentării numerelor în semn mărime rezidă în faptul că ea permite efectuarea facilă a operațiilor de înmulțire și împărțire.

Dezavantajele reprezentării numerelor în semn mărime sunt:

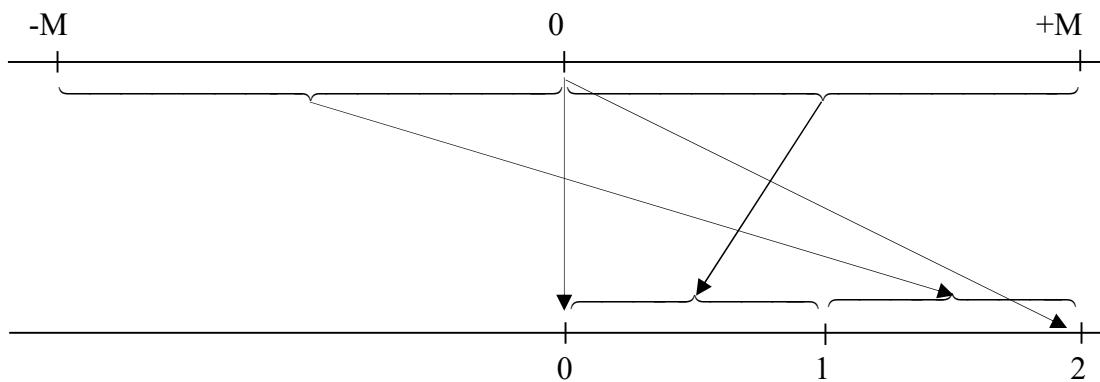
- necesită tratarea specifică a biților de semn în cazul operațiilor de adunare și scădere (la înmulțire și împărțire nici nu se poate altfel, în nici una dintre reprezentări);
- dedică două coduri pentru numărul 0: 0.0...0, respectiv 1.0...0, ceea ce face ca valorile reprezentabile cu ajutorul ei să fie cuprinse în intervalul $(-M; M)$ și nu $[-M; M]$, cum ar fi de dorit să se întâmple

4.1.1.2.2. Reprezentarea prin complement de unu

În reprezentarea prin complement de unu, numerele dintr-un interval oarecare, $(-M, +M)$ se reprezintă prin numere imagine din intervalul $[0, 2)$, după următoarea lege de corespondență:

$$c1(x) = \begin{cases} \frac{x}{M} \in (0;1), & \text{daca } x > 0 \\ 2 - 2^{-n} - \frac{|x|}{M} \in [1; 2 - 2^{-n}), & \text{daca } x < 0 \\ 0 \text{ sau } 2 - 2^{-n}, & \text{daca } x = 0 \end{cases} \quad (4.1.1.2.2-1)$$

Evident, legea de mai sus ne spune că, la fel ca în cazul reprezentării în semn-mărime, numerele pozitive se reprezintă prin numere din intervalul $(0; 1)$, iar numerele negative, prin numere din intervalul $(1; 2)$; numărul *zero* are și în acest caz o dublă imagine, de data aceasta, pe *zero*, respectiv pe $2 - 2^{-n}$:



Așadar, dacă notăm cu A un număr oarecare și a reprezentarea sa în complement de unu, atunci, dacă:

$$\tilde{A} = \pm c_{n-1} \dots c_0 . c_{-1} \dots c_{-m} \quad (4.1.1.2.2-2)$$

rezultă:

$$\tilde{a} = a_0 \cdot \underbrace{a_{-1} \dots a_{-(m+n)}}_{\text{caracterul ``.'' nu se reprezintă în calculator prin nimic, el fiind considerat în mod implicit}} \quad (4.1.1.2.2-3)$$

caracterul “.” nu se reprezintă în calculator prin nimic,
el fiind considerat în mod implicit

unde:

- $a_0 = \begin{cases} 0, & \text{daca } A \geq 0 \\ 1, & \text{daca } A < 0 \end{cases}$ $(4.1.1.2.2-4)$

- $a_{-i} = \begin{cases} c_{n-i}, & \text{daca } A \geq 0 \\ \bar{c}_{n-i}, & \text{daca } A < 0, \text{ unde } \bar{c}_{n-i} = NOT(c_{n-i}) \end{cases}, \forall i = 1 \dots m+n$ $(4.1.1.2.2-5)$

Așadar, numerele pozitive se reprezintă în complement de unu exact ca și prin semn-mărime. Numerele negative au bitul de semn ca în cazul reprezentării semn-mărime, iar ceilalți biți –valori complementare în raport cu acest caz.

Lungimea de cod a reprezentării complement de unu este dată de relația:

$$lc_cl = \log_2 \frac{2M}{r} \quad (4.1.1.2.2-6)$$

unde:

r: rezoluția cu care se iau în considerare numerele de reprezentat în intervalul $(-M; M)$.

Tabelele din figurile 4.1.1.2.2_1 și 4.1.1.2.2_2 exemplifică reprezentarea numerelor în complement de unu, în ipotezele:

- $(-M; M) = (-2; 2)$
- $r = 0.25$

respectiv:

- $(-M; M) = (-8; 8)$
- $r = 1$

Numărul de reprezentat	-1.75	-1.50	-1.25	-1.00	-0.75	-0.50	-0.25	0	0.25	0.50	0.75	1.00	1.25	1.50	1.75
Codul complement de unu	1000	1001	1010	1011	1100	1101	1110	0000 1111	0001	0010	0011	0100	0101	0110	0111
Valoarea codului complement de unu	1.000	1.125	1.250	1.375	1.500	1.625	1.750	0.000 1.875	0.125	0.250	0.375	0.500	0.625	0.750	0.875

Fig. 4.1.1.2.2_1. Reprezentarea în complement de unu a numerelor din intervalul $(-2; 2)$, considerate cu rezoluția 0.25.

Numărul de reprezentat	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
Codul complement de unu	1000	1001	1010	1011	1100	1101	1110	0000 1111	0001	0010	0011	0100	0101	0110	0111
Valoarea codului complement de unu	1.000	1.125	1.250	1.375	1.500	1.625	1.750	0.000 1.875	0.125	0.250	0.375	0.500	0.625	0.750	0.875

Fig. 4.1.1.2.2_2. Reprezentarea în complement de unu a numerelor din intervalul $(-8; 8)$, considerate cu rezoluția 1.

Evident, valoarea unui număr x al cărui cod în complement de unu este $c1(x)$ va fi dată de expresiile:

$$x = \begin{cases} M \cdot c1(x), & \text{daca } c1(x) \in (0;1) \\ -M \cdot (2 - 2^{-n} - c1(x)), & \text{daca } c1(x) \in (1; 2 - 2^{-n}) \\ 0, & \text{daca } c1(x) \in \{0; 2 - 2^{-n}\} \end{cases} \quad (4.1.1.2.2-7)$$

Se demonstrează că în reprezentarea complement de unu, suma a două numere se poate obține în doi pași, astfel:

- în primul pas, se adună între ei complementii de unu ai celor două numere, într-un proces în care bițiile de semn sunt tratați exact ca oricare alții
- în al doilea pas, se adună la suma obținută în pasul unu, în poziția cea mai puțin semnificativă, transportul rezultat în respectivul pas unu, transport ce, evident, poate fi fie 1, fie 0.

Exemplu:

Fie:

$$(-M; M) = (-8; 8), r=1$$

Rezultă:

$$lc_c1 = 4 \text{ și } n = 3$$

Fie:

$$1). a = -1, b = -5$$

Se va avea:

$$\begin{aligned} a &= -1_{10} = -001_2 = 1001_{sm} = 1110_{c1} \\ b &= -5_{10} = -101_2 = 1101_{sm} = 1010_{c1} \\ &\quad \begin{array}{r} 1110+ \\ 1010 \\ \hline 1\leftarrow 1000+ \\ \hline 1001 \end{array} \end{aligned}$$

cu:

$$1001_{c1} = 1110_{sm} = -110_2 = -6_{10}$$

2). $a = 1, b = 5$

Se va avea:

$$\begin{aligned} a &= 1_{10} = 001_2 = 0001_{sm} = 0001_{c1} \\ b &= 5_{10} = 101_2 = 0101_{sm} = 0101_{c1} \end{aligned}$$

$$\begin{array}{r} 0001+ \\ 0101 \\ \hline 0\leftarrow 0110+ \\ \hline 0110 \end{array}$$

cu:

$$0110_{c1} = 0110_{sm} = 110 = 6_{10}$$

Avantajele reprezentării numerelor în complement de unu sunt:

- permite efectuarea adunării prin introducerea în calcule a biților de semn în mod nedistinctiv față de restul biților
- permite înlocuirea scăderii prin adunarea la descăzut a complementului de unu al opusului scăzătorului; de reținut că: dacă $\tilde{x}|_{c1} = x_0.x_{-1}...x_{-p}$, atunci pentru $y = -x$, se va avea: $\tilde{y}|_{c1} = y_0.y_{-1}...y_{-p}$, cu $y_{-i} = \bar{x}_{-i}, \forall i = 0...p$; astăzi: dacă scăzătorul, în complement de unu, este 0011 (adică: +3), atunci numărul ce se va aduna la descăzut pentru realizarea scăderii va fi 1100 (adică: -3), iar dacă scăzătorul, în complement de unu, este 1100 (adică: -3), atunci numărul ce se va aduna la descăzut pentru realizarea scăderii va fi 0011 (adică: +3)

Dezavantajele reprezentării numerelor în complement de unu sunt:

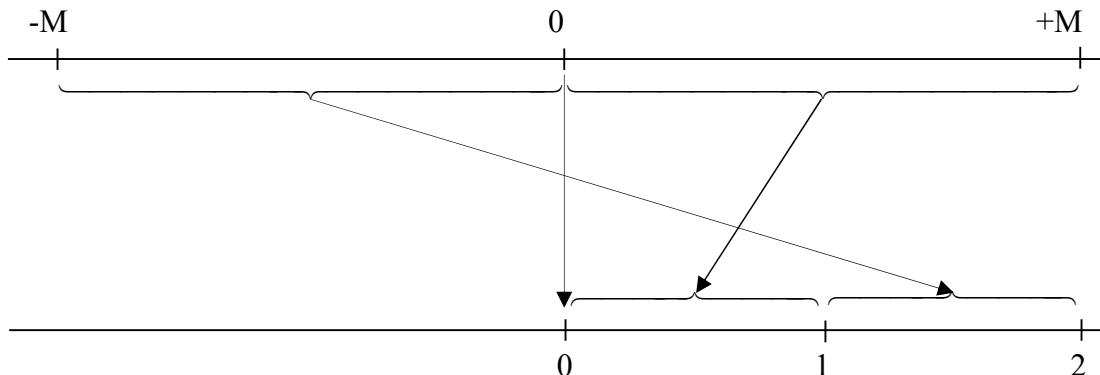
- dedică două coduri pentru numărul 0: 0.0...0, respectiv 1.1...1, ceea ce face ca valorile reprezentabile cu ajutorul ei să fie cuprinse în intervalul $(-M; M)$ și nu $[-M; M]$, cum se întâmplă în cazul altor reprezentări
- face dificilă execuția operațiilor de înmulțire și împărțire
- necesită parcurgerea a doi pași pentru efectuarea operației de adunare, ceea ce conduce la dublarea timpului de adunare, în raport cu alte reprezentări

4.1.1.2.3. Reprezentarea prin complement de doi

În reprezentarea prin complement de doi, numerele dintr-un interval oarecare, $(-M, +M)$ se reprezintă prin numere imagine din intervalul $[0, 2)$, după următoarea lege de corespondență:

$$c_2(x) = \begin{cases} \frac{x}{M} \in (0;1), & \text{daca } x > 0 \\ 2 - \frac{|x|}{M} \in [1;2), & \text{daca } x < 0 \\ 0, & \text{daca } x = 0 \end{cases} \quad (4.1.1.2.3-1)$$

Evident, legea de mai sus ne spune că, la fel ca în cazul reprezentării în semn-mărime și în complement de 1, numerele pozitive se reprezintă prin numere din intervalul $(0; 1)$. Numerele negative, însă, se reprezintă prin numere din intervalul $[1; 2)$ și nu $(1; 2)$, iar numărul *zero* are o singură imagine, pe *zero* însuși:



Așadar, dacă notăm cu A un număr oarecare și a reprezentarea sa în complement de unu, atunci, dacă:

$$\tilde{A} = \pm c_{n-1} \dots c_0 . c_{-1} \dots c_{-m} \quad (4.1.1.2.3-2)$$

rezultă:

$$\tilde{a} = a_0 \cdot a_{-1} \dots a_{-(m+n)} \quad (4.1.1.2.3-3)$$

← caracterul “.” nu se reprezintă în calculator prin nimic,
el fiind considerat în mod implicit

unde:

$$\bullet \quad a_0 = \begin{cases} 0, & \text{daca } A \geq 0 \\ 1, & \text{daca } A < 0 \end{cases} \quad (4.1.1.2.3-4)$$

$$\bullet \quad a_{-i} = \begin{cases} c_{n-i}, & \text{daca } A \geq 0 \text{ sau daca } \forall j > i \Rightarrow c_{n-j} = 0 \\ \bar{c}_{n-i}, & \text{daca } A < 0 \text{ si } \exists j > i | c_{n-j} \neq 0, \text{ unde } \bar{c}_{n-i} = NOT(c_{n-i}) \end{cases}, \forall i = 1..m+n \quad (4.1.1.2.3-5)$$

Așadar, numerele pozitive se reprezintă în complement de doi exact ca și în semn-mărime, respectiv ca și în complement de unu. Numerele negative au bitul de semn și biții situați în dreapta celui mai din dreapta bit nenul, precum și pe acesta însuși, la fel ca în cazul reprezentării semn-mărime, iar ceilalți biți –ne referim, evident, la biții de mărime–, valori complementare în raport cu acest caz. Bitul de semn va fi 1 și în reprezentarea în complement de 2, la fel ca și în semn-mărime și în complement de 1.

Lungimea de cod a reprezentării complement de doi este dată de relația:

$$lc_c2 = \log_2 \frac{2M}{r} \quad (4.1.1.2.3-6)$$

unde:

r: rezoluția cu care se iau în considerare numerele de reprezentat în intervalul [-M; M].

Evident, numărul cifrelor binare din dreapta bitului dedicat semnului este n :

$$n = lc_c2 - 1, \quad (4.1.1.2.3-7)$$

Tabelele din figurile 4.1.1.2.3_1 și 4.1.1.2.3_2 exemplifică reprezentarea numerelor în complement de doi, în ipotezele:

- $[-M; M] = [-2; 2]$
- $r = 0.25$

respectiv:

- $[-M; M] = [-8; 8]$
- $r = 1$

Numărul de reprezentat	-2.00	-1.75	-1.50	-1.25	-1.00	-0.75	-0.50	-0.25	0	0.25	0.50	0.75	1.00	1.25	1.50	1.75
Codul complement de doi	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
Valoarea codului complement de doi	1.000	1.125	1.250	1.375	1.500	1.625	1.750	1.875	0.000	0.125	0.250	0.375	0.500	0.625	0.750	0.875

Fig. 4.1.1.2.3_1. Reprezentarea în complement de doi a numerelor din intervalul [-2; 2), considerate cu rezoluția 0.25.

Numărul de reprezentat	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
Codul complement de doi	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
Valoarea codului complement de doi	1.000	1.125	1.250	1.375	1.500	1.625	1.750	1.875	0.000	0.125	0.250	0.375	0.500	0.625	0.750	0.875

Fig. 4.1.1.2.3_2. Reprezentarea în complement de doi a numerelor din intervalul [-8; 8), considerate cu rezoluția 1.

Evident, valoarea unui număr x al cărui cod în complement de doi este $c2(x)$ va fi dată de expresiile:

$$x = \begin{cases} M \cdot c2(x), & \text{daca } c2(x) \in (0;1) \\ -M \cdot (2 - c2(x)), & \text{daca } c2(x) \in [1;2) \\ 0, & \text{daca } c2(x) = 0 \end{cases} \quad (4.1.1.2.3-8)$$

Se demonstrează că în reprezentarea complement de doi, suma a două numere se poate obține într-un singur pas, adunând între ei complemenții de doi ai celor două numere, într-un proces în care bițiile de semn sunt tratați exact ca oricare alții.

Facem convenția de a nota cu litere minuscule numerele de reprezentat și cu litere majuscule imaginile lor în reprezentarea în complement de doi:

$$x \in [-M; M) \xleftarrow{c2} X \in [0; 2) \quad (4.1.1.2.3-9)$$

Fie $a \in [-M; M]$ și $b \in [-M; M]$ două numere cu proprietatea că:

$$a+b \in [-M; M] \quad (4.1.1.2.3-10)$$

și A și B imaginile lor prin $c2$.

Atunci:

$$C = \begin{cases} A + B, & \text{daca } A + B \in [0; 2) \\ A + B - 2, & \text{daca } A + B \in [2; 4) \end{cases} \quad (4.1.1.2.3-11)$$

are proprietatea că:

$$C \leftarrow \frac{c2}{c} \quad (4.1.1.2.3-12)$$

unde:

$$c = a + b \quad (4.1.1.2.3-13)$$

Într-adevăr:

i). dacă $a \geq 0$ și $b \geq 0$ atunci, evident, $a+b \geq 0$ și rezultă:

$$c2(a) + c2(b) = \frac{a}{M} + \frac{b}{M} = \frac{a+b}{M} = c2(a+b) \quad (4.1.1.2.3-14)$$

ii). dacă $a \geq 0$ și $b < 0$, cu $a \geq |b|$, atunci, evident, $a+b \geq 0$ și rezultă:

$$c2(a) + c2(b) = \frac{a}{M} + 2 - \frac{|b|}{M} = 2 + \frac{a - |b|}{M} \xrightarrow{\substack{\text{prin ignorarea} \\ \text{termenului "2"}}} \frac{a - |b|}{M} = c2(a+b) \quad (4.1.1.2.3-15)$$

iii). dacă $a \geq 0$ și $b < 0$, cu $a < |b|$, atunci, evident, $a+b < 0$ și rezultă:

$$c2(a) + c2(b) = \frac{a}{M} + 2 - \frac{|b|}{M} = 2 - \frac{|b|-a}{M} = c2(a+b) \quad (4.1.1.2.3-16)$$

iv). dacă $a < 0$ și $b \geq 0$, cu $b \geq |a|$, atunci, evident, $a+b \geq 0$ și rezultă:

$$c2(a) + c2(b) = 2 - \frac{|a|}{M} + \frac{b}{M} = 2 + \frac{b - |a|}{M} \xrightarrow{\substack{\text{prin ignorarea} \\ \text{termenului "2"}}} \frac{b - |a|}{M} = c2(a+b) \quad (4.1.1.2.3-17)$$

v). dacă $a < 0$ și $b \geq 0$, cu $|a| \geq b$, atunci, evident, $a+b < 0$ și rezultă:

$$c2(a) + c2(b) = 2 - \frac{|a|}{M} + \frac{b}{M} = 2 - \frac{|a|-b}{M} = c2(a+b) \quad (4.1.1.2.3-18)$$

vi). dacă $a < 0$ și $b < 0$, atunci, evident, $a+b < 0$ și rezultă:

$$c2(a) + c2(b) = 2 - \frac{|a|}{M} + 2 - \frac{|b|}{M} \xrightarrow{\text{prin ignorarea unui termen "2" }} 2 - \frac{|a|+|b|}{M} = c2(a+b) \quad (4.1.1.2.3-19)$$

Observație:

Ignorarea termenilor de valoare “2” înseamnă, în practică, neglijarea transportului generat în procesul de adunare din rangul 2^0 către rangul 2^1 .

Exemplu:

Fie:

$$[-M; M] = [-8; 8], r=1$$

Rezultă:

$$lc_c2 = 4 \text{ și } n = 3$$

Fie:

$$1). \ a = -1, b = -5$$

Se va avea:

$$a = -1_{10} = -001_2 = 1001_{sm} = 1111_{c2}$$

$$b = -5_{10} = -101_2 = 1101_{sm} = 1011_{c2}$$

$$\begin{array}{r} 1111+ \\ 1011 \\ \hline 1\leftarrow 1010 \\ \uparrow \text{se pierde} \end{array}$$

cu:

$$1010_{c2} = -6_{10}$$

$$2). \ a = 1, b = 5$$

Se va avea:

$$a = 1_{10} = 001_2 = 0001_{sm} = 0001_{c2}$$

$$b = 5_{10} = 101_2 = 0101_{sm} = 0101_{c2}$$

$$\begin{array}{r} 0001+ \\ 0101 \\ \hline 0\leftarrow 0110 \end{array}$$

cu:

$$0110_{c2} = 6_{10}$$

Avantajele reprezentării numerelor în complement de doi sunt:

- permite efectuarea adunării prin introducerea în calcule a biților de semn în mod nedistinctiv față de restul biților
- permite înlocuirea scăderii prin adunarea la descăzut a complementului de doi al opusului scăzătorului; de reținut că: dacă $\tilde{x}|_{c2} = x_0.x_{-1}...x_{-p}$, atunci pentru $y = -x$, se va avea:

$$\tilde{y}|_{c2} = y_0.y_{-1}...y_{-p}, \text{ cu } y_{-i} = \begin{cases} x_{-i}, & \text{dacă } \forall j > i \Rightarrow x_{-j} = 0 \\ \bar{x}_{-i}, & \text{dacă } \exists j > i | x_{-j} \neq 0 \end{cases}, \forall i = 0...p;$$

așadar: dacă scăzătorul, în complement de doi, este 0110 (adică: +6), atunci numărul ce se va aduna la descăzut pentru realizarea scăderii va fi 1010 (adică: -6, în complement de 2), iar dacă scăzătorul, în complement de doi, este 1010 (adică: -6), atunci numărul ce se va aduna la descăzut pentru realizarea scăderii va fi 0110 (adică: +6)

- numărul 0 are dedicat un singur cod: 0.0...0, ceea ce face ca valorile reprezentabile cu ajutorul ei să fie cuprinse în intervalul $[-M; M]$ și nu doar $(-M; M)$, cum se întâmplă în cazul altor reprezentări

Dezavantajele reprezentării numerelor în complement de doi sunt:

- face execuția operațiilor de înmulțire și împărțire mai dificilă decât în cazul reprezentării semn-mărime, dar, oricum, acceptabilă

4.1.2. Reprezentarea numerelor în virgulă flotantă

4.1.2.1. Principii

Reprezentarea numerelor în virgulă fixă nu dă satisfacție în privința rezoluției și preciziei, atunci când numerele variază în intervale largi. În astfel de cazuri, trebuie folosită aşa-numita reprezentare în virgulă flotantă. În această reprezentare, un număr R se reprezintă printr-o mantisă, MT , și un exponent, E , stabilite astfel încât:

$$R = MT \cdot B^E,$$

B fiind baza sistemului de reprezentare. În domeniul calculatoarelor, baza se alege 2, sau, în orice caz, o putere a lui 2. Se precizează că ea nu trebuie să apară explicit la nivel de circuite, ci poate fi și așa și este-subînțeleasă.

Există mai multe convenții de reprezentare a numerelor în virgulă flotantă. Cea mai răspândită și doar la prezentarea ei ne vom limita-

este cea introdusă de IEEE (Institute of Electronical and Electrical Engineers) în anul 1985, sub numele IEEE 754. Înainte de a prezenta reprezentarea IEEE 754, se impun câteva precizări privind exponentul, în general, în reprezentarea în virgulă flotantă.

Exponentul este un număr întreg. Asupra lui sunt instituite două exigențe. Pe de o parte, ar fi de dorit ca numărul *zero* să se reprezinte numai prin biți nuli, pentru ca el să fie ușor detectabil, ca și în reprezentarea în virgulă fixă. Pe de altă parte, se impune ca numerele nenule, dar foarte mici, să aibă același exponent ca numărul *zero*, pentru ca erorile introduse prin rotunjiri și trunchieri să nu fie mari. Ambele cerințe pot fi satisfăcute dacă se adoptă ca exponentul să apară translatat cu un deplasament (*bias*) pozitiv, stabilit astfel încât exprimarea sa prin biți exclusiv nuli să corespundă celei mai mici valori posibile (aceasta este o valoare negativă, evident).

Spre exemplificare, se prezintă, în tabelul din figura 4.1.2.1_1, cazul exponenților reprezentați pe 8 biți, cu bias de 127, respectiv de 128.

Exponentul translatat	Valoarea exponentului translatat	Valoarea exponentului efectiv	
		Bias=127	Bias=128
11111111	255	+128	+127
11111110	254	+127	+126
...
10000001	129	+2	+1
10000000	128	+1	0
01111111	127	0	-1
01111110	126	-1	-2
...
00000001	1	-126	-127
00000000	0	-127	-128

Fig. 4.1.2.1_1. Două exemple de translatare a exponenților.

Dacă atât mantisa, cât și exponentul se reprezintă pe 8 biți, cu biasul de +128, vom avea:

- pentru numărul *zero*:

$$\underbrace{0.0000000}_{\text{mantisa}} \quad \underbrace{00000000}_{\text{exponentul translatat}} \rightarrow 0$$

- pentru numărul cel mai apropiat de *zero*:

$$\underbrace{0.0000001}_{\text{mantisa}} \quad \underbrace{00000000}_{\text{exponentul translatat}} \rightarrow 2^{-7} \cdot 2^{-128} \cong 0$$

Dacă exponentul s-ar reprezenta netranslatat, atunci s-ar avea:

- pentru numărul *zero*:

$$\underbrace{0.0000000}_{\text{mantisa}} \quad \underbrace{00000000}_{\text{exponentul}} \rightarrow 0$$

- pentru numărul cel mai apropiat de *zero*:

$$\underbrace{0.0000001}_{\text{mantisa}} \quad \underbrace{00000000}_{\text{exponentul}} \rightarrow 2^{-7} \cdot 2^0 \gg 0 \text{ (a se citi: sensibil diferit de zero)}$$

Așadar, diferența între numărul care are toți biți nuli, în afară de bitul cel mai semnificativ al mantisei și numărul *zero* este 2^{-7} , când se lucrează cu exponentul netranslatat și 2^{-135} , când se lucrează cu exponentul translatat.

Revenind la standardul IEEE 754, facem mențiunea că el are, conform versiunii din 2019 –ultima-, variante de reprezentare pe 16, 32, 64, 128 și 256 biți, numite, respectiv, semiprecizie, simplă precizie, dublă precizie, cuadruplă precizie și octuplă precizie. Tabelul următor sintetizează parametrii esențiali ai fiecărei:

Lungimea reprezentării	Denumirea	Nr. biți pt. exponent	Nr. biți pt. mantisă (inclusiv semnul)	Biasul exponentului
16	semiprecizie	5	11	$2^4 - 1 = 15$
32	simplă precizie	8	24	$2^7 - 1 = 127$
46	dublă precizie	11	53	$2^{10} - 1 = 1023$
128	cuadruplă precizie	15	113	$2^{14} - 1 = 16\ 383$
256	octuplă precizie	19	237	$2^{18} - 1 = 262\ 143$

4.1.2.2. Standardul IEEE 754 pe 32 de biți

În standardul IEEE 754 pe 32 de biți, baza este 2 și ea nu se materializează, ci se subînțelege, exponentul este pe 8 biți, translatat cu un deplasament egal cu 127, iar mantisa este un număr real în intervalul [1; 2), reprezentat în semn-mărime, cu partea întreagă nematerializată, ci subînțeleasă. Așadar, în timp ce mantisa vizibilă este $\pm M$, mantisa efectivă este $MT = \pm 1.M$.

Câmpurile reprezentării IEEE 754 pe 32 de biți sunt următoarele:

3130	2322	0
S	E	M

unde:

- S este semnul mantisei: 1 pentru mantisă negativă, 0 pentru mantisă pozitivă
- E este exponentul pe 8 biți, translatat, cu bias=127
- M este excesul de 1 al mantisei (partea vizibilă a mantisei), pe 23 de biți

Valoarea unui număr real x reprezentat în standardul IEEE 754 pe 32 de biți este dată de expresia:

$$x = (-1)^S \cdot (1.M) \cdot 2^{E-127},$$

unde:

- $M \in [0; 1)$, cu rezolutia 2^{-23}
- $E \in [0; 254]$

Așadar, se va avea:

- cel mai mare număr pozitiv reprezentabil este:

0	1	8	9	31
0	1	1	...	1 1

și are valoarea:

$$cmMnpr = (-1)^0 \cdot (1.11\dots11) \cdot 2^{254-127} = +(1 + (1 - 2^{-23})) \cdot 2^{127} \cong +3.4 \cdot 10^{+38}$$

- cel mai mic număr negativ reprezentabil este:

0	1	8	9	31
1	1	1	...	1 1

și are valoarea:

$$cmmnnr = (-1)^1 \cdot (1.11\dots11) \cdot 2^{254-127} = -(1 + (1 - 2^{-23})) \cdot 2^{127} \cong -3.4 \cdot 10^{+38}$$

- cel mai mic număr pozitiv reprezentabil este:

0	1	8	9		31
0	0	...	0	1	0

și are valoarea:

$$cmmnpr = (-1)^0 \cdot (1.00...00) \cdot 2^{1-127} = +2^{-126} \cong +1.18 \cdot 10^{-38}$$

- cel mai mare număr negativ reprezentabil este:

0	1	8	9		31
1	0	...	0	1	0

și are valoarea:

$$cmMnnr = (-1)^1 \cdot (1.00...00) \cdot 2^{1-127} = -2^{-126} \cong -1.18 \cdot 10^{-38}$$

- numărul *zero* este:

0	1	8	9		31
0	0	...	0	0	0

și are valoarea:

$$zero = (-1)^0 \cdot (1.00...00) \cdot 2^{0-127} = +2^{-127} \cong +0.59 \cdot 10^{-38}$$

Standardul IEEE 754 rezervă unele combinații de biți pentru semnalarea unor situații deosebite. Astfel:

- combinația de biți corespunzătoare lui E=255 și M≠0 arată că nu se are de a face cu un număr, în cazuri cum sunt: $\frac{0}{0}, \sqrt{0}$ cantitate negativă
- combinația de biți corespunzătoare lui E=255 și M=0 arată "supradepășirea" ("overflow"), atunci când se ajunge la numere $x \in (-\infty; -3.4 \cdot 10^{38}) \cup (+3.4 \cdot 10^{38}; +\infty)$
- combinația de biți corespunzătoare lui E=0 și M≠0 arată "subdepășirea" ("underflow"), atunci când se ajunge la numere $x \in (-1.18 \cdot 10^{-38}; 0) \cup (0; +1.18 \cdot 10^{-38})$

4.1.2.3. Standardul IEEE 754 pe 64 de biți

În standardul IEEE 754 pe 64 de biți, baza este, de asemenea, 2 și ea nu se materializează, ci se subînțelege, exponentul este pe 11 biți, translatat cu un deplasament egal cu 1023, iar mantisa este un număr real în intervalul [1; 2), reprezentat în semn-mărime, cu partea întreagă

nematerializată, ci subînteleasă. Așadar, în timp ce mantisa vizibilă este $\pm M$, mantisa efectivă este $MT = \pm 1.M$.

Câmpurile reprezentării IEEE 754 pe 64 de biți sunt următoarele:

6362	5251	0
S	E	M

unde:

- S este semnul mantisei: 1 pentru mantisă negativă, 0 pentru mantisă pozitivă
- E este exponentul pe 11 biți, translatat, cu bias=1023
- M este excesul de 1 al mantisei (partea vizibilă a mantisei), pe 52 de biți

Valoarea unui număr real x reprezentat în standardul IEEE 754 pe 64 de biți este dată de expresia:

$$x = (-1)^S \cdot (1.M) \cdot 2^{E-1023},$$

unde:

- $M \in [0;1)$, cu rezolutia 2^{-52}
- $E \in [0; 2046]$

4.2. Dispozitive de adunare și scădere

4.2.1. Principii

Operația de adunare consistă în determinarea, pentru fiecare rang al reprezentării, a cifrei sumei și a cifrei transportului spre rangul de pondere imediat superioară, pornind de la cifrele celor doi operanzi corespunzătoare rangului în cauză și de la cifra transportului din rangul de pondere imediat inferioară, aşa cum se arată în figura 4.2.1_1. Evident, operanzzii sunt presupuși reprezentați în complement de 2. Pe cale de consecință, ponderile biților sunt, de la stânga la dreapta, $2^0, 2^{-1}, \dots, 2^{-(n-2)}, 2^{-(n-1)}$.

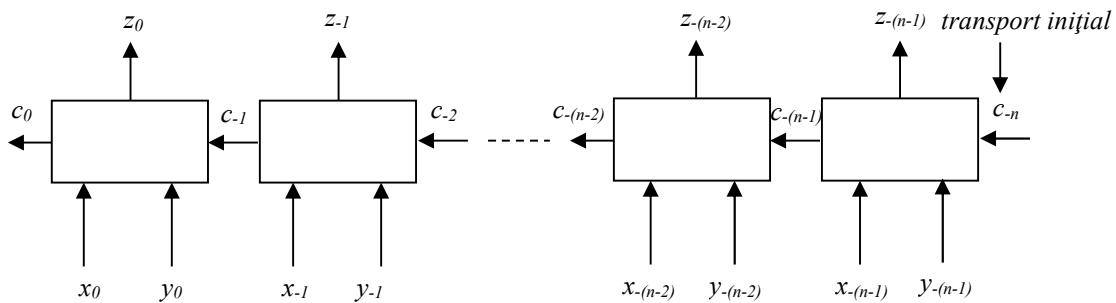


Fig. 4.2.1_1. O ilustrare a operației de adunare.

Așadar, pentru un rang "i", se vor avea situațiile din figura 4.2.1_2 și circuistica din figura 4.2.1_3.

Rezultatele obținute în procesul de adunare pot fi afectate de erori, atunci când ele se situează în afara intervalului de reprezentabilitate corespunzător numărului de biți pe care se lucrează. Se spune, în astfel de situații, că intervine fenomenul de "depășire". Evident, fenomenul de depășire poate apărea doar atunci când se adună operanzi cu același semn și se manifestă prin aceea că semnul rezultatului diferă de semnul comun al celor doi operanzi. Tabelul din figura 4.2.1.4 evidențiază clar cazurile de depășire, precum și modul în care ele pot fi detectate.

4.2. Dispozitive de adunare și scădere

4.2.1. Principii

Operația de adunare consistă în determinarea, pentru fiecare rang al reprezentării, a cifrei sumei și a cifrei transportului spre rangul de pondere imediat superioară, pornind de la cifrele celor doi operanzi corespunzătoare rangului în cauză și de la cifra transportului din rangul de pondere imediat inferioară, aşa cum se arată în figura 4.2.1_1. Evident, operanzi sunt presupuși reprezentați în complement de 2. Pe cale de consecință, ponderile biților sunt, de la stânga la dreapta, $2^0, 2^{-1}, \dots, 2^{-(n-2)}, 2^{-(n-1)}$.

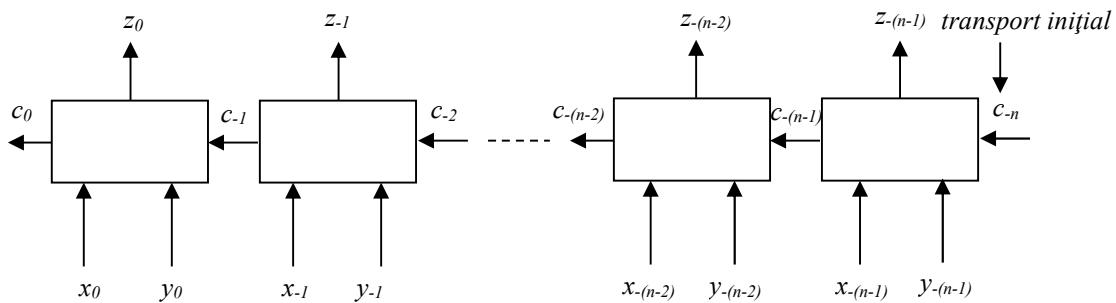


Fig. 4.2.1_1. O ilustrare a operației de adunare.

Așadar, pentru un rang "i", se vor avea situațiile din figura 4.2.1_2 și circuistica din figura 4.2.1_3.

Rezultatele obținute în procesul de adunare pot fi afectate de erori, atunci când ele se situează în afara intervalului de reprezentabilitate corespunzător numărului de biți pe care se lucrează. Se spune, în astfel de situații, că intervine fenomenul de "depășire". Evident, fenomenul de depășire poate apărea doar atunci când se adună operanzi cu același semn și se manifestă prin aceea că semnul rezultatului diferă de semnul comun al celor doi operanzi. Tabelul din figura 4.2.1.4 evidențiază clar cazurile de depășire, precum și modul în care ele pot fi detectate.

Intrări			Ieșiri	
$c_{-(i+1)}$	x_{-i}	y_{-i}	c_{-i}	z_{-i}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig. 4.2.1_2. Relațiile dintre $c_{-(i+1)}$, x_{-i} , y_{-i} , pe de o parte, și c_{-i} , z_{-i} , pe de altă parte.

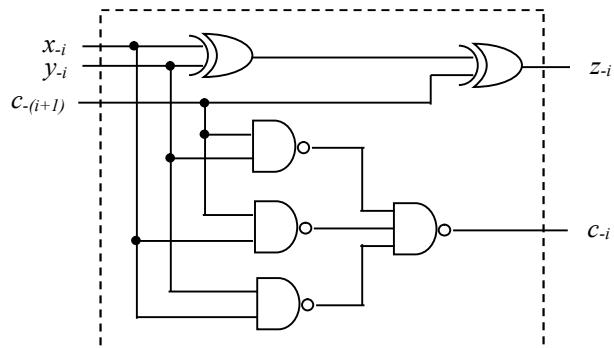


Fig. 4.2.1_3. Circuistica de determinare a biților z_{-i} și c_{-i} .

Intrări			Ieșiri		
c_{-1}	x_0	y_0	c_0	z_0	v
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	1	0

Fig. 4.2.1_4. Cu privire la fenomenul de "depășire".

Se demonstrează simplu că:

$$\begin{aligned} z_{-i} &= x_{-i} \oplus y_{-i} \oplus c_{-(i+1)} \\ c_{-i} &= x_{-i}y_{-i} + c_{-(i+1)}x_{-i} + c_{-(i+1)}y_{-i}, \end{aligned}$$

Circuistica de determinare, pe baza acestor ecuații, a biților z_{-i} și c_{-i} are schema din figura 4.2.1_3.

Rezultă, evident:

$$v = c_0 \oplus c_{-1}$$

4.2.2. Dispozitive de adunare seriale

4.2.2.1. Principii

Dispozitivele de adunare seriale efectuează adunarea, într-un ciclu de tact, pentru un singur rang. Se presupune, deci, că biții operanzilor sunt furnizați serial, în sincronism cu un semnal de tact, aşa cum se sugerează în figura 4.2.2.1_1.

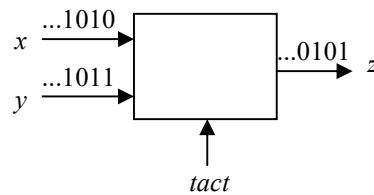


Fig. 4.2.2.1_1. Principiile adunării seriale.

4.2.2.2. Proiectarea unui dispozitiv de adunare serial folosind bistabile J-K

Din paragraful anterior se poate înțelege că adunarea serială presupune memorarea, în fiecare pas, a transportului rezultat în procesul de adunare, pentru a se face posibilă disponibilitatea lui în pasul de adunare următor. Având în vedere ecuația transportului, rezultă următoarea "diagramă a stărilor următoare":

Q_n^{xy}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Fig. 4.2.2.2_1. Diagrama stărilor următoare.

Tinând seamă de ecuația bistabilului $J-K$:

$$Q_{n+1} = J \cdot \bar{Q}_n + \bar{K} \cdot Q_n$$

se obțin, pentru J și K , următoarele diagrame, respectiv ecuații:

- pentru J :

Q_n^{xy}	00	01	11	10
0	0	0	1	0
1	*	*	*	*

$$\Rightarrow J = x \cdot y$$

Fig. 4.2.2.2_2. Diagrama intrării J .

- pentru K :

		xy	00	01	11	10	
		0	*	*	*	*	$\Rightarrow K = \bar{x} \cdot \bar{y} = \overline{x + y}$
		1	1	0	0	0	
Q							

Fig. 4.2.2.2_3. Diagrama intrării K .

Având în vedere ecuația sumei, rezultă următoarea "diagramă a ieșirii":

		xy	00	01	11	10	
		0	0	1	0	1	$\Rightarrow z = x \oplus y \oplus Q$
		1	1	0	1	0	
Q							

Fig. 4.2.2.2_4. Diagrama ieșirii.

În consecință, schema dispozitivului de adunare serial cu bistabil $J-K$ este următoarea:

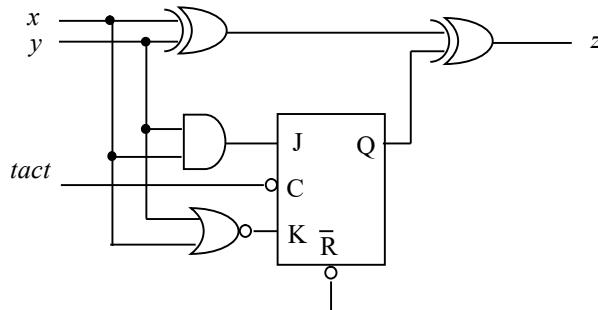


Fig. 4.2.2.2_5. Dispozitivul de adunare serial cu bistabil $J-K$.

4.2.2.3. Proiectarea unui dispozitiv de adunare serial folosind bistabile D

S-a arătat că:

$$\begin{aligned} z_{-i} &= x_{-i} \oplus y_{-i} \oplus c_{-(i+1)} \\ c_{-i} &= x_{-i} y_{-i} + c_{-(i+1)} x_{-i} + c_{-(i+1)} y_{-i} \end{aligned}$$

Din aceste relații, rezultă că schema unui dispozitiv de adunare serial cu bistabil D este cea din figura 4.2.2.3_1, în care dreptunghiul marcat cu semnul "+" reprezintă circuistica de determinare a biților sumei și transportului, a cărei schemă este redată în figura 4.2.1_3.

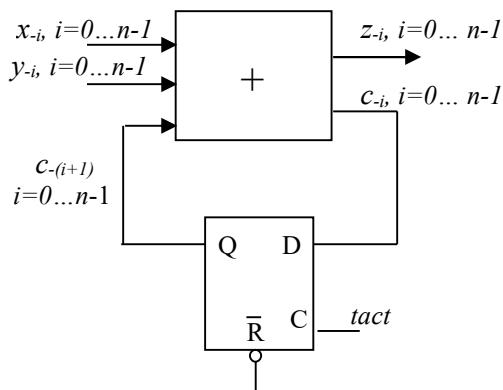


Fig. 4.2.2.3_1. Dispozitivul de adunare serial cu bistabil D .

4.2.2.4. Avantaje și dezavantaje ale dispozitivelor de adunare seriale

Avantajele dispozitivelor de adunare seriale sunt:

- simplitate
- implementare printr-un număr redus de circuite

Dezavantajul dispozitivelor de adunare seriale este:

- viteza de însumare redusă (pentru operanzi pe " n " biți, sunt necesare " n " perioade de tact; în plus, frecvența semnalului de tact trebuie aleasă astfel încât, în fiecare perioadă a sa, să existe suficient timp pentru stabilizarea lui c_{-i})

4.2.3. Dispozitive de adunare paralele

4.2.3.1. Principii

Dispozitivele de adunare paralele efectuează adunarea într-un ciclu de tact, pentru toate rangurile.

4.2.3.2. Dispozitive de adunare "ripple carry"

Se presupune că numerele sunt reprezentate în complement de 2, cu bitul corespunzător semnului în poziția c_0 . Schema de principiu a însumării paralele "ripple carry" se prezintă, atunci, aşa cum se arată în figura 4.2.3.2_1.

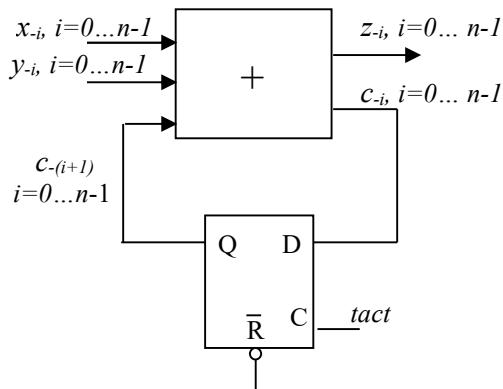


Fig. 4.2.2.3_1. Dispozitivul de adunare serial cu bistabil D .

4.2.2.4. Avantaje și dezavantaje ale dispozitivelor de adunare seriale

Avantajele dispozitivelor de adunare seriale sunt:

- simplitate
- implementare printr-un număr redus de circuite

Dezavantajul dispozitivelor de adunare seriale este:

- viteza de însumare redusă (pentru operanzi pe "n" biți, sunt necesare "n" perioade de tact; în plus, frecvența semnalului de tact trebuie aleasă astfel încât, în fiecare perioadă a sa, să existe suficient timp pentru stabilizarea lui c_{-i})

4.2.3. Dispozitive de adunare paralele

4.2.3.1. Principii

Dispozitivele de adunare paralele efectuează adunarea într-un ciclu de tact, pentru toate rangurile.

4.2.3.2. Dispozitive de adunare "ripple carry"

Se presupune că numerele sunt reprezentate în complement de 2, cu bitul corespunzător semnului în poziția c_0 . Schema de principiu a însumării paralele "ripple carry" se prezintă, atunci, aşa cum se arată în figura 4.2.3.2_1.

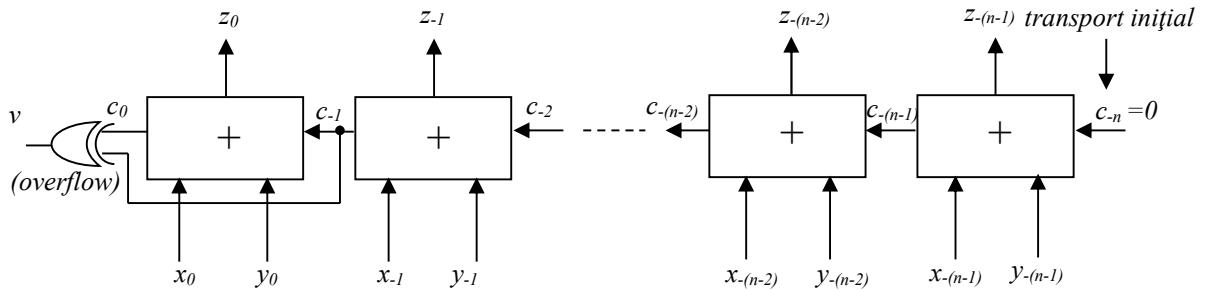


Fig. 4.2.3.2_1. Schema de principiu a însumării paralele "ripple carry".

Având în vedere schema circuistică de determinare a biților z_{-i} și c_{-i} - vezi figura 4.2.1_3-, rezultă că timpul de adunare, T_{ad} , al dispozitivului de adunare "ripple carry" este dat de relația:

$$T_{ad} = n \cdot (2\tau) + \tau = (2n + 1)\tau$$

Un dispozitiv de adunare "ripple carry" poate fi ușor transformat în dispozitiv de adunare / scădere, aducându-l la următoarea schemă:

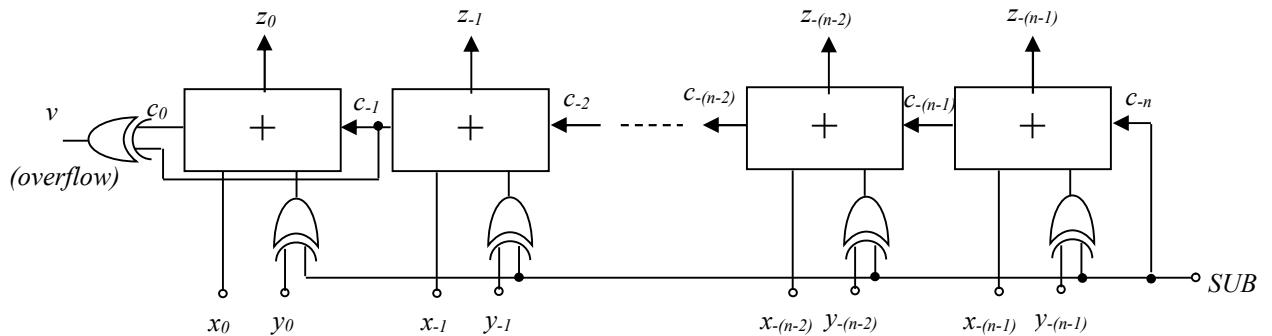


Fig. 4.2.3.2_2. Schema dispozitivului de adunare / scădere "ripple carry".

Evident:

- când $SUB=0$, vom avea:
 - la canalul 1 de intrare al dispozitivului: $x|_{c2}$
 - la canalul 2 de intrare al dispozitivului: $y|_{c2}$ (întrucât: $y_{-i} \oplus 0 = y_{-i}$), ceea ce face ca dispozitivul să efectueze operația $z|_{c2} = x|_{c2} + y|_{c2}$
- când $SUB=1$, vom avea:
 - la canalul 1 de intrare al dispozitivului: $x|_{c2}$
 - la canalul 2 de intrare al dispozitivului: $-y|_{c2}$ (întrucât: $y_{-i} \oplus 1 = \bar{y}_{-i}$ și $c_{-n}=1$), ceea ce face ca dispozitivul să efectueze operația $z|_{c2} = x|_{c2} + (-y|_{c2}) = x|_{c2} - y|_{c2}$

Observatie:

Se reamintește faptul că între complementul de 2 și complementul de 1 ai unui număr, în exprimare pe n biți, există relația: $c2(x) = cl(x) + 2^{-(n-1)}$

4.2.3.3. Dispozitive de adunare "carry lookahead"

Dispozitivele de adunare "carry lookahead" au apărut din ideea de a se reduce T_{ad} prin determinarea transporturilor c_{-i} direct în funcție de x_{-i} , y_{-i} , $x_{-(i+1)}$, $y_{-(i+1)}$, ..., $x_{-(n-1)}$, $y_{-(n-1)}$ și, eventual, c_{-n} , printr-o circuistică cu un număr redus de niveluri logice.

Se are:

$$c_{-i} = x_{-i}y_{-i} + c_{-(i+1)}x_{-i} + c_{-(i+1)}y_{-i} = \underbrace{g_{-i}}_{\text{g}_i} + \underbrace{(x_{-i} + y_{-i})c_{-(i+1)}}_{p_{-i}} \quad (4.2.3.3-1)$$

unde:

- g_{-i} este componenta lui c_{-i} generată în rangul “-i”
- p_{-i} este variabila care asigură regăsirea în c_{-i} a transportului propagat din rangul precedent, $c_{-(i+1)}$, atunci când acest transport este nenul

Mărimile g_{-i} și p_{-i} se pot sintetiza cu o circuistică de factura celei din figura 4.2.3.2_1.

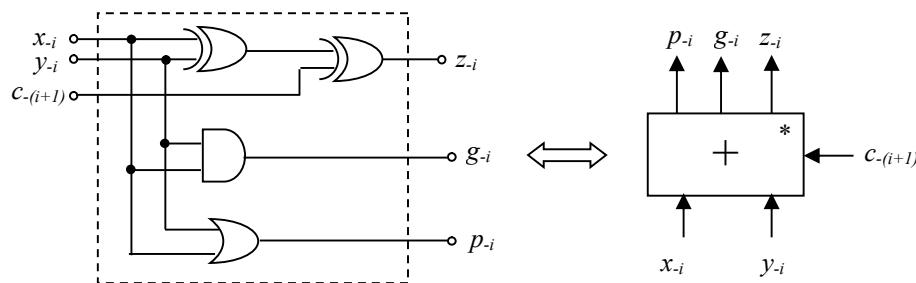


Fig. 4.2.3.3_1. Circuistica de sintetizare a lui z_{-i} , g_{-i} și p_{-i} .

Pe baza relației (4.2.3.3-1), se obține:

$$c_{-(i-1)} = x_{-(i-1)}y_{-(i-1)} + (x_{-(i-1)} + y_{-(i-1)})c_{-i} = g_{-(i-1)} + p_{-(i-1)}c_{-i} \quad (4.2.3.3-2)$$

iar din (4.2.3.3-2) și (4.2.3.2-1) rezultă:

$$c_{-(i-1)} = g_{-(i-1)} + p_{-(i-1)}g_{-i} + p_{-(i-1)}p_{-i}c_{-(i+1)} \quad (4.2.3.3-3)$$

Pentru concretizare, să considerăm cazul operării pe 4 biți. Vom avea:

$$c_{-3}=g_{-3}+p_{-3}c_{in} \quad (4.2.3.2-4)$$

$$c_{-2}=g_{-2}+p_{-2}g_{-3}+p_{-2}p_{-3}c_{in} \quad (4.2.3.2-5)$$

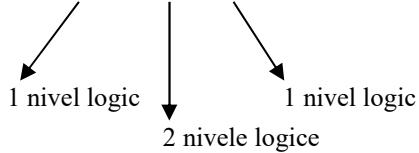
$$c_{-1}=g_{-1}+p_{-1}g_{-2}+p_{-1}p_{-2}c_{-3}=g_{-1}+p_{-1}g_{-2}+p_{-1}p_{-2}g_{-3}+p_{-1}p_{-2}p_{-3}c_{in} \quad (4.2.3.2-6)$$

$$c_0=g_0+p_0g_{-1}+p_0p_{-1}c_{-2}=g_0+p_0g_{-1}+p_0p_{-1}g_{-2}+p_0p_{-1}p_{-2}g_{-3}+p_0p_{-1}p_{-2}p_{-3}c_{in} \quad (4.2.3.2-7)$$

și, pe această bază, vom obține schema din figura 4.2.3.3_2.

Evident, circuistica devine considerabil mai complexă și mai costisitoare, pe măsură ce numărul de ranguri crește. Timpul de adunare este, însă, redus și nu depinde de numărul rangurilor.

$$T_{ad} = \tau_{p,g} + \tau_c + \tau_z = \tau + 2\tau + \tau = 4\tau \quad (4.2.3.2-8)$$



Observație:

S-a avut în vedere că $x_i \oplus z_i$ se determină în paralel cu p și g , nu consecutiv lor!

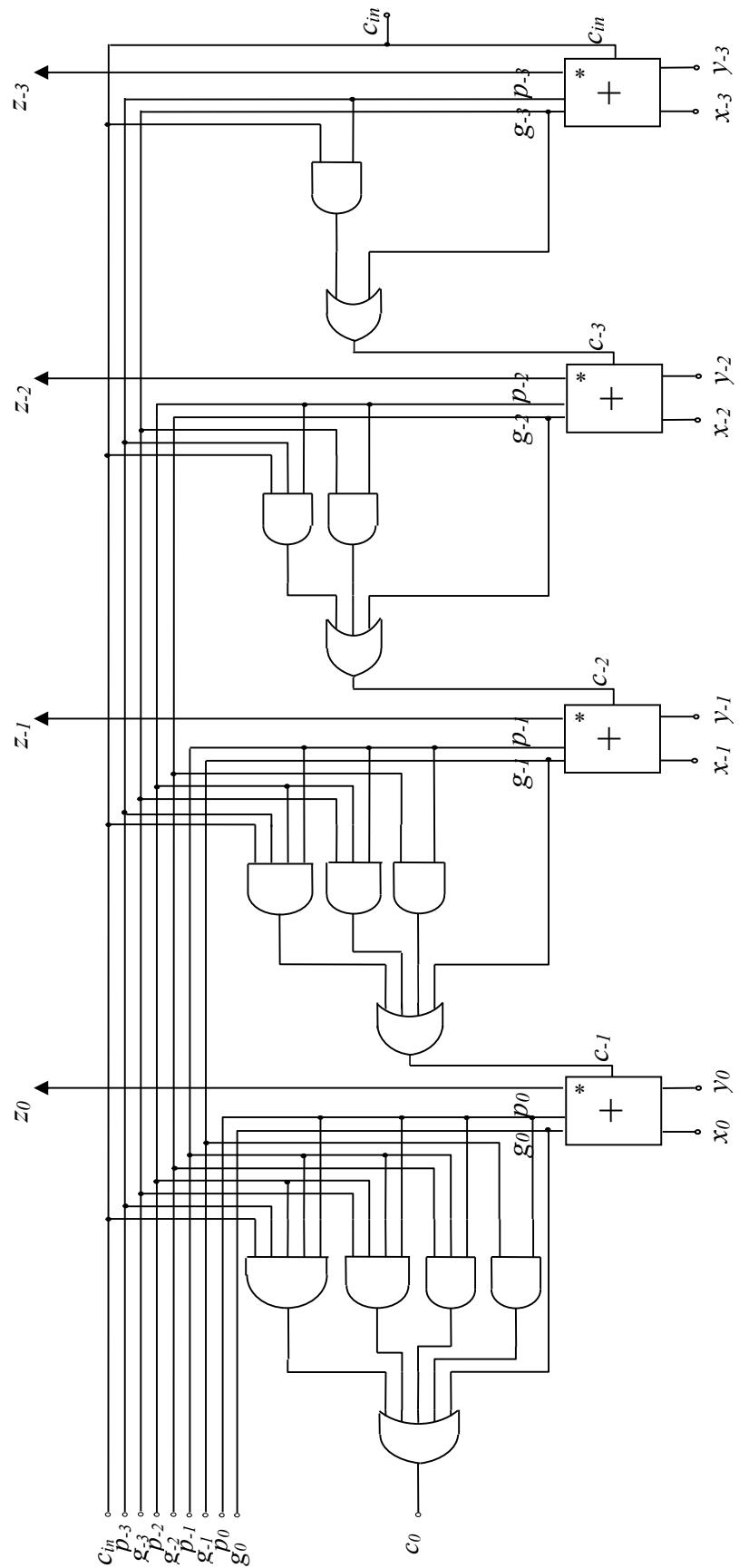


Fig. 4.2.3.3_2. Schema dispozitivului de însumare "carry lookahead".

4.3. Dispozitive de înmulțire

4.3.1. Principii

Reprezentările în care se efectuează în calculator operația de înmulțire sunt reprezentarea semn-mărime și reprezentarea complement de 2. Pentru ilustrarea principiilor de interes pentru implementarea acestei operații, alegem să folosim, în cele ce urmează, reprezentarea semn-mărime.

Fie de înmulțit două numere y și x din intervalul $(-M; M)$, considerat cu rezoluția r . Lungimea de cod a reprezentării semn mărime a acestor numere va fi –vezi relația (4.1.1.2.1-6)-:

$$lc_sm_o = \log_2(2M/r) = \log_2 M + 1 - \log_2 r \quad (4.3.1-1)$$

Întrucât, evident, intervalul de valori posibile pentru produsul celor două numere este $(-M^2; M^2)$, dacă adoptăm și pentru produs rezoluția r , cum este și firesc, rezultă că lungimea de cod necesară pentru reprezentarea produsului va fi:

$$lc_sm_p = \log_2(2M^2/r) = 2\log_2 M + 1 - \log_2 r = 2lc_sm_o - 1 + \log_2 r \quad (4.3.1-2)$$

Dacă impunem, în plus, ca cele două numere să fie pozitive, atunci, conform relației (4.1.1.2.1-5), imaginile lor semn-mărime vor fi:

$$sm(x) = x/M \quad (4.3.1-3)$$

$$sm(y) = y/M \quad (4.3.1-4)$$

iar produsul acestor imagini va fi:

$$p = (xy)/M^2 = sm(xy) \quad (4.3.1-5)$$

Privind acest p prin prisma relației (4.1.1.2.1-7), în ipotezele stabilite, se observă că el este tocmai imaginea semn-mărime a produsului z al numerelor x și y . Într-adevăr:

$$z = M^2 p = M^2((xy)/M^2) = xy \quad (4.3.1-6)$$

În cazul în care cel puțin unul dintre numerele care se înmulțesc este negativ, atunci, evident, cele de mai sus **nu mai sunt valabile**.

Fie, pentru operanzi, $(-M_{op}; M_{op}) = (-8; 8)$, $r=1$ și, prin consecință, $lc_sm_op = 4$ biți (inclusiv semnul) – a se revedea tabelul 4.1.1.2.1_2-, iar pentru produse, $(-M_p; M_p) = (-64; 64)$, $r=1$ și, prin consecință, $lc_sm_p = 7$ biți (inclusiv semnul), și fie, de asemenea, $x=6$ și $y=5$.

Constatăm că dacă înmulțim imaginile celor două numere –adică: numerele 0.750 și 0.625-, obținem numărul 0.46875, care, în condițiile stabilite, este imaginea semn-mărime a numărului $30 = 6 * 5$ (într-adevăr: $64 * 0.46875 = 30$).

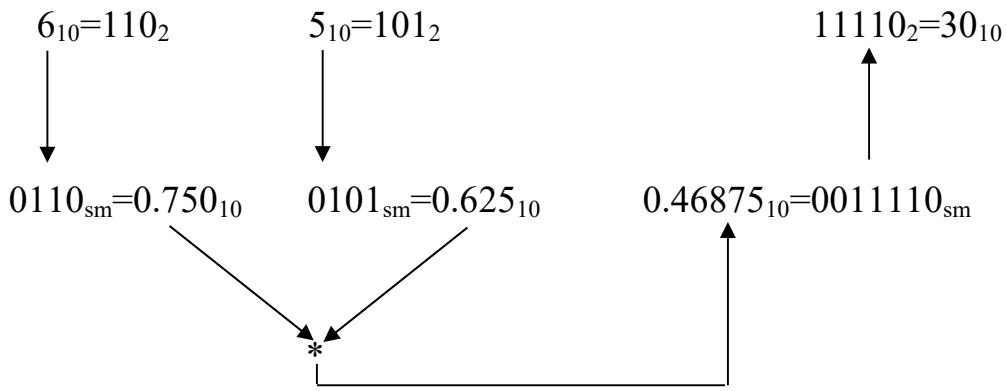


Fig. 4.3.1_1. O ilustrare a înmulțirii a două numere întregi prin intermediul imaginilor lor semn-mărime.

Algoritmul de înmulțire a celor doi operanzi întregi, de înmulțitul $y=110_2$ și înmulțitorul $x=101_2$ este, în principiu, cel cunoscut din aritmetică pentru înmulțirea numerelor zecimale, astfel încât vom avea:

$$\begin{array}{r}
 y \rightarrow 110 \cdot \leftarrow y_2 y_1 y_0 \\
 x \rightarrow \underline{101} \leftarrow x_2 x_1 x_0 \\
 \quad 110 \leftarrow 2^0 x_0 \cdot y \\
 \quad 000 \leftarrow 2^1 x_1 \cdot y \\
 \underline{\quad 110} \leftarrow 2^2 x_2 \cdot y \\
 11110 \leftarrow p = \sum_{j=0}^{n-1} 2^j x_j \cdot y, \text{ cu } n=3
 \end{array}$$

Evident, din punct de vedere al implementării, ar fi total inaceptabil să se procedeze ca mai sus, adică: într-o etapă să se calculeze, succesiv, produsele de forma $2^j x_j \cdot y$, iar în final, să se efectueze adunarea lor în vederea obținerii produsului căutat. O soluție mai rezonabilă ar rezulta dacă

lucrurile ar fi abordate în mod recurrent. Produsul p ar putea fi, atunci, exprimat astfel:

$$p = (((((0 + 2^0 x_0 \cdot y) + 2^1 x_1 \cdot y) + \dots + 2^{n-2} x_{n-2} \cdot y) + 2^{n-1} x_{n-1} \cdot y) \dots) p \quad (4.3.1-7)$$

$\boxed{\begin{array}{c} \square P_0 \\ \square P_1 \\ \square P_2 \\ \vdots \\ \square P_{n-1} \end{array}} \quad p$

iar calculele ar decurge după cum urmează:

y	\longrightarrow	110 .
x	\longrightarrow	101
$P_0 = 0$	\longrightarrow	00000 +
$x_0 y \cdot 2^0$	\longrightarrow	110
$P_1 = P_0 + x_0 y$	\longrightarrow	00110 +
$x_1 y \cdot 2^1$	\longrightarrow	000
$P_2 = P_1 + x_1 y$	\longrightarrow	00110 +
$x_2 y \cdot 2^2$	\longrightarrow	110
$p = P_2 + x_2 y$	\longrightarrow	11110

Fie $y'_0 y'_{-1} y'_{-2} y'_{-3}$ imaginea semn-mărime a lui $y = +y_2 y_1 y_0$ și $x'_0 x'_{-1} x'_{-2} x'_{-3}$ imaginea semn-mărime a lui $x = +x_2 x_1 x_0$. Rezultă:

$$y' = 2^0 \cdot 0 + 2^{-1} y'_{-1} + 2^{-2} y'_{-2} + 2^{-3} y'_{-3} \quad (4.3.1-8)$$

$$x' = 2^0 \cdot 0 + 2^{-1} x'_{-1} + 2^{-2} x'_{-2} + 2^{-3} x'_{-3} \quad (4.3.1-9)$$

Să vedem ce se întâmplă dacă, în loc să înmulțim cele două numere, înmulțim pe x' cu y' și, în plus, facem artificiul ca pe parcursul calculelor să operăm nu cu y' , ci cu $2y'$, luând, totodată, măsuri pentru ca rezultatul final să nu fie afectat.

$$\begin{aligned} p' &= y' \cdot x' = y' \cdot (2^0 \cdot 0 + 2^{-1} x'_{-1} + 2^{-2} x'_{-2} + 2^{-3} x'_{-3}) = \\ &= 2^0 \cdot 0 \cdot y' + 2^{-1} x'_{-1} y' + 2^{-2} x'_{-2} y' + 2^{-3} x'_{-3} y' = \\ &= (((((0 + x'_{-3} 2y') \cdot 2^{-1} + x'_{-2} 2y') \cdot 2^{-1} + x'_{-1} 2y') \cdot 2^{-1} + 0 \cdot 2y') \cdot 2^{-1} \\ &\quad \boxed{\begin{array}{c} \square P'_0 \\ \square P'_1 \\ \square P'_2 \\ \vdots \\ \square P'_3 \end{array}} \quad p' \\ &\quad \boxed{P'_4 = p'} \end{aligned} \quad (4.3.1-10)$$

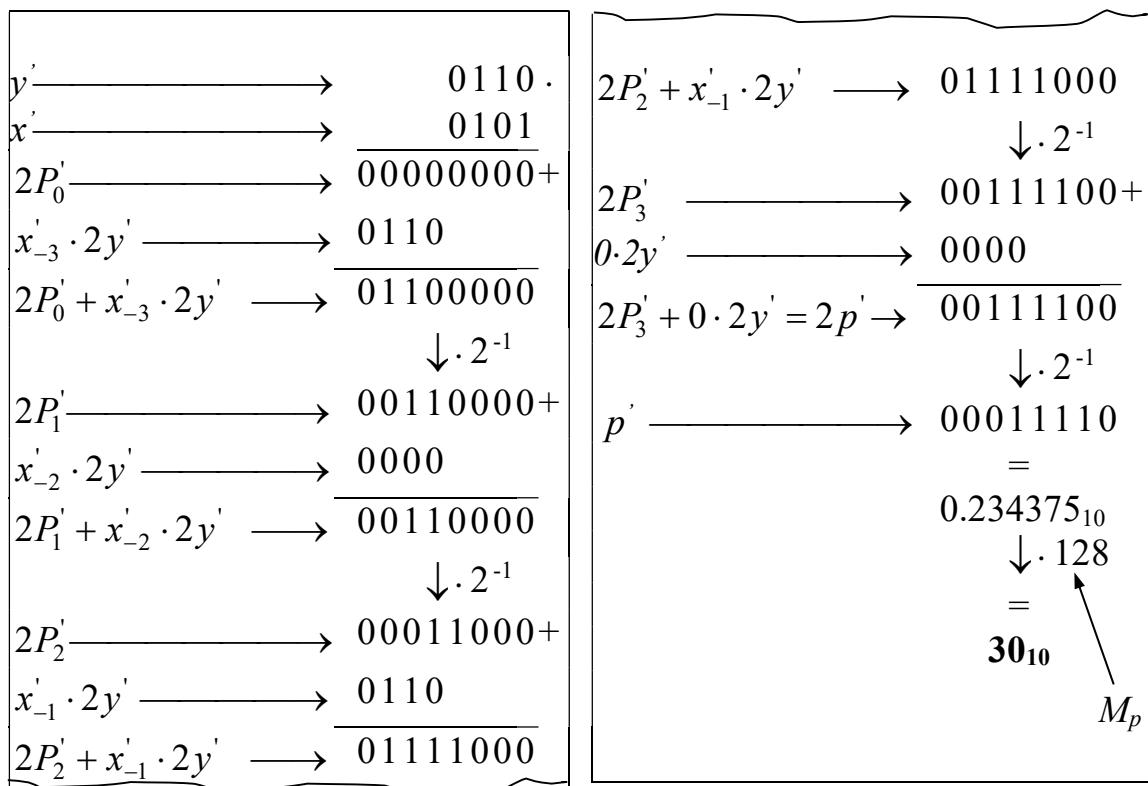
Prin artificiul de mai sus, au fost atinse două deziderate. Pe de o parte, s-a pus în evidență un algoritm recurrent de înmulțire, algoritm constând

într-un număr de pași egal cu lungimea de cod a reprezentării semn-mărime a operanzilor și în cadrul căruia în fiecare pas se efectuează o operație de adunare și una de înmulțire cu 2^{-1} , ambele ușor de implementat. Pe de altă parte, s-au asigurat premizele furnizării rezultatului nu pe 7 biți, cum ar impune relațiile (4.3.1-1) și (4.3.1-2), dacă s-ar opera efectiv cu x' și y' , ci pe 8 biți, lungime mult mai adekvată pentru un calculator. În acest caz, însă, M_p devine:

$$M_p = M_{op} \cdot 2M_{op} = 2M_{op}^2 = 2 \cdot 8^2 = 128 \quad (4.3.1-11)$$

ceea ce înseamnă că valoarea efectivă a produsului se va obține înmulțind exprimarea sa semn-mărime nu cu 64, ci cu 128.

Concret, lucrurile vor decurge după cum urmează:



4.3.2. Sinteză unui dispozitiv de înmulțire de numere în semn-mărime

Înmulțirea în semn mărime a două numere pozitive se realizează aşa cum s-a arătat în ultima parte a paragrafului 4.3.1. Înmulțirea în semn mărime a unui număr pozitiv cu un număr negativ, respectiv a două numere negative se realizează reducând problema la înmulțirea a două numere pozitive, după cum urmează: se rețin semnele operanziilor pentru ca, pe baza lor, să se stabilească semnul produsului, iar în calculele propriu-zise, se introduc, pe de o parte, operandul pozitiv, aşa cum este el, iar pe de altă parte opusul operandului negativ, respectiv, în al doilea caz, opusii celor doi operanzi negativi.

Fie $y'_0y'_{-1}y'_{-2}y'_{-3}$ imaginea semn-mărime a lui $y = s_y y_2 y_1 y_0$ și $x'_0x'_{-1}x'_{-2}x'_{-3}$ imaginea semn-mărime a lui $x = s_x x_2 x_1 x_0$, cu $s_y, s_x \in \{+, -\}$. Rezultă, aşadar, că, indiferent de valorile lui s_y și s_x , în calculele propriu-zise se vor introduce numerele $0y'_{-1}y'_{-2}y'_{-3} \equiv 0y_2y_1y_0$, respectiv $0x'_{-1}x'_{-2}x'_{-3} \equiv 0x_2x_1x_0$, iar semnul produsului va rezulta din expresia:

$$s_p = s_y \oplus s_x \quad (4.3.2-1)$$

Schema bloc a unui dispozitiv de înmulțire de numere în semn-mărime, concepută pe baza relației (4.3.1-10), este redată în figura 4.3.2_1.

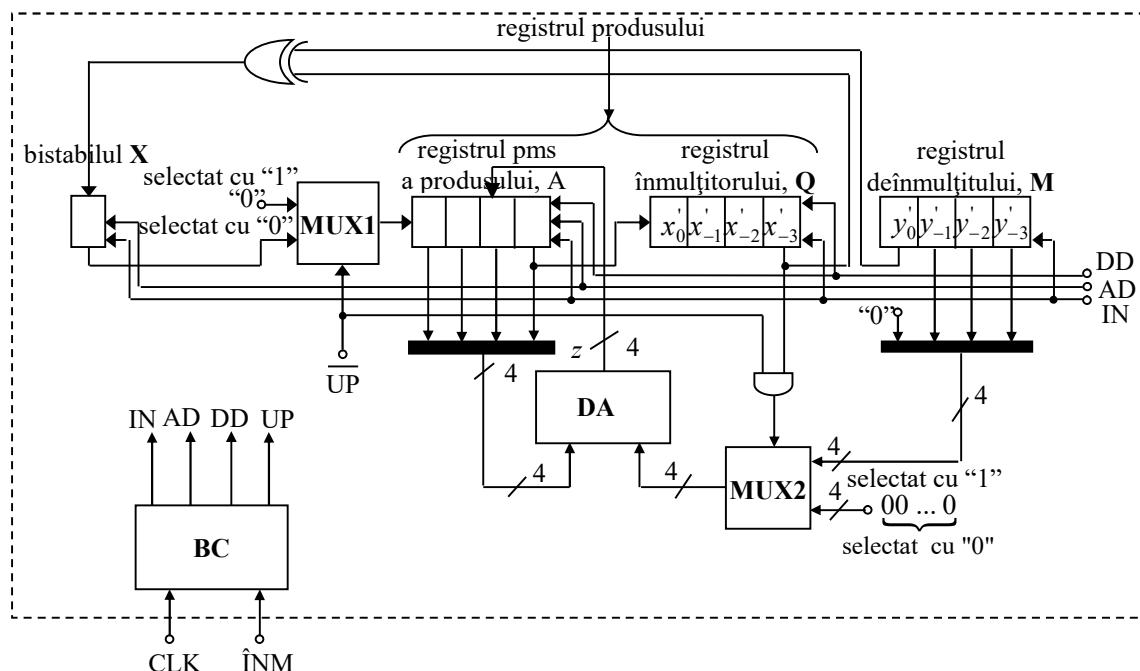


Fig. 4.3.2_1. Schema bloc a dispozitivului de înmulțire de numere în semn-mărime.

Elementele structurale ale schemei sunt:

- **X**: bistabil
- **MUX1**: multiplexor
- **A**: registru de deplasare dreapta, cu posibilități de încărcare paralelă
- **Q**: registru de deplasare dreapta, cu posibilități de încărcare paralelă
- **M**: registru cu posibilități de încărcare paralelă
- **DA**: dispozitiv de adunare pe 8 biți
- **MUX2**: multiplexor
- **BC**: bloc de comandă

Bistabilul **X** este destinat preluării semnului produsului, determinat cu ajutorul porții SAU EXCLUSIV în subpasul unu al ultimului pas al algoritmului, în scopul transmiterii lui spre rangul 0 al registrului **A**, în subpasul doi din cadrul aceluiași ultim pas (subpasul 1: adunare și încărcare rezultat; subpasul 2: înmulțire cu 2^{-1} = deplasare dreapta).

Multiplexorul **MUX1** are rolul de a introduce în rangul 0 al registrului **A** biți de valoare “0”, în subpașii de deplasare dreapta din cadrul pașilor ce-l preced pe ultimul, respectiv informația prezentă în bistabilul **X** –semnul produsului-, în subpasul de deplasare dreapta din cadrul pasului ultim.

Registrul **A** este sediul părții mai semnificative a produselor parțiale și, în final, sediul părții mai semnificative a produsului numerelor înmulțite. El este inițializat la zero la începutul fiecărei sesiuni de înmulțire. În primul subpas al fiecărui pas al algoritmului de înmulțire, registrul **A** este încărcat paralel cu suma dintre conținutul său anterior și rezultatul înmulțirii cifrei curente a înmulțitorului cu deînmulțitul. În al doilea subpas al fiecărui pas al algoritmului, registrul **A** este supus unei operații de deplasare la dreapta, în concatenare cu registrul **Q** –deplasare prin care se efectuează, de fapt, înmulțirea cu 2^{-1} prin care se generează produsele parțiale-; în timpul deplasării, prin intermediul multiplexorului **MUX1**, în rangul 0 al registrului **A** intră zero-uri, în pașii diferenți de ultimul, respectiv conținutul bistabilului **X**, reprezentând semnul produsului, în pasul ultim.

Registrul **Q** este, inițial, sediul înmulțitorului, dar, pe parcurs, în cadrul operațiilor de deplasare dreapta ce au loc în subpasul doi al fiecărui pas al algoritmului de înmulțire, biții înmulțitorului deja folosiți sunt ejectați prin partea dreaptă, în timp ce prin stânga intră ceea ce ieșe din rangul -3 al

registrului **A**, adică: câte un bit al produsului în curs de determinare. În finalul algoritmului, în registrul **Q** se va găsi partea mai puțin semnificativă a produsului numerelor ce se înmulțesc.

Registrul **M** este, de la început până la sfârșit, sediul deînmulțitului, adus aici în etapa de inițiere a unei noi sesiuni de înmulțire.

Multiplexorul **MUX2** are rolul de a transmite la intrarea 2 a dispozitivului de adunare **DA** fie deînmulțitul, atunci când cifra curentă a înmulțitorului –este vorba despre cifrele diferite de cea corespunzătoare semnului- este 1, fie valoarea *0000*, în cazul contrar. Atunci când cifra curentă a înmulțitorului este cea corespunzătoare semnului, se forțează trimiterea spre intrarea 2 a dispozitivului de adunare **DA** a valorii *0000*.

Dispozitivul de adunare **DA** are rolul de a calcula produsele parțiale, ca sumă dintre produsul parțial de până atunci și rezultatul înmulțirii cifrei curente a înmulțitorului cu deînmulțitul.

Blocul de comandă **BC** este un automat secvențial cu rolul de a sintetiza semnalele de comandă interne ale înmulțitorului, și anume: **IN**, **AD**, **DD** și **UP**. Semnificațiile acestor semnale sunt următoarele:

- **IN:** semnal de inițializare a înmulțitorului, generat intern, de către blocul de comandă **BC**, la începutul fiecărei noi sesiuni de lucru. Inițializarea contă în:
 - forțarea la *zero* a bistabilului **X** și a registrului **A**
 - încărcarea registrului **Q** cu înmulțitorul
 - încărcarea registrului **M** cu deînmulțitul
- **AD:** semnal de încărcare a registrului **A** cu partea mai semnificativă a noului produs parțial, constituit prin adunarea în dispozitivul de adunare, **DA**, a părții mai semnificative a produsului parțial curent, cu rezultatul înmulțirii cifrei curente a înmulțitorului cu deînmulțitul (de fapt: cu modului deînmulțitului) și a bistabilului **X** cu semnul produsului, în ultimul pas, semn determinat cu ajutorul porții SAU EXCLUSIV prezente în schemă, iar pe parcurs, cu informațiile nesemnificative și, în consecință neutilizate, de la ieșirea acestei porți.

- **DD:** semnal de deplasare la dreapta a informației din registrele **A** și **Q**, considerate în concatenație. Reamintim: în cadrul deplasărilor –deplasări prin care se efectuează, de fapt, înmulțirea cu 2^{-1} a produselor parțiale-, în rangul 0 al registrului **A** intră zero-uri, în pașii diferenți de ultimul, respectiv conținutul bistabilului **X**, reprezentând semnul produsului, în pasul ultim.
- **UP** semnal auxiliar ce marchează ajungerea în ultimul pas al algoritmului de înmulțire, pas în care sunt ajuși la rând biții celor doi operanzi corespunzători semnelor (remarcăm faptul că, în schemă, apare și negatul acestui semnal, **UP**).

Semnalele pe care schema le primește din exterior sunt:

- **CLK:** semnal de tact, primit de înmulțitor de la unitatea de comandă a procesorului. Acest semnal va pilota funcționarea tuturor elementelor secvențiale ale schemei: bistabilul **X**, registrele **A**, **Q** și **M** și blocul de comandă **BC**.
- **ÎNM:** semnal de declanșare a unei noi sesiuni de lucru a dispozitivului de înmulțire, primit de către acesta din partea unității de comandă a procesorului. Declanșarea unei noi sesiuni de lucru a dispozitivului de înmulțire se va face la fiecare tranziție de la 0 la 1 a acestui semnal.

Dacă stabilirea schemei bloc a înmulțitorului este -de la un punct încolo, cel puțin- o chestiune de imagine, nu la fel stau lucrurile cu proiectarea blocului său de comandă. Această proiectare se poate face algoritmice, aşa cum se va arăta în continuare.

Funcționarea dorită pentru schemă este cea descrisă prin organograma din figura 4.3.2_2. Se remarcă faptul că se parcurg 10 stări –o stare fiind un ciclu al semnalului de tact-, dintre care una este starea de inactivitate, **S0**, alta este starea de initializare, **S1**, iar celelalte opt, **S2-S9**, sunt stările corespunzătoare celor 4 pași ai algoritmului de înmulțire propriu-zisă, câte 2 pentru fiecare cifră binară din reprezentarea semn-mărime a înmulțitorului. Rezultă, aşadar, că blocul de comandă al înmulțitorului va fi un automat secvențial cu 10 stări. Să-l proiectăm, acum, cu metoda diagramelor VID, presupunând că implementarea vrem să o facem cu bistabile J-K.

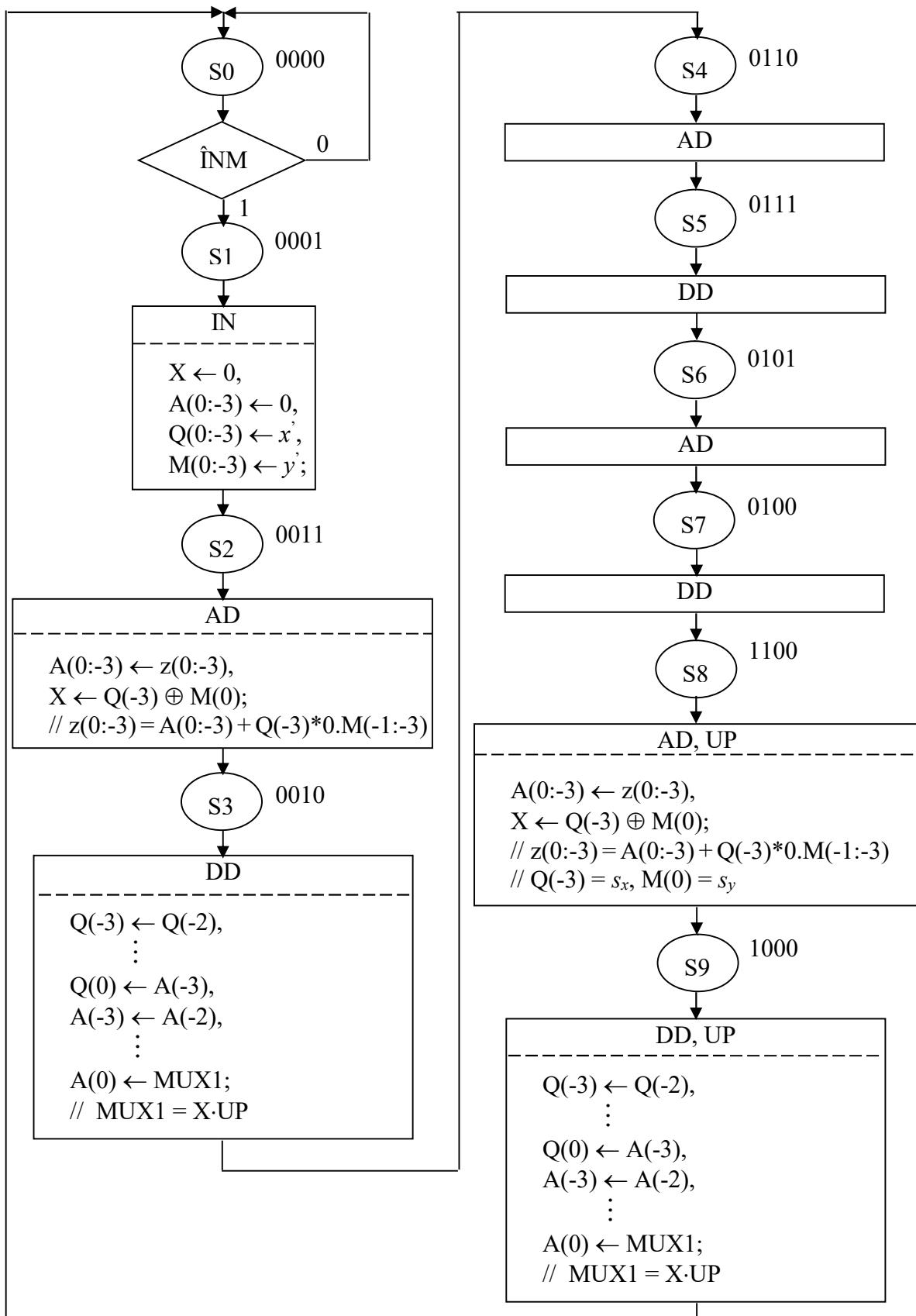


Fig. 4.3.2_2. Organigrama funcționării dispozitivului de înmulțire.

Mai întâi, se impune codificarea adiacentă a stărilor. Să admitem varianta de codificare indicată în figura 4.3.2_2. Diagrama acestei codificări este reprezentată în figura 4.3.2_3, în ipoteza că bistabilele blocului de comandă sunt notate cu $B_3 - B_0$.

		00	01	11	10
		S0	S1	S2	S3
00		S7	S6	S5	S4
01		S8	*	*	*
11		S9	*	*	*
10					

Fig. 4.3.2_3. Diagrama codificării stărilor.

În continuare, se întocmește, pentru fiecare din bistabilele $B_0 - B_3$, “diagrama stărilor următoare”, “diagrama lui J” și “diagrama lui K” –vezi figurile 4.3.2_4-4.3.2_15-, ultimele două ținând seama de ecuația de funcționare a bistabilului J-K:

$$Q_{n+1} = J \cdot \bar{Q}_n + \bar{K} \cdot Q_n \quad (4.3.2-2)$$

		00	01	11	10
		ÎNM	1	0	0
00		0	0	1	1
01		0	*	*	*
11		0	*	*	*
10		0	*	*	*

Fig. 4.3.2_4. Diagrama stărilor următoare pentru bistabilul B_0 .

		00	01	11	10
		ÎNM	*	*	0
00		0	*	*	1
01		0	*	*	*
11		0	*	*	*
10		0	*	*	*

$$\Rightarrow J_0 = \bar{B}_3 \cdot \bar{B}_2 \cdot \bar{B}_1 \cdot \hat{I}NM + B_2 \cdot B_1$$

Fig. 4.3.2_5. Diagrama lui J pentru bistabilul B_0 .

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	*	0	1	*
		01	*	1	0	*
		11	*	*	*	*
		10	*	*	*	*

$\Rightarrow K_0 = B_2 \cdot \overline{B}_1 + \overline{B}_2 \cdot B_1$

Fig. 4.3.2_6. Diagrama lui K pentru bistabilul B₀.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	0	1	1	1
		01	0	0	0	1
		11	0	*	*	*
		10	0	*	*	*

Fig. 4.3.2_7. Diagrama stăriilor următoare pentru bistabilul B₁.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	0	1	*	*
		01	0	0	*	*
		11	0	*	*	*
		10	0	*	*	*

$\Rightarrow J_1 = \overline{B}_2 \cdot B_0$

Fig. 4.3.2_8. Diagrama lui J pentru bistabilul B₁.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	*	*	0	0
		01	*	*	1	0
		11	*	*	*	*
		10	*	*	*	*

$\Rightarrow K_1 = B_2 \cdot B_0$

Fig. 4.3.2_9. Diagrama lui K pentru bistabilul B₁.

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
00				0	0	0	1
01				1	1	1	1
11				0	*	*	*
10				0	*	*	*

Fig. 4.3.2_10. Diagrama stăriilor următoare pentru bistabilul B₂.

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
00				0	0	0	1
01				*	*	*	*
11				*	*	*	*
10				0	*	*	*

$$\Rightarrow J_2 = B_1 \cdot \bar{B}_0$$

Fig. 4.3.2_11. Diagrama lui J pentru bistabilul B₂.

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
00				*	*	*	*
01				0	0	0	0
11				1	*	*	*
10				*	*	*	*

$$\Rightarrow K_2 = B_3$$

Fig. 4.3.2_12. Diagrama lui K pentru bistabilul B₂.

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
00				0	0	0	0
01				1	0	0	0
11				1	*	*	*
10				0	*	*	*

Fig. 4.3.2_13. Diagrama stăriilor următoare pentru bistabilul B₃.

		B ₁ B ₀	B ₃ B ₂		
		00	01	11	10
00		0	0	0	0
01	01	1	0	0	0
	11	*	*	*	*
10	10	*	*	*	*

$$\Rightarrow J_3 = B_2 \cdot \bar{B}_1 \cdot \bar{B}_0$$

Fig. 4.3.2_14. Diagrama lui J pentru bistabilul B₃.

		B ₁ B ₀	B ₃ B ₂		
		00	01	11	10
00		*	*	*	*
01	01	*	*	*	*
	11	0	*	*	*
10	10	1	*	*	*

$$\Rightarrow K_3 = \bar{B}_2$$

Fig. 4.3.2_15. Diagrama lui K pentru bistabilul B₃.

Așadar, ecuațiile intrărilor bistabilelor **B₀ – B₃** vor fi:

- pentru bistabilul **B₀**:

$$J_0 = \bar{B}_3 \cdot \bar{B}_2 \cdot \bar{B}_1 \cdot MUL + B_2 \cdot B_1 \quad (4.3.2-3)$$

$$K_0 = B_2 \cdot \bar{B}_1 + \bar{B}_2 \cdot B_1 \quad (4.3.2-4)$$

- pentru bistabilul **B₁**:

$$J_1 = \bar{B}_2 \cdot B_0 \quad (4.3.2-5)$$

$$K_1 = B_2 \cdot B_0 \quad (4.3.2-6)$$

- pentru bistabilul **B₂**:

$$J_2 = B_1 \cdot \bar{B}_0 \quad (4.3.2-7)$$

$$K_2 = B_3 \quad (4.3.2-8)$$

- pentru bistabilul **B₃**:

$$J_3 = B_2 \cdot \bar{B}_1 \cdot \bar{B}_0 \quad (4.3.2-9)$$

$$K_3 = \bar{B}_2 \quad (4.3.2-10)$$

A mai rămas să determinăm ecuațiile ieșirilor, ieșiri care sunt: IN, AD, DD și UP. Întocmim, pentru fiecare dintre acestea, câte o “diagramă a ieșirii” –vezi figurile 4.3.2_16 - 4.3.2_19-.

B_3B_2	B_1B_0	00	01	11	10
00	0	1	0	0	
01	0	0	0	0	
11	0	*	*	*	*
10	0	*	*	*	*

$$\Rightarrow IN = \bar{B}_2 \cdot \bar{B}_1 \cdot B_0$$

Fig. 4.3.2_16. Diagrama ieșirii IN.

B_3B_2	B_1B_0	00	01	11	10
00	0	0	1	0	
01	0	1	0	1	
11	1	*	*	*	*
10	0	*	*	*	*

$$\Rightarrow AD = \bar{B}_2 \cdot B_1 \cdot B_0 + B_2 \cdot B_1 \cdot \bar{B}_0 + B_3 \cdot B_2 + B_2 \cdot \bar{B}_1 \cdot B_0$$

Fig. 4.3.2_17. Diagrama ieșirii AD.

B_3B_2	B_1B_0	00	01	11	10
00	0	0	0	1	
01	1	0	1	0	
11	0	*	*	*	*
10	1	*	*	*	*

$$\Rightarrow DD = \bar{B}_2 \cdot B_1 \cdot \bar{B}_0 + B_2 \cdot B_1 \cdot B_0 + B_3 \cdot \bar{B}_2 + \bar{B}_3 \cdot B_2 \cdot \bar{B}_1 \cdot \bar{B}_0$$

Fig. 4.3.2_18. Diagrama ieșirii DD.

B_3B_2	B_1B_0	00	01	11	10
00	0	0	0	0	
01	0	0	0	0	
11	1	*	*	*	*
10	1	*	*	*	*

$$\Rightarrow UP = B_3$$

Fig. 4.3.2_19. Diagrama ieșirii UP.

Așadar, ecuațiile ieșirilor IN, AD, DD și UP vor fi:

$$IN = \bar{B}_2 \cdot \bar{B}_1 \cdot B_0 \quad (4.3.2-11)$$

$$AD = \bar{B}_2 \cdot B_1 \cdot B_0 + B_2 \cdot B_1 \cdot \bar{B}_0 + B_3 \cdot B_2 + B_2 \cdot \bar{B}_1 \cdot B_0 \quad (4.3.2-12)$$

$$DD = \bar{B}_2 \cdot B_1 \cdot \bar{B}_0 + B_2 \cdot B_1 \cdot B_0 + B_3 \cdot \bar{B}_2 + \bar{B}_3 \cdot B_2 \cdot \bar{B}_1 \cdot \bar{B}_0 \quad (4.3.2-13)$$

$$UP = B_3 \quad (4.3.2-14)$$

Pe baza ecuațiilor (4.3.2-3) - (4.3.2-14), se obține pentru blocul de comandă al dispozitivului de înmulțire de numere în semn-mărime schema din figura 4.3.2_20.

Considerăm contextul adecvat pentru a introduce și un alt element consacrat de descriere a funcționării schemelor logice secvențiale și anume: cronograma. Așadar, redăm în figura 4.3.2_21 cronograma schemei din figura 4.3.2_20.

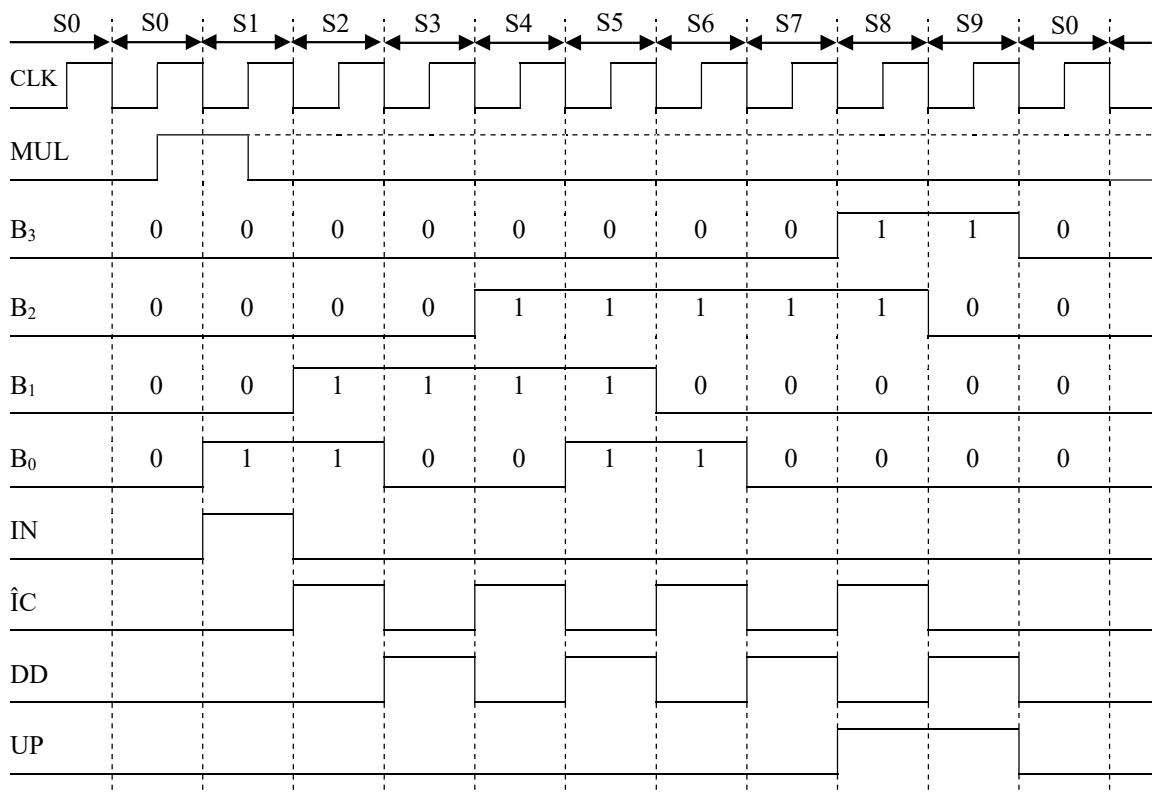


Fig. 4.3.2_21. Cronograma schemei din figura 4.3.2_20.

4.3.3. Sinteza unui dispozitiv de înmulțire combinațional

Procedeul de înmulțire considerat anterior implică o execuție secvențială. El are avantajul că se implementează printr-o circuistică relativ restrânsă, dar și dezavantajul că necesită pentru efectuarea înmulțirii un timp cu atât mai mare cu cât operează pe mai mulți biți. Progresele înregistrate de tehnologia microelectronică au făcut posibilă și convenabilă prin prisma performanță-cost realizarea de dispozitive de înmulțire combinaționale, ultrarapide.

La baza conceperii unui dispozitiv de înmulțire combinațional stau următoarele considerente:

Dacă cei doi operanzi, exprimați în semn-mărime, sunt:

$$y' = \overline{y'_0 y'_{-1} \dots y'_{-(n-1)}} \quad (4.3.3_1)$$

$$x' = \overline{x'_0 x'_{-1} \dots x'_{-(n-1)}} \quad (4.3.3_2)$$

modulele lor fiind:

$$|y'| = \overline{0 y'_{-1} \dots y'_{-(n-1)}} \quad (4.3.3_3)$$

$$|x'| = \overline{0 x'_{-1} \dots x'_{-(n-1)}} \quad (4.3.3_4)$$

atunci:

$$\begin{aligned} |P'| &= \overline{0 p'_{-1} p'_{-2} \dots p'_{-(2n-2)}} = |y'| \cdot |x'| = \left(\sum_{i=1}^{n-1} 2^{-i} y'_{-i} \right) \cdot \left(\sum_{i=1}^{n-1} 2^{-i} x'_{-i} \right) = \\ &= 2^{-2}(x'_{-1} y'_{-1}) + 2^{-3}(x'_{-1} y'_{-2} + x'_{-2} y'_{-1}) + \dots + 2^{-(2n-2)}(x'_{-(n-1)} y'_{-(n-1)}) \end{aligned} \quad (4.3.3_5)$$

iar:

$$P' = \overline{p'_0 p'_{-1} p'_{-2} \dots p'_{-(2n-2)}} \quad (4.3.3_6)$$

cu:

$$p'_0 = x'_0 \oplus y'_0 \quad (4.3.3_7)$$

$$p'_{-1} = c'_{-2} \quad (4.3.3_8)$$

$$p'_{-2} = x'_{-1} y'_{-1} + c'_{-3} \quad (4.3.3_9)$$

$$p'_{-3} = x'_{-1} y'_{-2} + x'_{-2} y'_{-1} + c'_{-4} \quad (4.3.3_10)$$

⋮

$$p'_{-(2n-2)} = x'_{-(n-1)} y'_{-(n-1)} \quad (4.3.3_11)$$

În cazul concret $n=4$, se va avea:

$$y' = \overline{y'_0 y'_{-1} y'_{-2} y'_{-3}} \quad (4.3.3_12)$$

$$x' = \overline{x'_0 x'_{-1} x'_{-2} x'_{-3}} \quad (4.3.3_13)$$

$$|y'| = \overline{0 y'_{-1} y'_{-2} y'_{-3}} \quad (4.3.3_14)$$

$$|x'| = \overline{0 x'_{-1} x'_{-2} x'_{-3}} \quad (4.3.3_15)$$

$$\begin{aligned} |P'| = & \overline{0 p'_{-1} p'_{-2} p'_{-3} p'_{-4} p'_{-5} p'_{-6}} = y' \cdot x' = 2^{-2}(x'_{-1} y'_{-1}) + 2^{-3}(x'_{-1} y'_{-2} + x'_{-2} y'_{-1}) + \\ & + 2^{-4}(x'_{-1} y'_{-3} + x'_{-2} y'_{-2} + x'_{-3} y'_{-1}) + 2^{-5}(x'_{-2} y'_{-3} + x'_{-3} y'_{-2}) + 2^{-6}(x'_{-3} y'_{-3}) \end{aligned} \quad (4.3.3_16)$$

iar:

$$P' = \overline{p'_0 p'_{-1} p'_{-2} p'_{-3} p'_{-4} p'_{-5} p'_{-6}} \quad (4.3.3_17)$$

cu:

$$p'_0 = x'_0 \oplus y'_0 \quad (4.3.3_18)$$

$$p'_{-1} = c'_{-2} \quad (4.3.3_19)$$

$$p'_{-2} = x'_{-1} y'_{-1} + c'_{-3} \quad (4.3.3_20)$$

$$p'_{-3} = x'_{-1} y'_{-2} + x'_{-2} y'_{-1} + c'_{-4} \quad (4.3.3_21)$$

$$p'_{-4} = x'_{-1} y'_{-3} + x'_{-2} y'_{-2} + x'_{-3} y'_{-1} + c'_{-5} \quad (4.3.3_22)$$

$$p'_{-5} = x'_{-2} y'_{-3} + x'_{-3} y'_{-2} \quad (4.3.3_23)$$

$$p'_{-6} = x'_{-3} y'_{-3} \quad (4.3.3_24)$$

Pe baza relațiilor (4.3.3_17)...(4.3.3_24), rezultă, pentru înmulțitorul combinațional pe 4 biți, schema din figura 4.3.3_1.

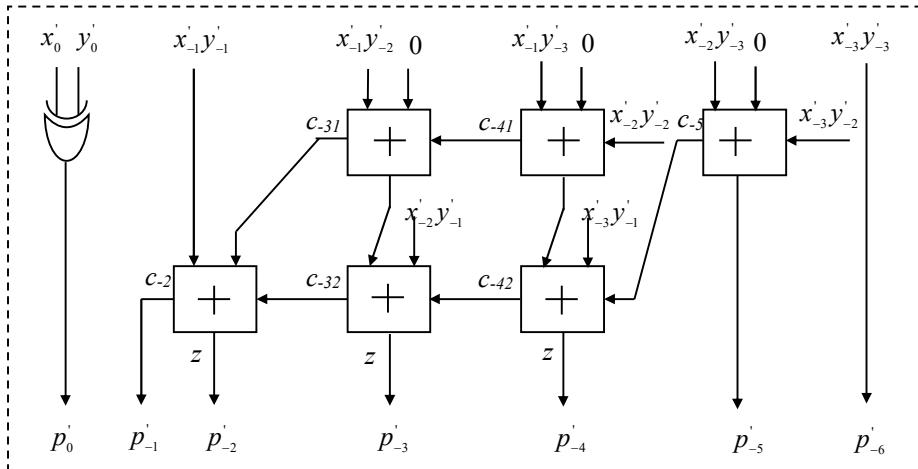


Fig. 4.3.3_1. Schema înmulțitorului combinational (cazul operanzilor pe 4 biți).

4.4. Dispozitive de împărțire

4.4.1. Principii

Operația de împărțire se efectuează în calculator cel mai convenabil în semn-mărime. În procesul de împărțire propriu-zisă, se operează cu modulele numerelor, semnul câtului urmând a fi stabilit separat, ca funcție *SAU EXCLUSIV* între bitul corespunzător semnului deîmpărțitului și bitul corespunzător semnului împărțitorului.

Fie cei doi operanzi –deîmpărțitul, respectiv împărțitorul- exprimați în semn-mărime, după cum urmează:

$$x' = \overline{x'_0 x'_{-1} \dots x'_{-(2n-2)}} \quad (4.4.1_1)$$

$$y' = \overline{y'_0 y'_{-1} \dots y'_{-(n-1)}} \quad (4.4.1_2)$$

Pentru modulele lor, vom avea:

$$|x'| = \overline{0 x'_{-1} \dots x'_{-(2n-2)}} \quad (4.4.1_3)$$

$$|y'| = \overline{0 y'_{-1} \dots y'_{-(n-1)}} \quad (4.4.1_4)$$

Este lesne de înțeles că, în semn mărime, dacă la nivel de module, deîmpărțitul ar fi mai mare sau egal cu împărțitorul, atunci ar rezulta un cât de modul supraunitar, care, în consecință, ar fi nereprezentabil (reamintim: în semn mărime, numerele pozitive sunt reprezentate prin numere din intervalul $(0; 1)$, iar numerele negative, prin numere din intervalul $(1; 2)$, în timp ce numărul 0 are dublă imagine, pe 0 și pe 1). Așadar, este obligatoriu ca:

$$|x'| < |y'| \quad (4.4.1_5)$$

putând avea fie:

$$|x'| \in \left[0; \frac{|y'|}{2} \right) \quad (4.4.1_6)$$

fie:

$$|x'| \in \left[\frac{|y'|}{2}; |y'| \right) \quad (4.4.1_7)$$

Evident, în primul caz $\frac{|x'|}{|y'|} < \frac{1}{2}$ și, în consecință, bitul de pondere 2^{-1} al câtului va fi 0, iar în cazul al doilea $\frac{|x'|}{|y'|} \geq \frac{1}{2}$ și, în consecință, bitul de pondere 2^{-1} al câtului va fi 1. Distincția între cele două cazuri se va face în practică prin testarea semnului expresiei $2|x'| - |y'|$.

Aceeași logică se va aplica, firește, și pentru determinarea bițiilor de pondere $2^{-2}, 2^{-3}, \dots$, considerând, însă, în locul deîmpărțitului inițial, ceea ce a mai rămas din el în urma pașilor precedenți, adică: deîmpărțiturile parțiale sau, cum se spune mai frecvent în literatură, “resturile parțiale”.

O precizare se impune. Așa cum s-a arătat în paragraful 4.3.1, la înmulțirea a doi operanzi reprezentați în semn-mărime pe câte “ n ” biți, se obține, natural, un rezultat exprimat pe $2n-1$ biți. Există, însă, posibilitatea ca acest rezultat să fie reprezentat pe $2n$ biți –așa cum este, evident, mult mai adecvat-, dacă se iau două măsuri: pe de o parte, se adaugă un bit de mărime cu valoarea 0, imediat după cifra corespunzătoare semnului, iar pe de altă parte, se calculează valoarea efectivă a numărului inițial înmulțind imaginea sa semn-mărime nu cu M , ci cu $2M$; și toate acestea, pentru că $sm(x)|_{lc_sm(x)=2n} = \frac{1}{2} sm(x)|_{lc_sm(x)=2n-1}$.

Pe baza aceleiași logici, vom considera deîmpărțitul reprezentat nu pe $2n-1$ biți, ci pe $2n$ biți, neuitând să ținem seamă că $x'|_{pe\ 2n\ biți} = \frac{1}{2} x'|_{pe\ 2n-1\ biți}$.

Ca o chintesență a celor de mai sus, rezultă algoritmul de împărțire prezentat în organograma din figura 4.4.1_1.

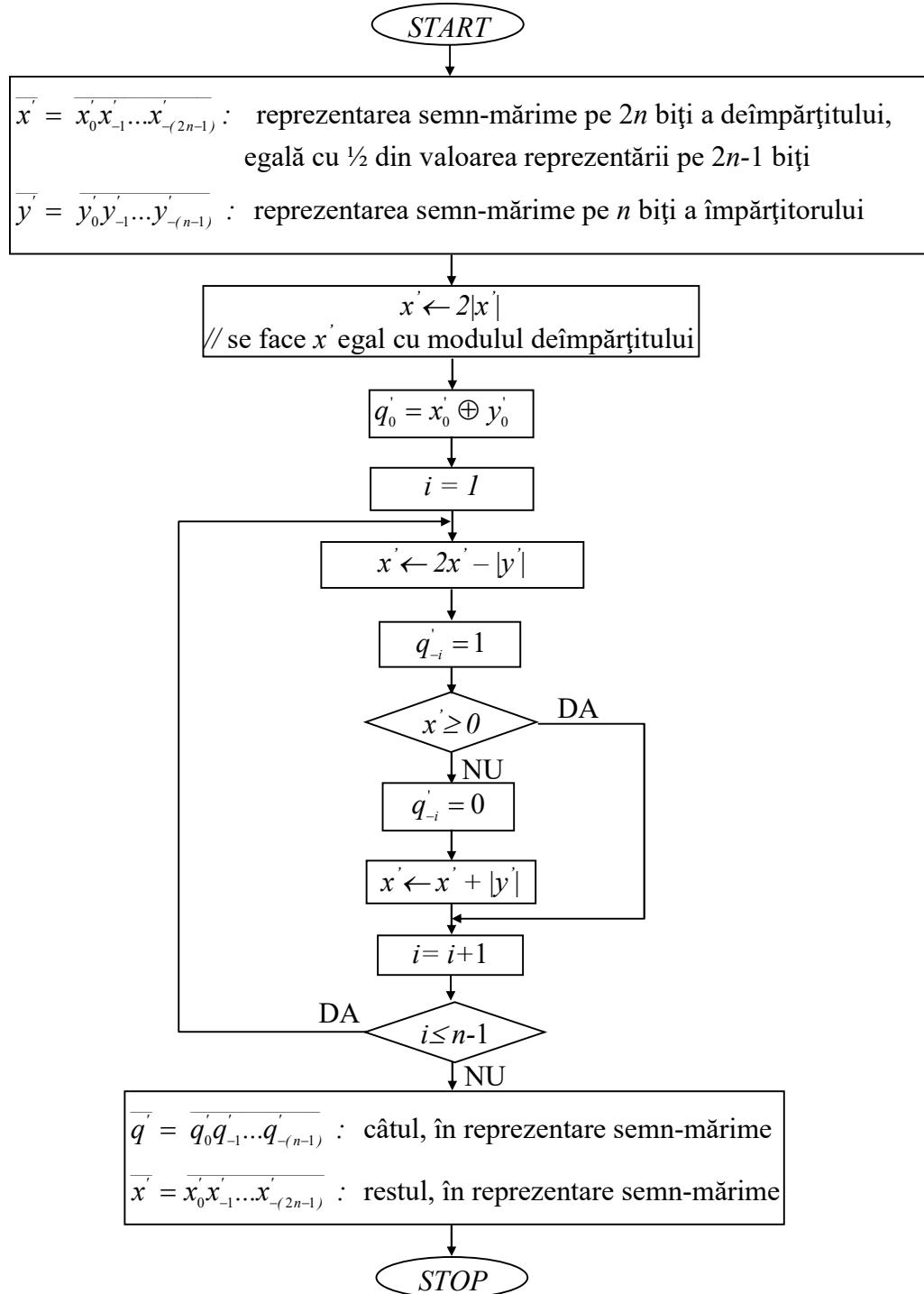


Fig. 4.4.1_1. Organigramma algoritmului fundamental de împărțire.

4.4.2. Sinteză unui dispozitiv de împărțire de numere în semn mărime, operând cu refacerea resturilor

Algoritmul fundamental de împărțire prezentat în organigramă din figura 4.4.1_1 constituie ceea ce se cheamă “metoda de împărțire cu refacerea resturilor”. Un dispozitiv de împărțire bazat pe această metodă are schema bloc din figura 4.4.2_1, concepută în ipoteza $n = 4$.

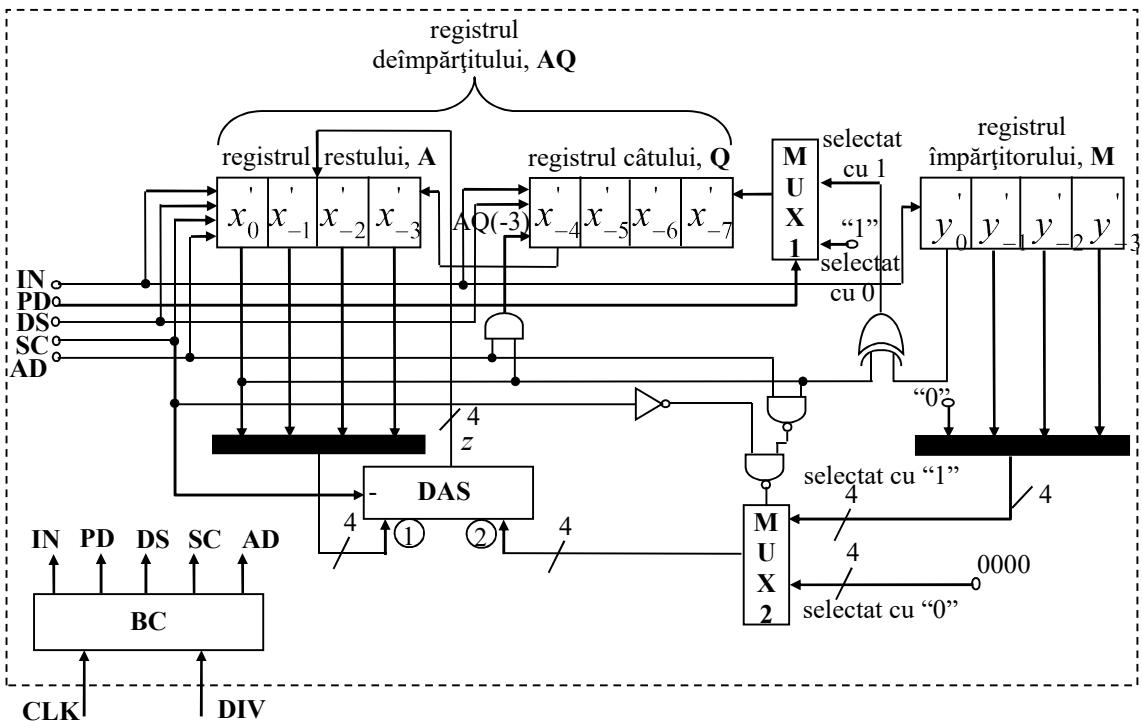


Fig. 4.4.2_1. Schema bloc a dispozitivului de împărțire cu refacerea resturilor.

Elementele structurale ale schemei sunt:

- **A:** registrul de deplasare stânga, cu posibilități de încărcare paralelă
- **Q:** registrul de deplasare stânga, cu posibilități de încărcare paralelă
- **M:** registrul cu posibilități de încărcare paralelă
- **DAS:** dispozitiv de adunare și scădere paralel, pe 4 biți
- **MUX1, MUX2:** multiplexoare
- **BC:** bloc de comandă

Registrul **A** este inițial, împreună cu registrul **Q**, sediul deîmpărțitului reprezentat în semn-mărime pe 8 biți, iar în final, sediul restului. Așa cum s-

a arătat, considerentele de scalare ar fi cerut ca, în cazul operării pe 4 biți la nivelul împărțitorului și câtului, deîmpărțitul să fie reprezentat pe 7 biți. Faptul că în schema noastră el apare pe 8 biți -precizăm din nou: reprezentarea semn-mărime pe 8 biți a unui număr are valoarea egală cu $\frac{1}{2}$ din valoarea reprezentării același număr pe 7 biți- face ca prima operație efectuată asupra registrului ce-l găzduiește să fie una de înmulțire cu 2. Această înmulțire se face prin deplasare la stânga cu un rang. Cu ocazia deplasării, bitul corespunzător semnului deîmpărțitului va dispărea, nu însă înainte ca el să fie folosit, împreună cu bitul corespunzător semnului împărțitorului, la determinarea semnului câtului, semn care va fi introdus în rangul -3 al registrului **Q**. În continuare, registrul **A** va fi, succesiv, supus câte unui triplet de operații, cuprinzând, în ordine: o deplasare la stânga în concatenare cu registrul **Q** (calculul lui $2|x'|$), o încărcare cu rezultatul scăderii din el a modulului împărțitorului (încărcarea cu $2x' - |y'|$ -vezi figura 4.4.1_1) și o încărcare cu rezultatul adunării la el a modulului împărțitorului (încărcarea cu $(2x' - |y'|) + |y'|$ -vezi figura 4.4.1_1) sau a valorii 0, după caz.

Registrul **Q** este, inițial, împreună cu registrul **A**, sediul deîmpărțitului, iar în final, sediul câtului. Cu ocazia primei deplasări la stânga la care el este supus în concatenare cu registrul **A**, în rangul său -3 va fi introdus semnul câtului. În continuare, la fiecare deplasare, în acest rang -3 se va introduce un 1, pentru alternativa că cifra câtului aflată în curs de determinare în pasul respectiv este 1. Dacă lucrurile vor sta contrar –adică: se va constata că $2x' - |y'| < 0$ -, atunci în ciclul de încărcare după adunare a registrului **A**, rangul -3 al registrului **Q** va fi forțat la 0 (asta prin semnalul **AQ(-3)** din figura 4.4.2_1; “**AQ(-3)**” vine de la **anulează Q(-3)**).

Registrul **M** este, de la început până la sfârșit, sediul împărțitorului, adus aici în etapa de inițiere a unei noi sesiuni de împărțire.

Multiplexorul **MUX1** are rolul de a furniza pentru introducere în rangul -3 al registrului **Q**, la început, când semnalul **PD** are valoarea 1, semnul câtului, iar apoi, când semnalul **PD** are valoarea 0, mereu câte un 1.

Multiplexorul **MUX2** are rolul de a furniza pentru intrarea 2 a dispozitivului de adunare și scădere **DAS**, în ciclurile de scădere, modulul împărțitorului, iar în ciclurile de adunare, fie modulul împărțitorului, astă dacă după scăderea ce le precede, conținutul lui **A** a ajuns negativ (adică: $A(0) = 1$), fie valoarea 0000, altfel.

Dispozitivul de adunare și scădere **DAS** este un dispozitiv de adunare și scădere binar paralel oarecare, cu rolul de a calcula, pe de o parte, diferențele $2x' - |y'|$, iar pe de altă parte, sumele ”de refacere”, respectiv ”de conservare” dintre aceste diferențe și $|y'|$, respectiv 0, după caz.

Blocul de comandă **BC** este un automat secvențial cu rolul de a sintetiza semnalele de comandă interne ale împărțitorului și anume: **IN**, **PD**, **DS**, **SC** și **AD**. Semnificațiile acestor semnale sunt următoarele:

- **IN:** semnal de inițializare a împărțitorului, generat intern, de către blocul de comandă **BC**, la începutul fiecărei noi sesiuni de lucru. Inițializarea constă în:
 - încărcarea registrului **A** cu partea mai semnificativă din reprezentarea semn-mărime pe 8 biți a deîmpărțitului
 - încărcarea registrului **Q** cu partea mai puțin semnificativă din reprezentarea semn-mărime pe 8 biți a deîmpărțitului
 - încărcarea registrului **M** cu împărțitorul, în reprezentare semn-mărime pe 4 biți
- **PD:** semnal de marcare a faptului că este în derulare prima operație de deplasare la stânga a informației din registrele **A** și **Q** concatenate. Acest semnal servește comandării multiplexorului **MUX1**, astfel încât el să furnizeze pentru introducere în rangul -3 al registrului **Q** fie semnul câtului, fie un 1.
- **DS:** semnal de deplasare la stânga a informației din registrele **A** și **Q**, considerate în concatenare. La prima deplasare (adică: în timp ce **PD=1**), în rangul -3 al registrului **Q** se va introduce semnul câtului, iar la celelalte deplasări, câte un 1, pentru alternativa că cifra câtului aflată în curs de determinare în pasul respectiv este 1. Dacă lucrurile vor sta contrar –adică: se va constata că $2x' - |y'| < 0$, atunci, în ciclul de adunare, rangul -3 al registrului **Q** va fi forțat la 0.
- **SC:** semnal de încărcare a registrului **A** cu rezultatul scăderii din el a modulului împărțitorului (semnal de calcul al lui $2x' - |y'|$).
- **AD:** semnal de încărcare a registrului **A** cu rezultatul adunării la el a modulului împărțitorului, respectiv a valorii *0000*, după caz, în funcție de semnul cantității $2x' - |y'|$.

Semnalele pe care schema le primește din exterior sunt:

- **CLK:** semnal de tact, primit de împărțitor de la unitatea de comandă a procesorului. Acest semnal va pilota funcționarea tuturor elementelor secvențiale ale schemei: registrele **A**, **Q** și **M** și blocul de comandă **BC**.
- **DIV:** semnal de declanșare a unei noi sesiuni de lucru a dispozitivului de împărțire, primit de către acesta din partea unității de comandă a procesorului. Declanșarea unei noi sesiuni de lucru a împărțitorului se va face la fiecare tranziție de la 0 la 1 a acestui semnal.

Blocul de comandă trebuie să aibă 12 stări:

- 1 stare de repaus
- 1 stare de inițializare
- 1 stare de deplasare la stânga preliminară a registrelor **A** și **Q**, tratate concatenat
- $3 \times 3 = 9$ stări de deplasare la stânga a registrelor **A** și **Q** tratate concatenat, scădere $2x' - |y'|$ și adunare $(2x' - |y'|) + |y'|$, respectiv $(2x' - |y'|) + 0$.

Vom proiecta acest bloc cu metoda diagramelor VID, cum am procedat și la dispozitivul de înmulțire. Evident, vom avea nevoie tot de 4 bistabile, blocul trebuind să aibă 12 stări, cum am văzut. Notăm aceste bistabile cu **B₃-B₀**.

Organograma de funcționare a împărțitorului este redată în figura 4.4.2_2.

CAP. 4. DESPRE PROCESOR. UNITATEA ARITMETICO-LOGICĂ

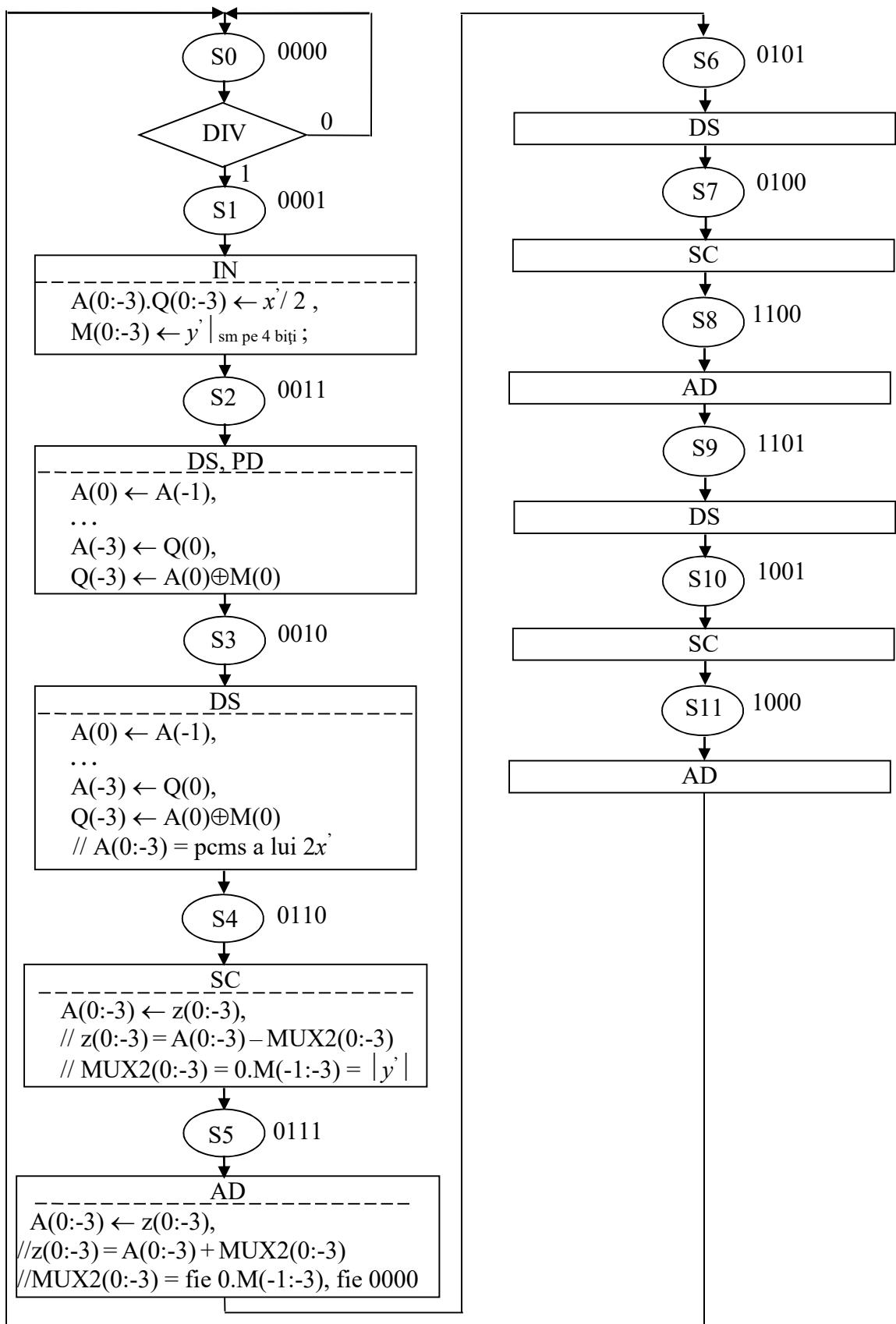


Fig. 4.4.2_2. Organigramă funcționării dispozitivului de împărțire cu refacerea resturilor.

În continuare, codificăm stările blocului de comandă, construind diagrama din figura 4.4.2_3.

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
				S0	S1	S2	S3
				S7	S6	S5	S4
				S8	S9	*	*
				S11	S10	*	*

Fig. 4.4.2_3. Diagrama codificării stărilor.

Mai departe, întocmim, pentru fiecare din bistabilele $B_0 - B_3$, “diagrama stărilor următoare”, “diagrama lui J” și “diagrama lui K” –vezi figurile 4.4.2_4-4.4.2_15-, ultimele două ținând seama de ecuația de funcționare a bistabilului J-K:

$$Q_{n+1} = J \cdot \overline{Q}_n + \overline{K} \cdot Q_n \quad (4.4.2-1)$$

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
				DIV	1	0	0
				0	0	1	1
				1	1	*	*
				0	0	*	*

Fig. 4.4.2_4. Diagrama stărilor următoare pentru bistabilul B_0 .

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
				DIV	*	*	0
				0	*	*	1
				0	*	*	*
				0	*	*	*

$$\Rightarrow J_0 = \overline{B}_3 \cdot \overline{B}_2 \cdot \overline{B}_1 \cdot DIV + B_2 \cdot B_1$$

Fig. 4.4.2_5. Diagrama lui J pentru bistabilul B_0 .

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	*	0	1	*
		01	*	1	0	*
		11	*	0	*	*
		10	*	1	*	*

$$\Rightarrow K_0 = \bar{B}_3 \cdot B_2 \cdot \bar{B}_1 + B_3 \cdot \bar{B}_2 \cdot \bar{B}_1 + \bar{B}_2 \cdot B_1$$

Fig. 4.4.2_6. Diagrama lui K pentru bistabilul B₀.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	0	1	1	1
		01	0	0	0	1
		11	0	0	*	*
		10	0	0	*	*

Fig. 4.4.2_7. Diagrama stăriilor următoare pentru bistabilul B₁.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	0	1	*	*
		01	0	0	*	*
		11	0	0	*	*
		10	0	0	*	*

$$\Rightarrow J_1 = \bar{B}_3 \cdot \bar{B}_2 \cdot B_0$$

Fig. 4.4.2_8. Diagrama lui J pentru bistabilul B₁.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	*	*	0	0
		01	*	*	1	0
		11	*	*	*	*
		10	*	*	*	*

$$\Rightarrow K_1 = B_2 \cdot B_0$$

Fig. 4.4.2_9. Diagrama lui K pentru bistabilul B₁.

CAP. 4. DESPRE PROCESOR. UNITATEA ARITMETICO-LOGICĂ

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	0	0	0	1
		01	1	1	1	1
		11	1	0	*	*
		10	0	0	*	*

Fig. 4.4.2_10. Diagrama stărilor următoare pentru bistabilul B₂.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	0	0	0	1
		01	*	*	*	*
		11	*	*	*	*
		10	0	0	*	*

$$\Rightarrow J_2 = B_1 \cdot \overline{B}_0$$

Fig. 4.4.2_11. Diagrama lui J pentru bistabilul B₂.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	*	*	*	*
		01	0	0	0	0
		11	0	1	*	*
		10	*	*	*	*

$$\Rightarrow J_2 = B_3 \cdot B_0$$

Fig. 4.4.2_12. Diagrama lui K pentru bistabilul B₂.

		B ₁ B ₀	B ₃ B ₂			
		00	01	11	10	
		00	0	0	0	0
		01	1	0	0	0
		11	1	1	*	*
		10	0	1	*	*

Fig. 4.4.2_13. Diagrama stărilor următoare pentru bistabilul B₃.

CAP. 4. DESPRE PROCESOR. UNITATEA ARITMETICO-LOGICĂ

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
			00	0	0	0	0
			01	1	0	0	0
			11	*	*	*	*
			10	*	*	*	*

$$\Rightarrow J_3 = B_2 \cdot \bar{B}_1 \cdot \bar{B}_0$$

Fig. 4.4.2_14. Diagrama lui J pentru bistabilul B₃.

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	01	11	10
			00	*	*	*	*
			01	*	*	*	*
			11	0	0	*	*
			10	1	0	*	*

$$\Rightarrow K_3 = \bar{B}_2 \cdot \bar{B}_1 \cdot \bar{B}_0$$

Fig. 4.4.2_15. Diagrama lui K pentru bistabilul B₃.

Așadar, ecuațiile intrărilor bistabilelor B₀ – B₃ vor fi:

- pentru bistabilul B₀:

$$J_0 = \bar{B}_3 \cdot \bar{B}_2 \cdot \bar{B}_1 \cdot DIV + B_2 \cdot B_1 \quad (4.4.2-2)$$

$$K_0 = \bar{B}_3 \cdot B_2 \cdot \bar{B}_1 + B_3 \cdot \bar{B}_2 \cdot \bar{B}_1 + \bar{B}_2 \cdot B_1 \quad (4.4.2-3)$$

- pentru bistabilul B₁:

$$J_1 = \bar{B}_3 \cdot \bar{B}_2 \cdot B_0 \quad (4.4.2-4)$$

$$K_1 = B_2 \cdot B_0 \quad (4.4.2-5)$$

- pentru bistabilul B₂:

$$J_2 = B_1 \cdot \bar{B}_0 \quad (4.4.2-6)$$

$$K_2 = B_3 \cdot B_0 \quad (4.4.2-7)$$

- pentru bistabilul B₃:

$$J_3 = B_2 \cdot \bar{B}_1 \cdot \bar{B}_0 \quad (4.4.2-8)$$

$$K_3 = \bar{B}_2 \cdot \bar{B}_1 \cdot \bar{B}_0 \quad (4.4.2-9)$$

A mai rămas să determinăm ecuațiile ieșirilor, ieșiri care sunt: IN, DS, PD, SC și AD. Întocmim, pentru fiecare dintre acestea, câte o “diagramă a ieșirii” –vezi figurile 4.4.2_16 - 4.4.2_20-.

B_1B_0	00	01	11	10	
B_3B_2	00	0	1	0	0
	01	0	0	0	0
	11	0	0	*	*
	10	0	0	*	*

$$\Rightarrow IN = \bar{B}_3 \cdot \bar{B}_2 \cdot \bar{B}_1 \cdot B_0$$

Fig. 4.4.2_16. Diagrama ieșirii IN.

B_1B_0	00	01	11	10	
B_3B_2	00	0	0	1	1
	01	0	1	0	0
	11	0	1	*	*
	10	0	0	*	*

$$\Rightarrow DS = B_2 \cdot \bar{B}_1 \cdot B_0 + \bar{B}_2 \cdot B_1$$

Fig. 4.4.2_17. Diagrama ieșirii DS.

B_1B_0	00	01	11	10	
B_3B_2	00	0	0	1	0
	01	0	0	0	0
	11	0	0	*	*
	10	0	0	*	*

$$\Rightarrow PD = \bar{B}_2 \cdot B_1 \cdot B_0$$

Fig. 4.4.2_18. Diagrama ieșirii PD.

B_1B_0	00	01	11	10	
B_3B_2	00	0	0	0	0
	01	1	0	0	1
	11	0	0	*	*
	10	0	1	*	*

$$\Rightarrow SC = \bar{B}_3 \cdot B_2 \cdot \bar{B}_0 + B_3 \cdot \bar{B}_2 \cdot B_0$$

Fig. 4.4.2_19. Diagrama ieșirii SC.

		B ₁ B ₀	B ₃ B ₂	00	01	11	10
		B ₁ B ₀	B ₃ B ₂	00	0	0	0
		B ₁ B ₀	B ₃ B ₂	01	0	1	0
		B ₁ B ₀	B ₃ B ₂	11	1	*	*
		B ₁ B ₀	B ₃ B ₂	10	1	*	*
				$\Rightarrow AD = B_3 \cdot \bar{B}_0 + B_2 \cdot B_1 \cdot B_0$			

Fig. 4.4.2_20. Diagrama ieșirii AD.

Așadar, ecuațiile ieșirilor IN, DS, PD, SC și AD vor fi:

$$IN = \bar{B}_3 \cdot \bar{B}_2 \cdot \bar{B}_1 \cdot B_0 \quad (4.4.2-10)$$

$$DS = B_2 \cdot \bar{B}_1 \cdot B_0 + \bar{B}_2 \cdot B_1 \quad (4.4.2-11)$$

$$PD = \bar{B}_2 \cdot B_1 \cdot B_0 \quad (4.4.2-12)$$

$$SC = \bar{B}_3 \cdot B_2 \cdot \bar{B}_0 + B_3 \cdot \bar{B}_2 \cdot B_0 \quad (4.4.2-13)$$

$$AD = B_3 \cdot \bar{B}_0 + B_2 \cdot B_1 \cdot B_0 \quad (4.4.2-14)$$

Pe baza ecuațiilor (4.4.2-2) - (4.4.2-14), se obține pentru blocul de comandă al dispozitivului de împărțire de numere în semn-mărime schema din figura 4.4.2_21.

În figura 4.4.2_22, se prezintă cronograma de funcționare a schemei din figura 4.4.2_21.

CAP. 4. DESPRE PROCESOR. UNITATEA ARITMETICO-LOGICĂ

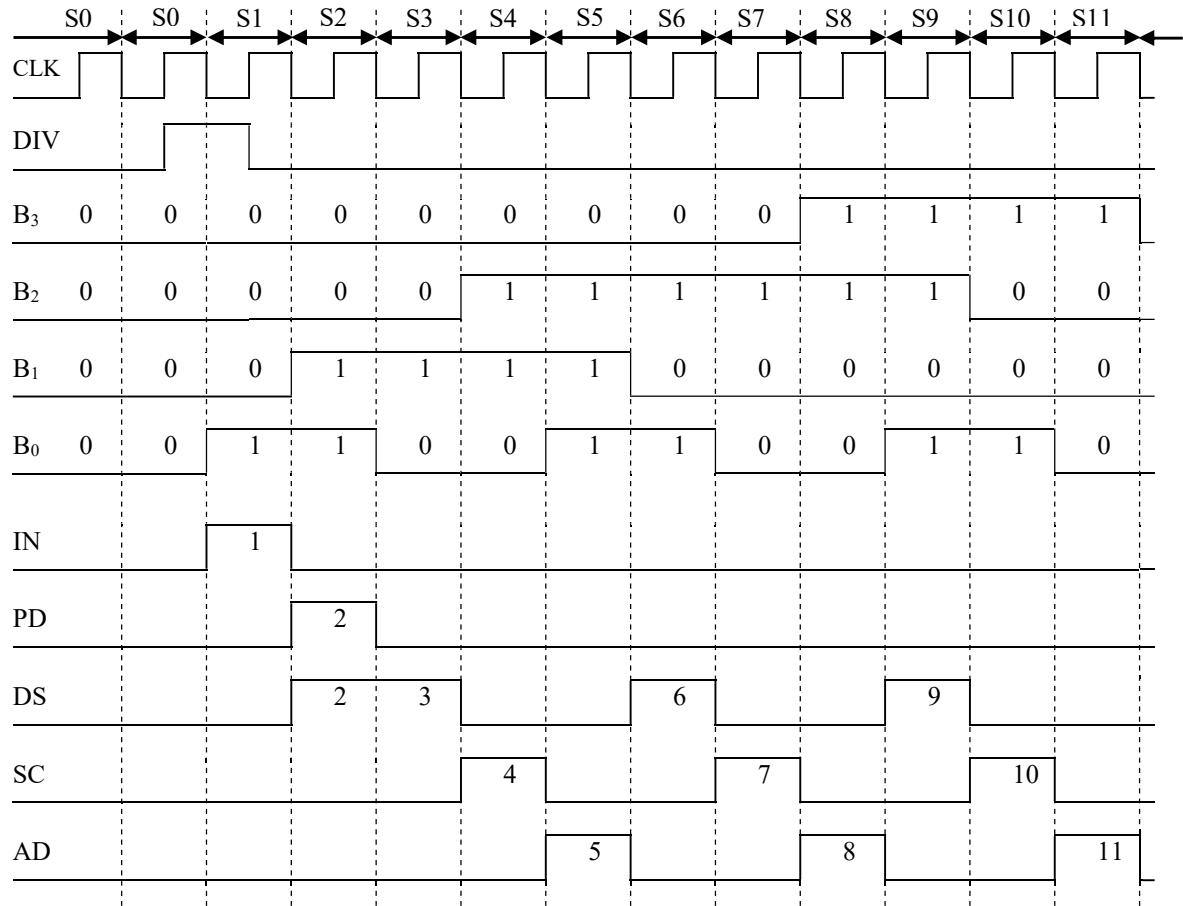
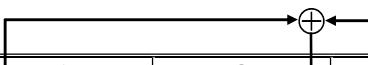


Fig. 4.4.2_22. Cronograma de funcționare a blocului de comandă al dispozitivului de împărțire cu refacerea resturilor.

Observație:

Acțiunile semnalelor de comandă **IN**, **PD**, **DS**, **SC** și **AD** se produc **efectiv** pe frontul ridicător (posibil, însă, și pe cel coborâtor) al semnalului de tact din ciclul în care ele sunt active.

În final, se prezintă, în tabelul care face obiectul figurii 4.4.2_23, un exemplu de împărțire cu refacerea resturilor.



Starea	$B_3B_2B_1B_0$	A	Q	M	Operăția
0	0000	xxxx	xxxx	xxxx	START \Rightarrow linia 1
1	0001	xxxx	xxxx	xxxx	IN \Rightarrow linia 2
2	0011	1000	1110	0011	DS, PD \Rightarrow linia 3
3	0010	0001	1101	0011	DS \Rightarrow linia 4
4	0110	0011 - 0011	1011	0011	SC \Rightarrow linia 5
5	0111	0000 + 0000	1011	0011	AD \Rightarrow linia 6
6	0101	0000	1011	0011	DS \Rightarrow linia 7
7	0100	0001 - 0011	0111	0011	SC \Rightarrow linia 8
8	1100	110 + 0011	0111	0011	AD \Rightarrow linia 9
9	1101	0001	0110	0011	DS \Rightarrow linia 10
10	1001	0010 - 0011	1101	0011	SC \Rightarrow linia 11
11	1000	111 + 0011	1101	0011	AD \Rightarrow linia 12
0	0000	0010	1100	0011	NIMIC
0	0000	0010	1100	0011	NIMIC

Fig. 4.4.2_23. Exemplu de împărțire cu refacerea resturilor: $x = -14$, $y = 3$.

CAP. 4. DESPRE PROCESOR. UNITATEA ARITMETICO-LOGICĂ

5

DESPRE PROCESOR. UNITATEA DE REGISTRE

5.1. Introducere

Așa cum s-a arătat în capitolul 1, **unitatea de registre** reprezintă o **memorie de manevră**, realizată pe același cip și în aceeași tehnologie cu **unitatea aritmetico-logică** și cu **unitatea de comandă** -și, în consecință, la fel de rapidă ca și acestea-, la care accesele se fac mult mai simplu și mai eficient decât la memoria propriu-zisă.

Două linii arhitecturale s-au dezvoltat la nivelul unităților de registre. Una dintre ele se caracterizează prin **heterogenitate**, iar cealaltă -prin **omogenitate**.

În cazul arhitecturilor heterogene, registrele se diferențiază sub aspectul rolului pe care îl pot juca în funcționarea procesorului și, în consecință, al modului în care intervin sau pot interveni în diversele instrucții-mașină.

În cazul arhitecturilor omogene, registrele sunt echivalente și, în consecință, oricare dintre ele ar putea juca, la un moment dat, un anumit rol în cadrul unei anumite instrucții. Trebuie spus că, în practică, nici heterogenitatea și nici omogenitatea nu au fost duse la cotele maximale. În arhitecturile heterogene, este obișnuit să se întâlnească un număr de registre echivalente între ele sau diferențiate doar în mică măsură, după cum este obișnuit ca în arhitecturile omogene un număr de registre să disponă și de caracteristici în plus față de cele comune tuturor sau să se diferențieze ușor unele de altele.

În scopuri ilustrative, se prezintă, în continuare, unitatea de registre a unui procesor imaginär, cu **date pe 8 biți** și **adrese pe 16 biți** –fie *PD* numele lui-, pe care îl vom considera, pe mai departe în cadrul cursului, drept principal caz de studiu.

5.2. Unitatea de registre a procesorului PD

Unitatea de registre a procesorului *PD* cuprinde registrele reprezentate prin dreptunghiuri colorate în gri în figura 5.2_1, figură în care, în sprijinul unei cât mai bune înțelegeri a lucrurilor, a fost redat întregul procesor. Respectivele registre sunt: *A*, *B*, *C*, *F*, *X*, *SP*, *B1*, *B2* și *B3*. Le vom prezenta rând pe rând.

Registrele *A*, *B*, și *C*

Registrele *A*, *B* și *C* au o lungime de **8 biți** și sunt **echipotente** între ele. Principala lor menire este aceea de a juca, **în procesul de execuție a instrucțiilor**, ca **alternativă la locațiile de memorie**, rolul de reședință a unui **operand sau chiar a ambilor** (în arhitecturile pe care le anvizajăm, aşa cum se va vedea, o instrucție poate avea maximum doi operanzi), respectiv de a **prelupa rezultatele obținute**. Este ușual –de fapt, e regulă– ca rezultatele execuției instrucțiilor să fie oferite de procesoare prin locația de memorie sau registrul inițial pe post de sediu al operandului unu –eventual unic-. Evident, respectivul operand, prin suprascriere, se pierde. Atunci când valoarea sa este și pe mai departe necesară în rularea programului, ea trebuie salvată, fie în memorie, fie într-un registru. Evident, a doua alternativă ar fi cea de preferat, întotdeauna, dar, uneori, ea nu poate fi aplicată, deoarece se întâmplă ca nici unul dintre registre să nu fie la momentul în cauză liber.

Registrul *F*

Registrul *F* are rolul de a memora în bistabilele sale informații privind rezultatele obținute în cadrul operațiilor efectuate de unitatea aritmeticologică. La unele procesoare –inclusiv la *PD* se au în vedere doar rezultatele operațiilor care implică o prelucrare efectivă (adunare, scădere, **ȘI**, **SAU** etc.), iar la altele, inclusiv operațiile de transfer simplu dintr-un loc în altul. Bistabilele registrului *F* poartă numele de fanioane. Ele sunt notate cu *Z*, *S*, *C*, *V* și *P*.

Fanionul *Z* arată dacă rezultatul ultimei operații este nul sau nenul. Când rezultatul este nul, atunci *Z=1*, când rezultatul este nenul, atunci *Z=0*.

CAP. 5. DESPRE PROCESOR. UNITATEA DE REGISTRE

Fanionul S arată dacă rezultatul ultimei operații este negativ sau nenegativ (adică: pozitiv sau zero). Când rezultatul este negativ, atunci $S=1$, când rezultatul este nenegativ, atunci $S=0$.

Fanionul C arată dacă în timpul ultimei operații a apărut sau nu un transport din rangul cel mai semnificativ, 2^7 , spre rangul inexistent, 2^8 . Când a apărut transport, atunci $C=1$, când nu a apărut transport, atunci $C=0$.

Fanionul V arată dacă rezultatul ultimei operații este sau nu eronat ca urmare a fenomenului de depășire. Când rezultatul este eronat, atunci $V=1$, când rezultatul este neeronat, atunci $V=0$.

Fanionul P arată dacă rezultatul ultimei operații are un număr par sau un număr impar de biți cu valoarea “1”. Când rezultatul are un număr par de biți cu valoarea “1”, atunci $P=1$, când rezultatul are un număr impar de biți cu valoarea “1”, atunci $P=0$.

Toate aceste fanioane se poziționează implicit în procesul de execuție a instrucțiilor. Există, însă, și instrucții speciale, a căror obiect este exclusiv poziționarea fanioanelor.

Registrul X

Registrul X este constituit prin concatenarea registrelor XH și XL (H și L vin, respectiv, de la “*high*” și “*low*”). Prin urmare, X este un regisztr pe 16 biți. Rolul regisztrului X este acela de a fi unul dintre furnizorii de adrese de memorie, alături SP , PC și IRE , după cum schema din fig. 5.2.1 ne sugerează. Evident, regisztrul X comunică cu magistrala de date prin jumătățile sale, XH și XL . La fel, SP , PC și IRE comunică cu magistrala de date prin jumătățile lor, SPH , SPL , PCH , PCL , $IREH$, $IREL$, respectiv, toate aceste registre întregi fiind pe 16 biți, în timp ce magistrala de date este pe 8 biți.

Registrul SP

Registrul SP (SP vine de la “*stack pointer*”) este constituit prin concatenarea registrelor SPH și SPL (H și L vin, respectiv, de la “*high*” și “*low*”). Prin urmare, SP este un regisztr pe 16 biți. Rolul regisztrului SP este acela de a face posibilă gestionarea unei părți din memoria RAM a

calculatorului după principiul stivei: *last input-first output* (prescurtat: *LIFO*); respectiva parte va fi numită simplu “stivă”. În procesorul *PD*, fiecare sesiune de lucru la nivelul stivei manevrează doi octeți. Aceștia pot proveni din registrele *A* și *F* sau *B* și *C* sau *XH* și sau *PCH* și *PCL*, în cazul în care se face scriere în stivă, respectiv pot viza registrele *A* și *F* sau *B* și *C* sau *XH* și *XL* sau *PCH* și *PCL*, în cazul în care se face citire din stivă. Tandemurile *A-F*, *B-C*, *XH-XL*, *PCH-PCL* vor fi referite prin abrevierile *AF*, respectiv *BC*, respectiv *X* respectiv *PC*.

Exploatarea unei zone de memorie după principiul stivei presupune, mai întâi, **încărcarea registrului *SP* cu adresa cea mai de sus a respectivei zone**, iar apoi respectarea cu strictețe a următoarei proceduri:

A. la scriere:

1. se **decrementează** *SP* cu o unitate
2. se scrie la adresa din *SP* **octetul superior** din perechea de doi octeți ce trebuie depusă în stivă
3. se **decrementează** *SP* cu o unitate
4. se scrie la adresa din *SP* **octetul inferior** din perechea de doi octeți ce trebuie depusă în stivă

B. la citire:

1. se citește de la adresa din *SP* **octetul inferior** din perechea de doi octeți ce trebuie descărcată din stivă
2. se **incrementează** *SP* cu o unitate
3. se citește de la adresa din *SP* **octetul superior** din perechea de doi octeți ce trebuie descărcată din stivă
4. se **incrementează** *SP* cu o unitate

Registrele *B1*, *B2* și *B3*

Registrele *B1*, *B2* și *B3* sunt registre auxiliare, de uz exclusiv intern. *B1* joacă rol de tampon (“*buffer*”) între magistrala pe care se vehiculează datele și unitatea aritmetico-logică. *B2* joacă rol de tampon între unitatea aritmetico-logică și magistrala pe care se vehiculează datele. *B3* joacă rol de tampon între selectorul de adrese, *AS*, și magistrala pe care se transmit în afară, spre memorie și porturi, adresele. *B2* și *B3* au ieșirile cu trei stări logice. Activarea respectivelor ieșiri se face cu ajutorul semnalelor *OB_B2*, respectiv *OB_B3*, care înseamnă: “**on the bus B2**”, respectiv “**on the bus B3**”.

CAP. 5. DESPRE PROCESOR. UNITATEA DE REGISTRE

6

DESPRE PROCESOR. UNITATEA DE COMANDĂ

6.1. Preliminarii. Paradigma von Neumann

După cum s-a văzut, nucleul unui calculator este constituit din **procesor, memorie și interfețe**, la nivelul procesorului distingându-se: unitatea aritmetico-logică, unitatea de registre și unitatea de comandă.

Unitatea aritmetico-logică și algoritmii pe baza cărora ea funcționează s-au studiat în detaliu în capitolul 4.

Unitatea de registre, cu locul și rolul elementelor sale în ansamblul procesorului, a făcut obiectul capitolului 5.

Despre *unitatea de comandă* s-a spus în prima parte a lucrării că ea reprezintă partea unui procesor care are în sarcină generarea comenziilor interne și externe necesare funcționării calculatorului în ansamblul său.

Există mai multe tipuri de unități de comandă; în cadrul prezentei lucrări, se vor avea în vedere doar unitățile de comandă numite, ca și procesoarele în care sunt integrate, *von Neumann*. Acestea se caracterizează prin aceea că respectă următoarea paradigmă, numită “paradigma *von Neumann*”:

- 1). Un program constă în instrucții care operează la un moment asupra unui singur flux de date.
- 2). Instrucțiile se execută una după alta și nu mai multe simultan, sub aspectul producerii de rezultate; la nivel micro, însă, este posibil ca operații elementare aparținând de două sau mai multe instrucții să fie executate suprapus, fără alterarea ordinii de generare de rezultate macro, față de cazul în care instrucțiile s-ar executa pur secvențiat.
- 3). O instrucție poate implica introducerea în procesare a maximum doi operanzi.
- 4). Instrucțiile se reprezintă prin coduri care sunt fie de lungime fixă, fie de lungime variabilă.

- 5). Un cod de instrucție cuprinde câmpuri prin care se specifică:
- operația ce face obiectul instrucției
 - operandul / operanții sau informații despre locul în care se află operandul / operanții
 - informații despre locul unde trebuie depus rezultatul
- 6). O instrucție este urmată în execuție, în mod implicit, de instrucția situată în memorie imediat după ea. În cazul instrucțiilor de salt, de apel de subrutină și de revenire din subrutină, instrucția următoare se determină de către fiecare dintre ele în mod specific, prin algoritmul înglobat special în cadrul lor, în acest sens.
- 7). Programele aflate în rulare sunt “încărcate”, adică: au instrucțiile amplasate în memorie, în acord cu logica programului, în locații de adrese consecutiv crescătoare sau rezultate din logica internă a instrucțiilor de salt, de apel de subrutină și de revenire din subrutină.

6.2. Elementele constitutive ale unităților de comandă von Neumann

O unitate de comandă von Neumann constă, în esență, din următoarele blocuri:

- un regiszru de adresare a instrucțiilor, numit *pointer de instrucție*, **IP** (*Instruction Pointer*) sau *numărător al programului*, **PC** (*Program Counter*);
- un regiszru sediu al cuvântului instrucție ce instanțiază instrucția aflată în rulare, numit *regisztrul instrucției*, **IR** (*Instruction Register*);
- un decodificator al cuvântului instrucție ce instanțiază instrucția aflată în rulare, numit, din rațiuni istorice, *decodificator al codului operației*, **OCD** (*Operation Code Decoder*);
- un bloc de secvențiere a comenziilor, numit *secvențiator*, **S** (*Sequencer*);
- un bloc de pilotare temporală, numit *generator de tact*, **CG** (*Clock Generator*);
- un bloc de inițializare, numit *generator de reset*, **RG** (*Reset Generator*).

Figura 6.2_1 surprinde o unitate de comandă von Neumann, ca parte a unui procesor ipotetic, definit de pe poziții didactice –procesorul *PD*, referit și în capitolul precedent.

CAP. 6. DESPRE PROCESOR. UNITATEA DE COMANDĂ

Se remarcă prezența în figura 6.2_1 a tuturor celor 6 blocuri din compoñenþa unităþii de comandă standard *von Neumann*. Să ne oprim puþin, în continuare, asupra fiecăruiu dintre ele.

Registrul PC

Registrul **PC** -*numărătorul programului* sau *pointerul de instrucþie*- are rolul de a indica adresa următoarei instrucþii sau, când instrucþiile se citesc fragmentar, adresa următorului fragment al instrucþiei în curs de rulare.

Având în vedere paradigma *von Neumann*, registrul **PC** funcþionează, de regulă, în regim de numărare –mai exact: incrementare- și doar în cazul instrucþiilor de salt, apel de subrutină și revenire din subrutină, în regim de încărcare paralelă; aceasta, în ideea de a prelua ceea ce se generează în cadrul acestor instrucþii, pe post de adresă a instrucþiei următoare.

Registrul IR

Registrul **IR** –*registrul instrucþiei*- joacă rolul de sediu al codului operaþiei care face obiectul instrucþiei. Este obiþnuit ca registrul **IR** să aibă extensiile pentru păstrarea unor informaþii conexe codului operaþiei, cum ar fi informaþiile privind locul în care se află operanþii sau unde trebuie depus rezultatul, respectiv informaþii pe baza cărora se determină –eventual în urma unor calcule- locul operanþilor și-sau al rezultatului.

În arhitectura considerată, extensiile registrului **IR** sunt **IREL**, respectiv **IREH**. Acestea sunt registre pe câte 8 biþi, concatenabile între ele astfel încât să poată prelua și păstra și informaþii pe 16 biþi.

Aşa cum se va vedea, în arhitectura considerată instrucţiile au lungimi de 1-4 octeţi. În timpul procesului de rulare, aceştia sunt păstraţi la nivelul procesorului după cum urmează:

- octetul 1: în registrul **IR**
- octetul 2: în registrul **IREL**
- octetul 3: în registrul **IREH**
- octetul 4: în registrul **B1**

A nu se deduce din cele de mai sus că registrul **B1** ar fi şi el extensie a registrului **IR**. Registrul **B1** are, după cum s-a arătat în capitolul precedent şi după cum se va vedea în continuare, cu totul alte roluri şi, numai conjunctural, în cazul unui număr redus de instrucţii, el este pe post de extensie a registrului **IR**.

Decodificatorul codului operaţiei

Decodificatorul codului operaţiei are rolul de a decodifica codul din **IR** şi de a indica secvenţiatorului **S**, pe de o parte, operaţia ce trebuie executată, iar pe de altă parte, operandul / operanzzii sau modul în care trebuie să fie determinat locul operandului / operanzilor şi-sau al rezultatului.

Se precizează că există o multitudine de modalităţi prin care operandul / operanzzii, respectiv locul operandului / operanzilor şi-sau locul rezultatului se pot specifica; o astfel de modalitate poartă, consacrat, numele de “**mod de adresare**”. În figura 6.2_2, sunt sintetizate modurile de adresare cu care este înzestrată arhitectura de calcul considerată.

Se face precizarea că unele dintre modurile de adresare surprinse în figura 6.2_2 sunt referite de către unii producători şi-sau utilizatori de procesoare prin alți termeni decât cei adoptaţi în context de noi; de altfel, cititorul este rugat să rețină că unicitatea termenilor şi a semnificaţiilor pe care termenii le au este, în domeniul calculatoarelor, mult prea frecvent, doar un deziderat.

CAP. 6. DESPRE PROCESOR. UNITATEA DE COMANDĂ

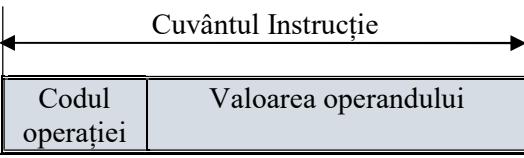
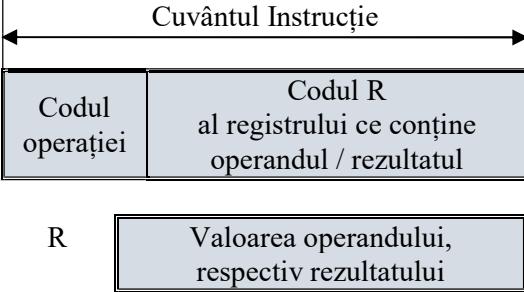
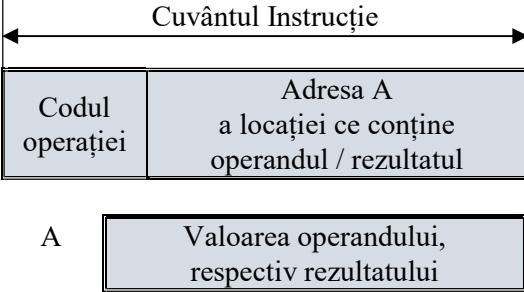
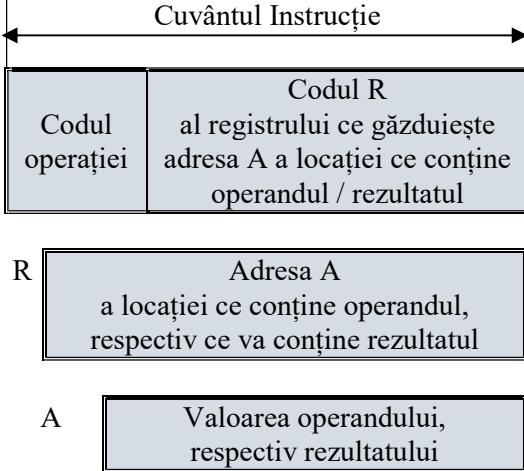
Nr. crt.	Modul de adresare	Definirea modului de adresare
1.	Adresarea imediată 	Se numește adresare imediată specificarea unui operand, O, prin cuprinderea valorii sale efective în însuși cuvântul instrucție, prin biți dedicați , alăturați celor prin care se specifică codul operației.
2.	Adresarea de registru 	Se numește adresare de registru specificarea unui operand în cazul găzduirii sale într-un registru sau a registrului vizat să găzduiască un rezultat, prin indicarea codului R al registrului în cauză, prin biți dedicați din cuvântul instrucție, alăturați celor prin care se specifică codul operației.
3.	Adresarea directă 	Se numește adresare directă specificarea unui operand în cazul găzduirii sale într-o locație de memorie sau a locației de memorie vizată să găzduiască un rezultat prin indicarea adresei A a locației în cauză, prin biți dedicați din cuvântul instrucție, alăturați celor prin care se specifică codul operației.
4.	Adresarea indirectă 	Se numește adresare indirectă specificarea unui operand în cazul găzduirii sale într-o locație de memorie sau a locației de memorie vizată să găzduiască un rezultat prin indicarea adresei A a locației în cauză prin intermediul unui registrator , al cărui cod R este precizat prin biți dedicați din cuvântul instrucție, alăturați celor prin care se specifică codul operației.

Fig. 6.2_2. Modurile de adresare ale arhitecturii de calcul din figura 6.2_1
(se continuă).

CAP. 6. DESPRE PROCESOR. UNITATEA DE COMANDĂ

5. Adresarea indexată	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;">Codul operației</td><td style="width: 25%; padding: 5px;">Codul R al registrului ce conține indexul I ce se adună la adresa de referință A pentru obținerea adresei locației ce conține operandul / rezultatul</td><td style="width: 50%; padding: 5px;">Adresa de referință A la care se adună indexul I pentru obținerea adresei locației ce conține operandul / rezultatul</td></tr> </table> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%; text-align: right; padding-right: 5px;">R</td><td style="width: 90%; background-color: #e0e0e0; padding: 5px; text-align: center;">Indexul I</td></tr> </table> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%; text-align: right; padding-right: 5px;">A + I</td><td style="width: 90%; background-color: #e0e0e0; padding: 5px; text-align: center;">Valoarea operandului, respectiv rezultatului</td></tr> </table>	Codul operației	Codul R al registrului ce conține indexul I ce se adună la adresa de referință A pentru obținerea adresei locației ce conține operandul / rezultatul	Adresa de referință A la care se adună indexul I pentru obținerea adresei locației ce conține operandul / rezultatul	R	Indexul I	A + I	Valoarea operandului, respectiv rezultatului	<p>Se numește adresare indexată specificarea unui operand în cazul găzduirii sale într-o locație de memorie sau a locației de memorie vizată să găzduiască un rezultat prin indicarea adresei locației în cauză prin două componente aditive ale sale:</p> <ul style="list-style-type: none"> ➤ una reprezentând o adresă de referință numită adresa de bază, B, dată ca și adresa din cazul adresării indirecte, deci, prin intermediul unui registru, al cărui cod R este precizat prin biți dedicați din cuvântul instrucție, alăturați celor prin care se specifică codul operației ➤ una reprezentând un deplasament, D, față de adresa de referință, dată ca și adresa din cazul adresării directe, deci, prin biți dedicați din cuvântul instrucție, alăturați celor prin care se specifică codul operației.
Codul operației	Codul R al registrului ce conține indexul I ce se adună la adresa de referință A pentru obținerea adresei locației ce conține operandul / rezultatul	Adresa de referință A la care se adună indexul I pentru obținerea adresei locației ce conține operandul / rezultatul							
R	Indexul I								
A + I	Valoarea operandului, respectiv rezultatului								
6. Adresarea relativă sau adresarea cu bază	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; padding: 5px;">Codul operației</td><td style="width: 25%; padding: 5px;">Codul R al registrului ce conține adresa de bază B la care se adună deplasamentul D pentru obținerea adresei locației ce conține operandul / rezultatul</td><td style="width: 50%; padding: 5px;">Deplasamentul D, care se adună la adresa de bază B din registru de cod R, pentru obținerea adresei locației ce conține operandul / rezultatul</td></tr> </table> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%; text-align: right; padding-right: 5px;">R</td><td style="width: 90%; background-color: #e0e0e0; padding: 5px; text-align: center;">Adresa de bază B</td></tr> </table> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%; text-align: right; padding-right: 5px;">B + D</td><td style="width: 90%; background-color: #e0e0e0; padding: 5px; text-align: center;">Valoarea operandului, respectiv rezultatului</td></tr> </table>	Codul operației	Codul R al registrului ce conține adresa de bază B la care se adună deplasamentul D pentru obținerea adresei locației ce conține operandul / rezultatul	Deplasamentul D, care se adună la adresa de bază B din registru de cod R, pentru obținerea adresei locației ce conține operandul / rezultatul	R	Adresa de bază B	B + D	Valoarea operandului, respectiv rezultatului	<p>Se numește adresare relativă sau adresare cu bază specificarea unui operand în cazul găzduirii sale într-o locație de memorie sau a locației de memorie vizată să găzduiască un rezultat prin indicarea adresei locației în cauză prin două componente aditive ale sale:</p> <ul style="list-style-type: none"> ➤ una reprezentând o adresă de referință numită adresa de bază, B, dată ca și adresa din cazul adresării indirecte, deci, prin intermediul unui registru, al cărui cod R este precizat prin biți dedicați din cuvântul instrucție, alăturați celor prin care se specifică codul operației ➤ una reprezentând un deplasament, D, față de adresa de referință, dată ca și adresa din cazul adresării directe, deci, prin biți dedicați din cuvântul instrucție, alăturați celor prin care se specifică codul operației.
Codul operației	Codul R al registrului ce conține adresa de bază B la care se adună deplasamentul D pentru obținerea adresei locației ce conține operandul / rezultatul	Deplasamentul D, care se adună la adresa de bază B din registru de cod R, pentru obținerea adresei locației ce conține operandul / rezultatul							
R	Adresa de bază B								
B + D	Valoarea operandului, respectiv rezultatului								

Fig. 6.2_2. Modurile de adresare ale arhitecturii de calcul din figura 6.2_1
(se continuă).

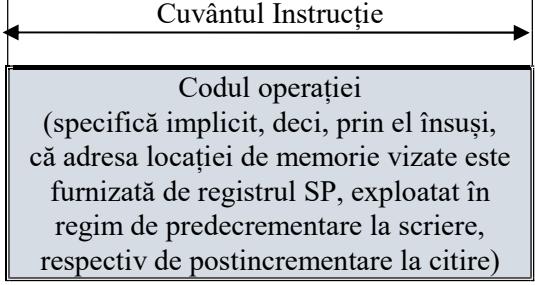
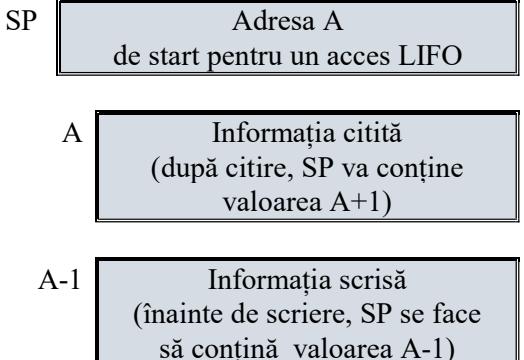
7. Adresarea LIFO 	<p>Se numește adresare LIFO referirea unor locații de memorie pentru depunerea în ele a conținuturilor unor registre, respectiv pentru descărcarea conținuturilor lor în anumite registre, cu ajutorul adresei conținută de registrul pointer de stivă, decrementat înaintea fiecărei operații de scriere și incrementat după fiecare operație de citire, deci funcționând în disciplina Last In First Out (LIFO).</p>
	

Fig. 6.2_2. Modurile de adresare ale arhitecturii de calcul din figura 6.2_1.
(continuare)

Secvențiatorul

Secvențiatorul S are rolul de a genera ansamblul semnalelor ce materializează operațiile elementare la care sunt reduse instrucțiile, în mod specific pentru fiecare instrucție, conform indicațiilor primite de la decodificatorul codului operației.

Rularea unei instrucții are două etape:

- 1). **Etapa de extragere sau aducere (“fetch”)** a codului operației și a informațiilor privind locul operanzilor și-sau rezultatului indisociabile de codul operației.
- 2). **Etapa de execuție** propriu-zisă a instrucției.

Etapa de aducere cuprinde un ciclu de citire din memorie identic pentru toate instrucțiile. În cadrul acestui ciclu, se citește locația cu adresa prezentă în registrul **PC** și conținutul ei se duce în registrul **IR**.

Etapa de execuție cuprinde unul sau mai multe cicluri de citire, respectiv de scriere, vizând memoria sau un port, respectiv operații pur interne procesorului, specifice pentru fiecare instrucție. Aceste cicluri au ca obiectiv, în principal, citirea operanzilor, respectiv scrierea rezultatului, dar, uneori, mai înainte de asta, aducerea informațiilor privind locul operanzilor și-sau rezultatului, reprezentând entități din cuvântul instrucție alăturate codului operației, dar separate de cea care-l conține pe acesta –vezi paragraful cu modurile de adresare-.

Semnalele de comandă generate de sevențiator acționează, în majoritatea covârșitoare a lor, în interiorul procesorului și doar un număr redus în afara sa. În arhitectura considerată, semnale de comandă interne sunt:

- **LD_IR**
- **OB_A**
- **ADD**
- etc.,

iar semnale de comandă externe:

- **MRQ**
- **IORQ**
- **RD**
- **WR** .

Semnalul **LD_IR (load IR)** are rolul de a determina încărcarea registrului **IR** cu informația prezentă la intrările sale. Este activ pe frontul ridicător.

Semnalul **OB_A (on the bus A)** are rolul de a determina punerea pe liniile de date ale procesorului a conținutului registrului **A**, prin activarea ieșirilor cu 3 stări ale acestui regisztr. Este activ pe nivelul ridicat, adică: pe nivelul de “1”.

Semnalul ADD (***add***) are rolul de a specifica unității aritmetico-logice că are de efectuat o adunare. Este activ pe nivelul ridicat, adică: pe nivelul de “1”.

Semnalul \overline{MRQ} (“***memory request***”) are rolul de a mobiliza memoria în vederea efectuării unei operații. Operația poate fi de citire sau de scriere, dependent de valoarea logică a semnalelor \overline{RD} și \overline{WR} . Este activ pe nivelul coborât, adică: pe nivelul de “0”.

Semnalul \overline{IORQ} (“***input-output request***”) are rolul de a mobiliza unitatea de porturi în vederea efectuării unei operații. Operația poate fi de citire sau de scriere, dependent de valoarea logică a semnalelor \overline{RD} și \overline{WR} . Este activ pe nivelul coborât, adică: pe nivelul de “0”.

Semnalul \overline{RD} (“***read***”) are rolul de a specifica memoriei sau unității de porturi că operația dispusă de procesor prin semnalele \overline{MRQ} , respectiv \overline{IORQ} este una de citire. Este activ pe nivelul coborât, adică: pe nivelul de “0”.

Semnalul \overline{WR} (“***write***”) are rolul de a specifica memoriei sau unității de porturi că operația dispusă de procesor prin semnalele \overline{MRQ} , respectiv \overline{IORQ} este una de scriere. Este activ pe nivelul coborât, adică: pe nivelul de “0”.

Operația de citire / scriere în derulare la un moment va viza o locație de memorie sau alta, respectiv un port sau altul, în funcție de valoarea informației numită *adresă*, transmisă memoriei, respectiv unității de porturi, prin grija unității de comandă, în timp ce semnalele \overline{MRQ} , respectiv \overline{IORQ} sunt active.

Suportul fizic pe care se face vehicularea adreselor între procesor și memorie, respectiv porturi este așa-numita “*magistrală de adrese*”, formată, în cazul arhitecturii considerate, din 16 linii: *A0-A15*. Liniile magistralei de adrese sunt cu trei stări logice, unidirectionale.

Suportul fizic pe care se face vehicularea datelor între procesor și memorie, respectiv porturi, ca și între memorie, respectiv porturi și procesor este așa-numita “*magistrală de date*”, formată, în cazul arhitecturii considerate, din 8 linii: *D0-D7*. Liniile magistralei de date sunt cu trei stări logice, bidirectionale.

Generatorul de tact

Generatorul de tact, **CG**, are rolul de a furniza așa-numitul “*semnal de tact*” sau “*semnal de ceas*”, *CLK* (“*clock*”), cu care își sincronizează funcționarea secvențiatorul, dar și celelalte blocuri secvențiale ale unității de comandă și, în ultimă instanță, întregul calculator.

Generatorul de reset

Generatorul de reset, **RG**, are rolul de a furniza un semnal de inițializare a secvențiatorului și a altor blocuri cu caracter secvențial, de fiecare dată când calculatorul se pune sub tensiune, respectiv când se apasă un buton dedicat, numit “*buton RESET*”.

Minimal, generatorul de reset asigură aducerea registrului **PC** la un conținut predefinit (zero, în cazul celor mai multe procesoare, inclusiv în cazul în studiu), reprezentând adresa instrucției de început a primului program ce se impune a fi rulat (de regulă: un program de autotestare sau un program încărcațor), respectiv instituirea stării inițiale a secvențiatorului, astfel încât el să-și demareze activitatea cu un ciclu *fetch*.

6.3. Instrucțiile procesoarelor von Neumann. Studiu de caz pe setul de instrucții al procesorului PD

6.3.1. Aspecte introductive

Procesorul *PD* dispune, la nivel generic, de 17 instrucții:

1. Instrucția de copiere (*copy*), “**COPY**”
2. Instrucția de adunare (*add*), “**ADD**”
3. Instrucția de scădere (*subtract*), “**SUB**”
4. Instrucția de efectuare a operației ȘI (*and*), “**AND**”
5. Instrucția de efectuare a operației SAU (*or*), “**OR**”
6. Instrucția de efectuare a operației SAU EXCLUSIV (*exclusiv or*), “**XOR**”
7. Instrucția de incrementare (*increment*), “**INC**”
8. Instrucția de decrementare (*decrement*), “**DEC**”
9. Instrucția de deplasare la dreapta (*shift right*), “**SHR**”
10. Instrucția de deplasare la stânga (*shift left*), “**SHL**”
11. Instrucția de intrare (*input*), “**IN**”
12. Instrucția de ieșire (*output*), “**OUT**”
13. Instrucția de scriere în stivă (*push*), “**PUSH**”
14. Instrucția de citire din stivă (*pop*), “**POP**”
15. Instrucția de salt (*jump*), “**JUMP**”
16. Instrucția de apel de subrutină (*call*), “**CALL**”
17. Instrucția de revenire din subrutină (*return*), “**RET**”

Înainte de prezentarea efectivă a acestor instrucții, se instituie următoarele notații:

- *op*: pentru desemnarea operației cu care se identifică o instrucție
- *x*: pentru desemnarea sediului operandului 1 sau a sediului comun al operandului 1 și al rezultatului sau a sediului operandului 2, distincția făcându-se din context; operandul 1 poate fi și unicul
- *y*: pentru desemnarea operandului 2 sau a sediului operandului 2, distincția făcându-se din context
- *A, B, C*: pentru desemnarea regisrelor A, B, C sau a conținuturilor lor, distincția făcându-se din context
- *@X*: pentru desemnarea locației de memorie pointată de adresa din registrul X sau a conținutului ei, distincția făcându-se din context
- *@aaaa*: pentru desemnarea locației de memorie pointată de adresa “aaaa” sau a conținutului ei, distincția făcându-se din context
- *@aa*: pentru desemnarea portului de adresă “aa” sau a conținutului lui, distincția făcându-se din context
- *aa*: pentru desemnarea unei valori pe 8 biți
- *aaaa*: pentru desemnarea unei valori pe 16 biți

Se face precizarea că, pentru simplitate, instrucțiile procesorului *PD* nu fac uz de toate cele șapte moduri de adresare introduse cu ocazia prezentării decodificatorului codului operației:

- adresarea imediată
- adresarea de registru
- adresarea directă
- adresarea indirectă
- adresarea indexată
- adresarea cu bază
- adresarea LIFO

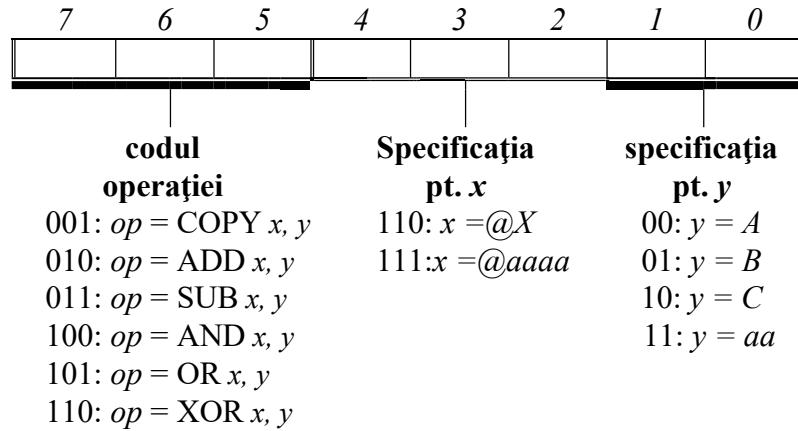
6.3.2. Codificarea instrucțiilor

Se convine ca octetul 1 al instrucțiilor –reamintim: instrucțiile procesorului *PD* se reprezintă pe 1...4 octeți- să aibă următoarele câmpuri:

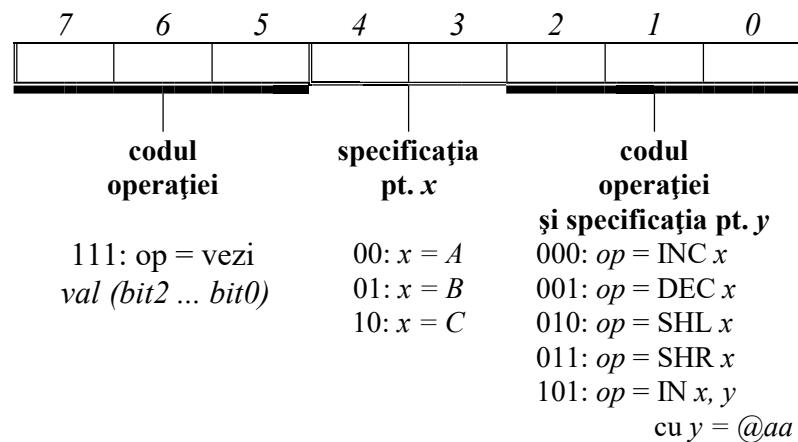
a). când $val (bit7 \dots bit5) \in [001_2 \dots 110_2]$ și $val (bit4 \dots bit3) \in [00_2 \dots 10_2]$:

7	6	5	4	3	2	1	0
codul operației			specificația pt. x			specificația pt. y	
001: $op = \text{COPY } x, y$			00: $x = A$			000: $y = A$	
010: $op = \text{ADD } x, y$			01: $x = B$			001: $y = B$	
011: $op = \text{SUB } x, y$			10: $x = C$			010: $y = C$	
100: $op = \text{AND } x, y$						011: $y = aa$	
101: $op = \text{OR } x, y$						100: $y = @X$	
110: $op = \text{XOR } x, y$						101: $y = @aaaa$	

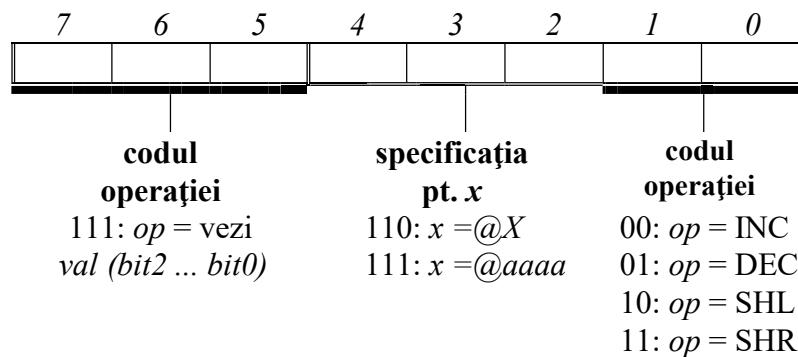
b). când $val (bit7 \dots bit5) \in [001_2 \dots 110_2]$ și $val (bit4 \ bit3) = 11_2$:



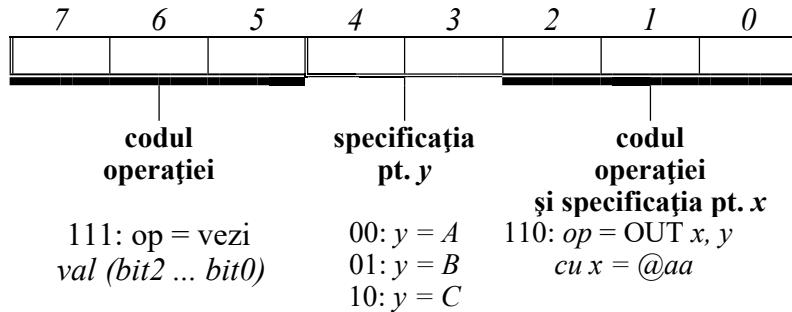
c). când $val (bit7 \dots bit5) = 111_2$ și $val (bit4 \ bit3) \in [00_2 \dots 10_2]$
și $val (bit2 \dots bit0) \in [000_2 \dots 101_2]$:



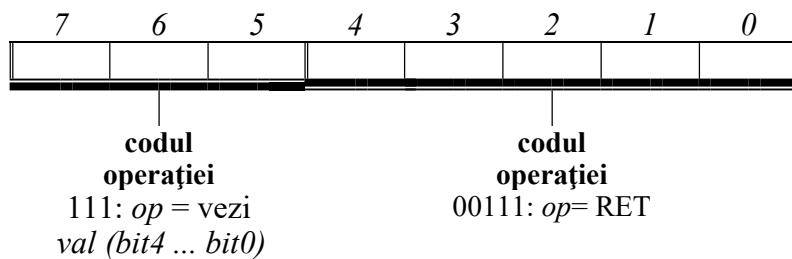
d). când $val (bit7 \dots bit5) = 111_2$ și $val (bit4 \ bit3) = 11_2$



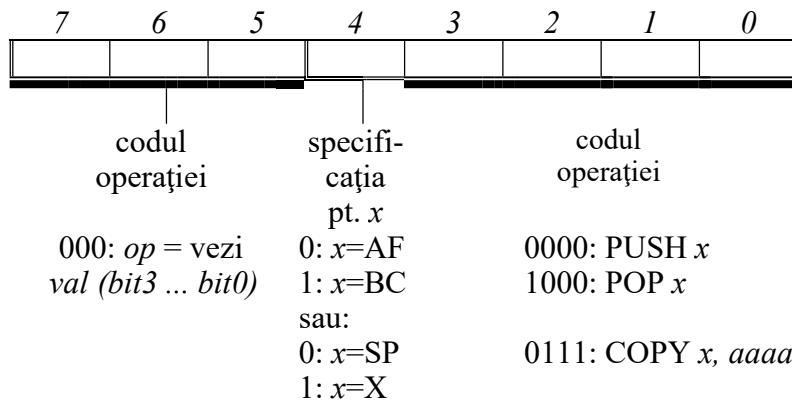
e). când $val(bit_7 \dots bit_5) = 111_2$ și $val(bit_4\ bit_3) \in [00_2 \dots 10_2]$
 și $val(bit_2 \dots bit_0) = 110_2$:



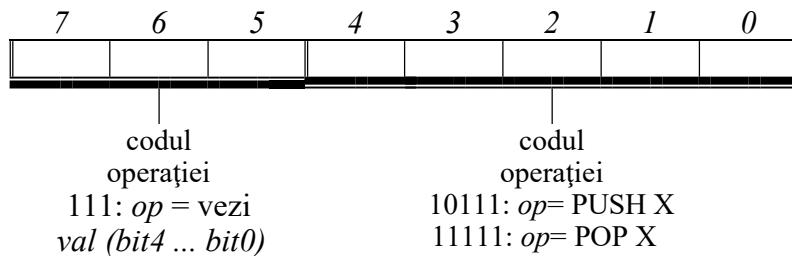
f). când $val(bit7 \dots bit5) = 111_2$ și $val(bit4 \ bit3) = 00_2$
 și $val(bit2 \dots bit0) = 111_2$:



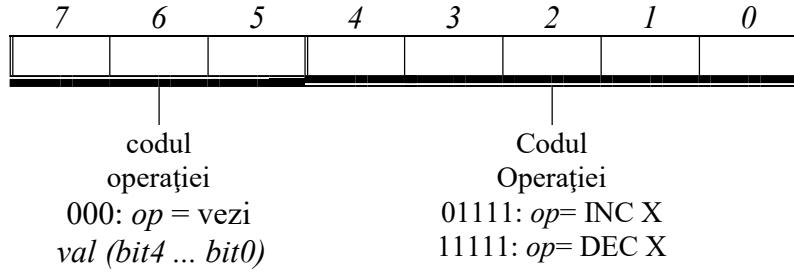
g). când $val(bit7 \dots bit5) = 000_2$ și $val(bit3 \dots bit0) \in \{0000, 1000, 0111_2\}$:



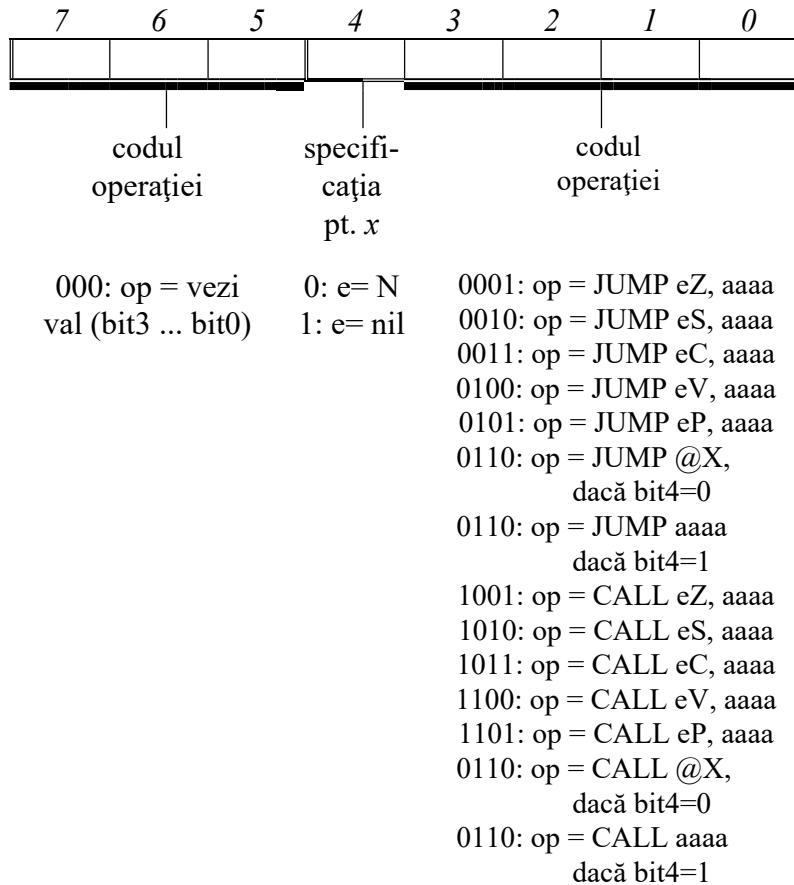
h). când $val(bit7 \dots bit5) = 111_2$ și $val(bit3 \dots bit0) = 1111_2$:



i). când $val (bit7 \dots bit5) = 000_2$ și $val (bit3 \dots bit0) = 1111_2$:



j). când $val (bit7 bit6 bit5) = 000_2$ și $val (bit2 bit1 bit0) \in \{001\dots110_2\}$:



Ca o consecință a convențiilor de codificare de mai sus, rezultă tabelul din figura 6.3.2_1. Acesta cuprinde, într-o manieră sintetică, întregul set de instrucții al procesorului *PD*, cu modurile de adresare pe care le admit.

CAP. 6. DESPRE PROCESOR. UNITATEA DE COMANDĂ

6 5	4 3	2 1 0	Notăție	L	7 6 5	4 3	2 1 0	Notăție	L	7 6 5	4 3	2 1 0	Notăție	L
001	00	000	COPY A, A	1	010	00	000	ADD A, A	1	101	00	000	SUB A, A	1
001	00	001	COPY A, B	1	010	00	001	ADD A, B	1	101	00	001	SUB A, B	1
001	00	010	COPY A, C	1	010	00	010	ADD A, C	1	101	00	010	SUB A, C	1
001	00	011	COPY A, aa	2	010	00	011	ADD A, aa	2	101	00	011	SUB A, aa	2
001	00	100	COPY A, @X	1	010	00	100	ADD A, @X	1	101	00	100	SUB A, @X	1
001	00	101	COPY A, @aaaa	3	010	00	101	ADD A, @aaaa	3	101	00	101	SUB A, @aaaa	3
001	01	000	COPY B, A	1	010	01	000	ADD B, A	1	101	01	000	SUB B, A	1
001	01	001	COPY B, B	1	010	01	001	ADD B, B	1	101	01	001	SUB B, B	1
001	01	010	COPY B, C	1	010	01	010	ADD B, C	1	101	01	010	SUB B, C	1
001	01	011	COPY B, aa	2	010	01	011	ADD B, aa	2	101	01	011	SUB B, aa	2
001	01	100	COPY B, @X	1	010	01	100	ADD B, @X	1	101	01	100	SUB B, @X	1
001	01	101	COPY B, @aaaa	3	010	01	101	ADD B, @aaaa	3	101	01	101	SUB B, @aaaa	3
001	10	000	COPY C, A	1	010	10	000	ADD C, A	1	101	10	000	SUB C, A	1
001	10	001	COPY C, B	1	010	10	001	ADD C, B	1	101	10	001	SUB C, B	1
001	10	010	COPY C, C	1	010	10	010	ADD C, C	1	101	10	010	SUB C, C	1
001	10	011	COPY C, aa	2	010	10	011	ADD C, aa	2	101	10	011	SUB C, aa	2
001	10	100	COPY C, @X	1	010	10	100	ADD C, @X	1	101	10	100	SUB C, @X	1
001	10	101	COPY C, @aaaa	3	010	10	101	ADD C, @aaaa	3	101	10	101	SUB C, @aaaa	3
001	11	000	COPY @X, A	1	010	11	000	ADD @X, A	1	101	11	000	SUB @X, A	1
001	11	001	COPY @X, B	1	010	11	001	ADD @X, B	1	101	11	001	SUB @X, B	1
001	11	010	COPY @X, C	1	010	11	010	ADD @X, C	1	101	11	010	SUB @X, C	1
001	11	011	COPY @X, aa	2	010	11	011	ADD @X, aa	2	101	11	011	SUB @X, aa	2
001	11	100	COPY @aaaa, A	3	010	11	100	ADD @aaaa, A	3	101	11	100	SUB @aaaa, A	3
001	11	101	COPY @aaaa, B	3	010	11	101	ADD @aaaa, B	3	101	11	101	SUB @aaaa, B	3
001	11	110	COPY @aaaa, C	3	010	11	110	ADD @aaaa, C	3	101	11	110	SUB @aaaa, C	3
001	11	111	COPY @aaaa, aa	4	010	11	111	ADD @aaaa, aa	4	101	11	111	SUB @aaaa, aa	4
100	00	000	AND A, A	1	101	00	000	OR A, A	1	110	00	000	XOR A, A	1
100	00	001	AND A, B	1	101	00	001	OR A, B	1	110	00	001	XOR A, B	1
100	00	010	AND A, C	1	101	00	010	OR A, C	1	110	00	010	XOR A, C	1
100	00	011	AND A, aa	2	101	00	011	OR A, aa	2	110	00	011	XOR A, aa	2
100	00	100	AND A, @X	1	101	00	100	OR A, @X	1	110	00	100	XOR A, @X	1
100	00	101	AND A, @aaaa	3	101	00	101	OR A, @aaaa	3	110	00	101	XOR A, @aaaa	3
100	01	000	AND B, A	1	101	01	000	OR B, A	1	110	01	000	XOR B, A	1
100	01	001	AND B, B	1	101	01	001	OR B, B	1	110	01	001	XOR B, B	1
100	01	010	AND B, C	1	101	01	010	OR B, C	1	110	01	010	XOR B, C	1
100	01	011	AND B, aa	2	101	01	011	OR B, aa	2	110	01	011	XOR B, aa	2
100	01	100	AND B, @X	1	101	01	100	OR B, @X	1	110	01	100	XOR B, @X	1
100	01	101	AND B, @aaaa	3	101	01	101	OR B, @aaaa	3	110	01	101	XOR B, @aaaa	3
100	10	000	AND C, A	1	101	10	000	OR C, A	1	110	10	000	XOR C, A	1
100	10	001	AND C, B	1	101	10	001	OR C, B	1	110	10	001	XOR C, B	1
100	10	010	AND C, C	1	101	10	010	OR C, C	1	110	10	010	XOR C, C	1
100	10	011	AND C, aa	2	101	10	011	OR C, aa	2	110	10	011	XOR C, aa	2
100	10	100	AND C, @X	1	101	10	100	OR C, @X	1	110	10	100	XOR C, @X	1
100	10	101	AND C, @aaaa	3	101	10	101	OR C, @aaaa	3	110	10	101	XOR C, @aaaa	3
100	11	000	AND @X, A	1	101	11	000	OR @X, A	1	110	11	000	XOR @X, A	1
100	11	001	AND @X, B	1	101	11	001	OR @X, B	1	110	11	001	XOR @X, B	1
100	11	010	AND @X, C	1	101	11	010	OR @X, C	1	110	11	010	XOR @X, C	1
100	11	011	AND @X, aa	2	101	11	011	OR @X, aa	2	110	11	011	XOR @X, aa	2
100	11	100	AND @aaaa, A	3	101	11	100	OR @aaaa, A	3	110	11	100	XOR @aaaa, A	3
100	11	101	AND @aaaa, B	3	101	11	101	OR @aaaa, B	3	110	11	101	XOR @aaaa, B	3
100	11	110	AND @aaaa, C	3	101	11	110	OR @aaaa, C	3	110	11	110	XOR @aaaa, C	3
100	11	111	AND @aaaa, aa	4	101	11	111	OR @aaaa, aa	4	110	11	111	XOR @aaaa, aa	4

Fig. 6.3.2_1. Instrucțiile procesorului PD (se continuă).

7 6 5	4 3	2 1 0	<i>Notătie</i>	L	0 1 0	0 0	0 0 0	<i>Notătie</i>	L	7 6 5	4 3	2 1 0	<i>Notătie</i>	L
111	00	000	INC A	1	000	00	000	PUSH AF	1					
111	00	001	DEC A	1	000	00	001	JUMP NZ, aaaa	3					
111	00	010	SHL A	1	000	00	010	JUMP NS, aaaa	3					
111	00	011	SHR A	1	000	00	011	JUMP NC, aaaa	3					
111	00	100			000	00	100	JUMP NV, aaaa	3					
111	00	101	IN A, @aa	2	000	00	101	JUMP NP, aaaa	3					
111	01	110	OUT @aa, A	2	000	01	110	JUMP @X	1					
111	01	111	RET	1	000	01	111	LD SP, aaaa	3					
111	01	000	INC B	1	000	01	000	POP AF	1					
111	01	001	DEC B	1	000	01	001	CALL NZ, aaaa	3					
111	01	010	SHL B	1	000	01	010	CALL NS, aaaa	3					
111	01	011	SHR B	1	000	01	011	CALL NC, aaaa	3					
111	10	100			000	10	100	CALL NV, aaaa	3					
111	10	101	IN B, @aa	2	000	10	101	CALL NP, aaaa	3					
111	10	110	OUT @aa, B	2	000	10	110	CALL @X	1					
111	10	111	PUSH X		000	10	111	INC X	1					
111	10	000	INC C	1	000	10	000	PUSH BC	1					
111	10	001	DEC C	1	000	10	001	JUMP Z, aaaa	3					
111	11	010	SHL C	1	000	11	010	JUMP S, aaaa	3					
111	11	011	SHR C	1	000	11	011	JUMP C, aaaa	3					
111	11	100			000	11	100	JUMP V, aaaa	3					
111	11	101	IN C, @aa	2	000	11	101	JUMP P, aaaa	3					
111	11	110	OUT @aa, C	2	000	11	110	JUMP aaaa	3					
111	11	111	POP X		000	11	111	LD X, aaaa	3					
111	11	000	INC @X	1	000	11	000	POP BC	1					
111	11	001	DEC @X	1	000	11	001	CALL Z, aaaa	3					
111	00	010	SHL @X	1	000	00	010	CALL S, aaaa	3					
111	00	011	SHR @X	1	000	00	011	CALL C, aaaa	3					
111	00	100	INC @aaaa	3	000	00	100	CALL V, aaaa	3					
111	00	101	DEC @aaaa	3	000	00	101	CALL P, aaaa	3					
111	00	110	SHL @aaaa	3	000	00	110	CALL aaaa	3					
111	00	111	SHR @aaaa	3	000	00	111	DEC X	1					

Legendă: L: lungimea în octeți a instrucțiilor.

Fig. 6.3.2_1. Instrucțiile procesorului PD (continuare).

6.3.3. Conținutul instrucțiilor

1. Instrucția de copiere (COPY).

Notatie:

COPY x, y

unde:

$x \in \{A, B, C, @X, @aaaa, X, SP\}$

$y \in \{A, B, C, @X, @aaaa, aa, aaaa\}$, cu $y \in \{@X, @aaaa\} \Rightarrow x \notin \{@X, @aaaa\}$ și $x \in \{X, SP\} \Leftrightarrow y = aaaa$

Descriere:

Copiază y în x

Codificare:

Așa cum rezultă din figura 6.3.2_1, instrucția COPY poate avea 1-4 octeți, în funcție de modurile de adresare pe care le folosește. Semnificațiile octetilor sunt următoarele:

a). cazul $L=1$ octet

Octetul 1 (unicul) reprezintă codul operației. În plus, în cazul adresărilor “de registru” și “indirectă”, tot el indică și registrul vizat / locația de memorie vizată prin respectivele adresări.

b). cazul $L=2$ octeți

Octetul 1 reprezintă codul operației. În plus, în cazul adresărilor “de registru” și “indirectă”, tot el indică și registrul vizat / locația de memorie vizată prin respectivele adresări.

Octetul 2 reprezintă operandul efectiv -adică: pe aa -, cazul $L=2$ octeți conținând o adresare imediată.

c). cazul $L=3$ octeți

Octetul 1 reprezintă codul operației. În plus, tot el indică și registrul vizat prin adresarea “de registru” pe care cazul $L=3$ octeți o conține inevitabil.

Octetul 2 reprezintă partea mai puțin semnificativă a adresei locației de memorie vizată sau a operandului pe 16 biți -adică: pe $aaaa_L$ -, cazul $L=3$ octeți conținând fie o adresare directă, fie o adresare imediată cu destinația X sau SP.

Octetul 2 reprezintă partea mai semnificativă a adresei locației de memorie vizată sau a operandului pe 16 biți -adică: pe $aaaa_H$ -, cazul $L=3$ octeți conținând fie o adresare directă, fie o adresare imediată cu destinația X sau SP.

d). cazul $L=4$ octeți

Octetul 1 reprezintă codul operației.

Octetul 2 reprezintă partea mai puțin semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_L$, cazul $L=4$ octeți conținând o adresare directă.

Octetul 3 reprezintă partea mai semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_H$, cazul $L=4$ octeți conținând o adresare directă.

Octetul 4 reprezintă operandul efectiv -adică: pe aa , cazul $L=4$ octeți conținând o adresare imediată.

2. Instrucția de adunare (ADD).

Notăție:

ADD x, y

unde:

$x \in \{A, B, C, @X, @aaaa\}$

$y \in \{A, B, C, @X, @aaaa, aa\}$, cu $y \in \{@X, @aaaa\} \Rightarrow x \notin \{@X, @aaaa\}$

Descriere:

Adună pe x cu y și oferă rezultatul în x și informații despre rezultat în F .

Codificare:

Tot ce s-a spus în privința codificării la instrucția COPY este valabil și pentru instrucția ADD.

3. Instrucția de scădere (SUB).

Notăție:

SUB x, y

unde:

$x \in \{A, B, C, @X, (aaaa)\}$

$y \in \{A, B, C, @X, (aaaa), aa\}$, cu $y \in \{@X, (aaaa)\} \Rightarrow x \notin \{@X, (aaaa)\}$

Descriere:

Scade din x pe y și oferă rezultatul în x și informații despre rezultat în F .

Codificare:

Tot ce s-a spus în privința codificării la instrucția COPY este valabil și pentru instrucția SUB.

4. Instrucția de efectuare a operației SI (AND).

Notatie:

AND x, y

unde:

$x \in \{A, B, C, @X, (aaaa)\}$

$y \in \{A, B, C, @X, (aaaa), aa\}$, cu $y \in \{@X, (aaaa)\} \Rightarrow x \notin \{@X, (aaaa)\}$

Descriere:

Efectuează operația SI bit la bit între x și y și oferă rezultatul în x și informații despre rezultat în F .

Codificare:

Tot ce s-a spus în privința codificării la instrucția COPY este valabil și pentru instrucția AND.

5. Instrucția de efectuare a operației SAU (OR).

Notatie:

OR x, y

unde:

$x \in \{A, B, C, @X, (aaaa)\}$

$y \in \{A, B, C, @X, (aaaa), aa\}$, cu $y \in \{@X, (aaaa)\} \Rightarrow x \notin \{@X, (aaaa)\}$

Descriere:

Efectuează operația SAU bit la bit între x și y și oferă rezultatul în x și informații despre rezultat în F .

Codificare:

Tot ce s-a spus în privința codificării la instrucția COPY este valabil și pentru instrucția OR.

6. Instrucția de efectuare a operației SAU EXCLUSIV (XOR).

Notatie:

XOR x, y

unde:

$x \in \{A, B, C, @X, (aaaa)\}$

$y \in \{A, B, C, @X, (aaaa), aa\}$, cu $y \in \{@X, (aaaa)\} \Rightarrow x \notin \{@X, (aaaa)\}$

Descriere:

Efectuează operația SAU EXCLUSIV bit la bit între x și y și oferă rezultatul în x și informații despre rezultat în F .

Codificare:

Tot ce s-a spus în privința codificării la instrucția COPY este valabil și pentru instrucția XOR.

7. Instrucția de incrementare (INC).

Notatie:

INC x

unde:

$x \in \{A, B, C, @X, @aaaa, X\}$

Descriere:

Incrementează x și oferă informații despre rezultat în F .

Codificare:

Așa cum rezultă din figura 6.3.2_1, instrucția INC poate avea fie 1 octet, fie 3 octeți, în funcție de modurile de adresare pe care le folosește. Semnificațiile octețiilor sunt următoarele:

a). cazul $L = 1$ octet

Octetul 1 (unicul) reprezintă codul operației. În plus, în cazul adresărilor “la registru” și “indirectă”, tot el indică și registrul / locația de memorie vizat / vizată prin respectivele adresări.

b). cazul $L = 3$ octeți

Octetul 1 reprezintă codul operației.

Octetul 2 reprezintă partea mai puțin semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_L$, cazul $L=3$ octeți conținând o adresare directă.

Octetul 3 reprezintă partea mai semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_H$, cazul $L=3$ octeți conținând o adresare directă.

8. Instrucția de decrementare (DEC).

Notatie:

DEC x

unde:

$x \in \{A, B, C, @X, @aaaa, X\}$

Descriere:

Decrementează x și oferă informații despre rezultat în F .

Codificare:

Tot ce s-a spus în privința codificării la instrucția INC este valabil și pentru instrucția DEC.

9. Instrucția de deplasare la dreapta (SHR).

Notărie:

SHR x

unde:

$x \in \{A, B, C, @X, @aaaa\}$

Descriere:

Deplasează la dreapta cu o poziție biții lui x și oferă informații despre rezultat în F . În procesul de deplasare, în rangul de pondere 2^7 al lui x se introduce valoarea 0, iar ceea ce a fost în rangul de pondere 2^0 se transferă în fanionul C .

Codificare:

Tot ce s-a spus în privința codificării la instrucția INC este valabil și pentru instrucția SHR.

10. Instrucția de deplasare la stânga (SHL).

Notărie:

SHL x

unde:

$x \in \{A, B, C, @X, @aaaa\}$

Descriere:

Deplasează la stânga cu o poziție biții lui x și oferă informații despre rezultat în F . În procesul de deplasare, în rangul de pondere 2^0 al lui x se introduce valoarea 0, iar ceea ce a fost în rangul de pondere 2^7 se transferă în fanionul C .

Codificare:

Tot ce s-a spus în privința codificării la instrucția INC este valabil și pentru instrucția SHL.

11. Instrucția de intrare (IN).

Notărie:

IN x, y

unde:

$x \in \{A, B, C\}$

$y = @aa$

Descriere:

Introdu în x o copie a informației de un octet aflată într-un port de intrare y .

Codificare:

Așa cum rezultă din figura 6.3.2_1, instrucția IN are 2 octeți. Semnificațiile octeților sunt următoarele:

Octetul 1 reprezintă codul operației. În plus, tot el indică și registrul vizat prin adresarea “la registru” pe care instrucția o conține inevitabil.

Octetul 2 reprezintă adresa portului vizat -adică: pe aa .

12. Instrucția de ieșire (OUT).

Notatie:

OUT x, y

unde:

$$x = @aa$$

$$y \in \{A, B, C\}$$

Descriere:

Scoate într-un port de ieșire x o copie a octetului aflat în y .

Codificare:

Tot ce s-a spus în privința codificării la instrucția IN este valabil și pentru instrucția OUT.

13. Instrucția de scriere în stivă (PUSH).

Notatie:

PUSH x

unde:

$$x \in \{AF, BC, X\}$$

Descriere:

Depune x în stivă.

Codificare:

Așa cum rezultă din figura 6.3.2_1, instrucția PUSH are 1 octet. Semnificațiile sale sunt următoarele:

Octetul 1 (unicul) reprezintă codul operației. În plus, tot el indică și perechea de registre vizată prin adresarea “de registru” pe care instrucția o conține inevitabil.

14. Instrucția de citire din stivă (POP).

Notatie:

POP x

unde:

$x \in \{AF, BC, X\}$

Descriere:

Încarcă x prin descărcarea stivei.

Codificare:

Tot ce s-a spus în privința codificării la instrucția PUSH este valabil și pentru instrucția POP.

15. Instrucția de salt (JUMP).

Notatie:

dacă $W \neq nil$:

JUMP eW, x

dacă $W = nil$:

JUMP x

unde:

$e \in \{nil, N\}$

$W \in \{Z, S, C, V, P, nil\}$

$x \in \{@X, aaaa\}$

Descriere:

Ieșî din secvența normală de rulare, eventual condiționat, și continuă programul cu instrucția localizată prin x; altfel spus: fă un salt. Dacă $W \neq nil$, atunci ieșirea din secvență se face condiționat de valoarea fanionului indicat de W: dacă $e = nil$, atunci, pentru salt, respectivul fanion trebuie să fie la unu; dacă $e = N$, atunci, pentru salt, respectivul fanion trebuie să fie la zero. Când condiția de salt nu este îndeplinită, programul continuă, ca și în cazul instrucțiilor ordinare, cu instrucția imediat următoare din punct de vedere topologic.

Codificare:

Așa cum rezultă din figura 6.3.2_1, instrucția JUMP poate avea fie 1 octet, fie 3 octeți, în funcție de modurile de adresare pe care le folosește. Semnificațiile octețiilor sunt următoarele:

a). cazul $L = 1 \text{ octet}$

Octetul 1 (unicul) reprezintă codul operației. În plus, tot el indică și faptul că adresa instrucției la care se vizează să se efectueze saltul se află în registrul X.

b). cazul $L = 3 \text{ octeți}$

Octetul 1 reprezintă codul operației, iar în cazul instrucțiilor de salt condiționat –în plus, condiția impusă.

Octetul 2 reprezintă partea mai puțin semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_L$, cazul $L=3 \text{ octeți}$ conținând o adresare directă.

Octetul 3 reprezintă partea mai semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_H$, cazul $L=3 \text{ octeți}$ conținând o adresare directă.

Evident, în cazurile în care saltul trebuie executat, acest lucru se realizează prin încărcarea registrului PC cu conținutul registrului X –dacă $x = @X$, respectiv cu octetii 2 și 3 ai instrucției –dacă $x = aaaa$.

16. Instrucția de apel de subrutină (CALL).

Notatie:

dacă $W \neq nil$:

 CALL eW, x

dacă $W = nil$:

 CALL x

unde:

$e \in \{nil, N\}$

$W \in \{Z, S, C, V, P, nil\}$

$x \in \{@X, aaaa\}$

Descriere:

Ieși din secvența normală de rulare, eventual condiționat, și continuă programul cu instrucția localizată prin x , după ce salvezi în stivă adresa instrucției imediat următoare din punct de vedere topologic; altfel spus: fă un salt reversibil sau *apeleză o subrutină*. Dacă $W \neq nil$, atunci apelarea subruteinei se face condiționat de valoarea fanionului indicat de W: dacă $e = nil$, atunci, pentru apel, respectivul fanion trebuie să fie la unu; dacă $e = N$, atunci, pentru apel, respectivul fanion trebuie să fie la zero. Când condiția de apel nu este îndeplinită, programul continuă, ca și în cazul instrucțiilor ordinare, cu instrucția imediat următoare din punct de vedere topologic.

Codificare:

Așa cum rezultă din figura 6.3.2_1, instrucția CALL poate avea fie 1 octet, fie 3 octeți, în funcție de modurile de adresare pe care le folosește. Semnificațiile octețiilor sunt următoarele:

a). cazul $L = 1 \text{ octet}$

Octetul 1 (unicul) reprezintă codul operației. În plus, tot el indică și faptul că adresa instrucției la care se vizează să se efectueze apelul –altfel spus: adresa de început a subruteinei- se află în registrul X.

b). cazul $L = 3 \text{ octeți}$

Octetul 1 reprezintă codul operației, iar în cazul instrucțiilor de apel condiționat –în plus, condiția impusă.

Octetul 2 reprezintă partea mai puțin semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_L$ -, cazul $L=3 \text{ octeți}$ conținând o adresare directă.

Octetul 3 reprezintă partea mai semnificativă a adresei locației de memorie vizată -adică: pe $aaaa_H$ -, cazul $L=3 \text{ octeți}$ conținând o adresare directă.

Evident, în cazurile în care apelul trebuie executat, acest lucru se realizează prin încărcarea registrului PC cu conținutul registrului X –dacă $x = @X$ -, respectiv cu octetii 2 și 3 ai instrucției –dacă $x = aaaa$, după ce conținutul său, reprezentând adresa instrucției imediat următoare din punct de vedere topologic, a fost salvat în stivă.

17. Instrucția de revenire din subrutină (RET).

Notărie:

RET

Descriere:

Ieși din secvența normală de rulare și continuă programul cu instrucția a cărei adresă se află în vârful stivei sau, altfel spus: *revino din subrutină* (evident, dacă s-a operat corect asupra stivei, atunci în momentul execuției instrucției RET, în vârful stivei se găsește adresa instrucției imediat următoare din punct de vedere topologic instrucției CALL care a apelat subrutina –vezi descrierea instrucției CALL).

Codificare:

Așa cum rezultă din figura 6.3.2_1, instrucția RET are 1 octet. Semnificațiile sale sunt următoarele:

Octetul 1 (unicul) reprezintă codul operației. În plus, tot el indică și faptul că adresa instrucției la care se vizează să se producă revenirea –altfel spus: *adresa de revenire*- se află în vârful stivei.

Evident, revenirea din subrutină se face prin descărcarea primilor 2 octeți din vârful stivei în registrul PC.

6.4. Implementarea instrucțiilor în procesoarele *von Neumann*

6.4.1. Aspecte introductive

În acest subcapitol, se va ilustra ce se întâmplă și cum se desfășoară lucrurile la nivel elementar, pentru a se face posibilă procesarea datelor, procesare care, aşa cum s-a arătat încă de la începutul lucrării noastre, înseamnă: acceptarea, reținerea, înțelegerea și executarea de instrucții. Vor fi considerate câteva instrucții reprezentative și se vor construi cronogramele semnalelor de comandă interne și externe prin care ele se implementează. Principiile organizării interne și funcționării calculatoarelor vor fi, astfel, clar și definitiv înțelese.

Așa cum s-a arătat cu ocazia prezentării sevențiatorului, rularea unei instrucții are două etape:

- 1). **Etapa de extragere sau aducere (“*fetch*”)** a codului operației și a informațiilor privind locul operanzilor și-sau rezultatului indisociabile de codul operației.
- 2). **Etapa de execuție** propriu-zisă a instrucției.

Etapa de aducere cuprinde un ciclu de citire din memorie identic pentru toate instrucțiile. În cadrul acestui ciclu, se citește locația cu adresa prezentă în registrul **PC** și conținutul ei se duce în registrul **IR**.

Etapa de execuție cuprinde unul sau mai multe cicluri de citire, respectiv de scriere, vizând memoria sau un port, respectiv operații pur interne procesorului, specifice pentru fiecare instrucție. Aceste cicluri au ca obiectiv, în principal, citirea operanzilor, respectiv scrierea rezultatului, dar, uneori, mai înainte de asta, aducerea informațiilor privind locul operanzilor și-sau rezultatului, reprezentând entități din cuvântul instrucție alăturate codului operației, dar separate de cea care-l conține pe acesta –vezi paragraful cu modurile de adresare–.

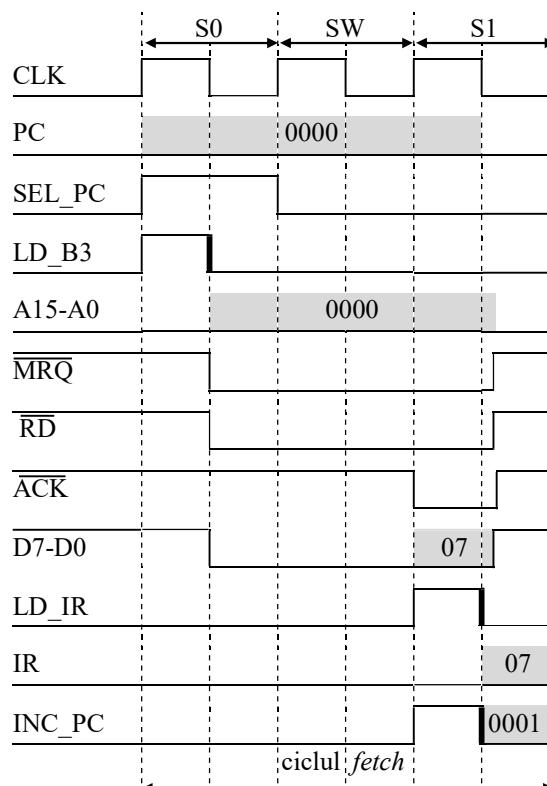
La pornirea calculatorului, are loc un proces de initializare al cărui efect minimal este aducerea la zero a registrului PC și instituirea stării inițiale a secvențiatorului, astfel încât el să-și demareze activitatea cu un ciclu *fetch*.

6.4.2. Implementarea ciclului *fetch*

Având în vedere obiectivele ciclului *fetch*, cronograma sa este cea care face obiectul figurii 6.4.2_1.

Notă:

Pentru înțelegerea a ceea ce urmează, este obligatorie folosirea schemei procesorului PD –figura 6.2_1- și schemei memoriei avută în vedere –figura 2_2-.



Notă: Cu linii îngroșate s-au evidențiat fronturile active.

Fig. 6.4.2_1. Cronograma ciclului *fetch*.

Figura 6.4.2_1 arată că ciclul *fetch* debutează pe frontul ridicător al semnalului **CLK**.

Semnalul **SEL_PC** are rolul de a selecta la ieșirea multiplexorului de adrese **AM** informația din **PC**, adică: adresa instrucției care urmează a se executa. Această adresă se încarcă în registrul tampon de adrese **B3**, pe frontul coborâtor al semnalului **LD_B3**. Simultan, se lansează spre memorie o cerere de lucru, prin activarea semnalului **MRQ** și se anunță că operația de executat este una de citire, prin activarea semnalului **RD**. Deodată cu activarea semnalului **MRQ**, pe liniile de adrese **A0-A15** se plasează adresa locației de memorie vizate, adică: conținutul registrului **B3**. Memoria răspunde cererii primite, activând semnalul **ACK** în momentul în care termină de efectuat citirea și anume: după $1 \frac{1}{2}$ perioade ale semnalului de tact (evident, aceasta este o ipoteză de lucru). În acest moment, informația citită se află pe liniile de date **D0-D7** și, implicit, la intrările tuturor registrelor conectate la acestea. După ce sesizează trecerea la 0 a semnalului **ACK**, sevențiatorul **S** activează semnalul **LD_IR**. Pe frontul său coborâtor, acest semnal va determina preluarea informației furnizate de memorie –adică: a codului operației instrucției ce urmează a se executa- în registrul **IR**. Imediat în continuare, sevențiatorul dezactivează semnalul **MRQ**, ceea ce face ca la nivelul memoriei să fie dezactivat semnalul **ACK**. Până la încheierea ciclului de tact curent, are loc decodificarea codului operației, astfel încât din următorul ciclu să poată începe execuția, adică: secvența de operații elementare pe care instrucția în cauză o implică. Așa cum se poate observa pe cronomogramă, deodată cu semnalul **LD_IR** este activat și semnalul **INC_PC**, și el activ pe front coborâtor. Prin această activare, se pregătește în **PC** adresa la care cel mai probabil urmează să se facă următorul acces, neîntâmplându-se asta doar dacă instrucția tocmai ”adusă” este una de salt pe 1 octet, una de apel de subrutină pe 1 octet sau una de revenire din subrutină.

De remarcat că, în ipoteza de lucru adoptată, între stările **S0** și **S1** a fost inserată o perioadă de tact în care nu se întâmplă nimic nou față de perioada precedentă. O asemenea perioadă poartă numele de stare de aşteptare sau “stare *wait*”. Cu ajutorul stărilor *wait*, se realizează sincronizarea dintre procesor și memorie, astfel încât ele să poată conlucra chiar dacă au ritmuri de “mișcare” diferite. Evident, semnalul \overline{ACK} este cel în funcție de care se stabilește numărul stărilor de aşteptare ce se inserează într-un ciclu de lucru cu memoria.

6.4.3. Implementarea instrucției “COPY SP, aaaa”

Așa cum s-a arătat, activitatea procesorului demarează cu 0 în registrul **PC**. Asta înseamnă că prima instrucție rulată va fi cea de la adresa 0. Întrucât una dintre primele operații care trebuie efectuate la nivel *software* este inițializarea pointerului de stivă, să presupunem că începând de la adresa 0 se află instrucția:

COPY SP, 0

Această presupunere implică o alta, și anume: că stiva este implementată începând de la adresa $FFFF_{16}$ spre adrese mai mici (ne reamintim: lucrul cu stiva se face decrementând **SP** înaintea efectuării unei scrisori).

Notă:

În cele ce urmează, valorile hexazecimale se vor marca asociind succesiunii cifrelor prin care ele se exprimă sufixul “H”. Se face convenția ca atunci când prima cifră a unei valori hexazecimale este o literă (evidenț, una din literele A-F), în fața acesteia să se adauge cifra 0, pentru a se crea posibilitatea deosebirii succesiunilor de caractere care reprezintă numere de construcții alfanumerice de altă natură.

Instrucția “**COPY SP, aaaa**” este una pe 3 octeți. Ea se prezintă în memorie, când $aaaa=0000_{16}$, aşa cum se arată în figura 6.4.3_1.

Adresa [în hexa]	Conținuturile locațiilor cu adresele date de col (1) [în hexa]	Semnificația informației din col (2)	Instrucțiile aflate în locațiile cu adresele date de col (1)
(1)	(2)	(3)	(4)
0000	07	octet 1	COPY SP, 0000H
0001	00	octet 2	
0002	00	octet 3	
0003	16	octet 1	JUMP 0100H
0004	00	octet 2	
0005	01	octet 3	

Fig. 6.4.3_1. Conținutul zonei de memorie de adrese 0000H-0005H.

Presupunem că ciclul *fetch* a fost deja parcurs. Prin urmare, în registrul **IR** se află codul 07H, iar registrul **PC** a ajuns la conținutul 0001H.

Prin decodificarea codului 07H, secvențiatorul află că are de executat instrucția “**COPY SP, aaaa**”, despre care “știe” că este una pe 3 octeți, și, prin urmare, declanșează 2 cicluri de citire din memorie, similare ciclului *fetch*, asistate, de asemenea, de registrul **PC**, pentru aducerea octeților 2 și 3 ai instrucției, care reprezintă valoarea **aaaa**. În aceste cicluri, însă, în locul activării semnalului **LD_IR**, în timp ce informațiile citite sunt stabile pe liniile de date, se activează semnalele **LD_SPL**, respectiv **LD_SPH**. Se înțelege că efectul cumulativ al acțiunii acestor două semnale va fi încărcarea registrului **SP** cu valoarea 0000H.

Întrucât la sfârșitul fiecărui ciclu de citire registrul **PC** este incrementat (de reținut: incrementarea registrului **PC** cu ocazia oricărei citiri făcută sub asistență să reprezintă o regulă absolută!), la finele instrucției, în **PC** se va găsi valoarea 0003H, reprezentând adresa următoarei instrucții.

Cronogramele celor 2 cicluri de citire, pe care le vom numi “*execuție 1*”, respectiv “*execuție 2*”, sunt redate în figurile 6.4.3_2, respectiv 6.4.3_3.

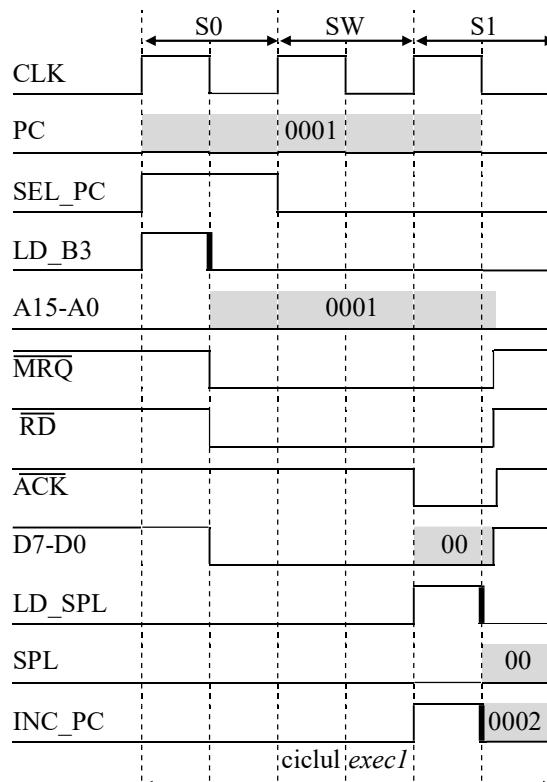


Fig. 6.4.3_2. Cronograma ciclului execuție 1 al instrucției "COPY SP, aaaa"
cu aaaa = 0000H

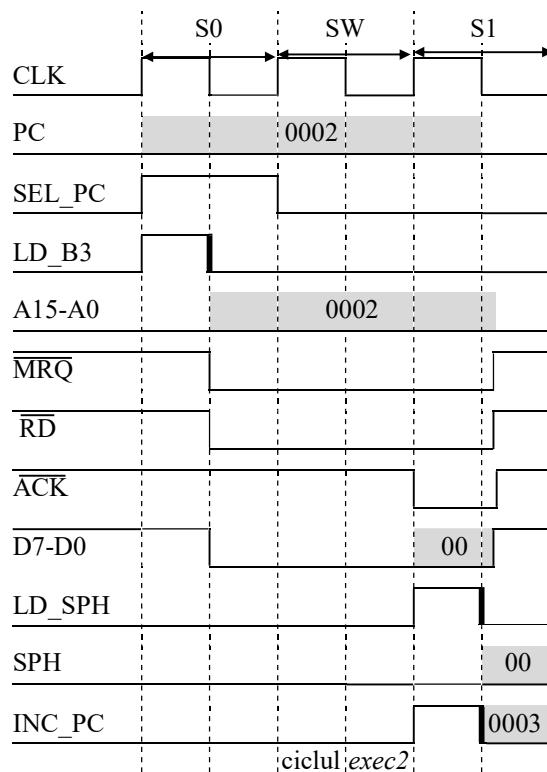


Fig. 6.4.3_3. Cronograma ciclului execuție 2 al instrucției "COPY SP,
aaaa"

6.4.4. Implementarea instrucției “JUMP aaaa”

L-a întocmirea figurii 6.4.3_1, s-a presupus că în zona de memorie considerată, imediat după instrucția “**COPY SP, 0000H**” –adică: începând de la adresa 0003H- se află instrucția “**JUMP 0100H**”. Să analizăm cum ar rula “programul” considerat, în continuare.

După ciclul *execuție 2* al instrucției “**COPY SP, 0000H**”, se va executa un nou ciclu *fetch*, de data aceasta cu referire la adresa 0003H, prin care în registrul **IR** va fi adus octetul 16H, adică: codul instrucției “**JUMP aaaa**”.

Instrucția “**JUMP aaaa**” fiind și ea una pe 3 octeți, vor urma alte 2 cicluri de citire, omoloage ciclurilor *execuție 1* și *execuție 2* de la instrucția “**COPY SP, aaaa**”, în cadrul cărora, cu ajutorul semnalelor **LD_IREL**, respectiv **LD_IREH**, se încarcă registrele **IREL** și **IREH** cu informația aflată la adresele 0004H, respectiv 0005H, adică: cu octetii 2 și 3 ai instrucției. Întrucât această informație reprezintă adresa la care se face saltul, adică: adresa locației unde se află instrucția ce trebuie introdusă în rulare după instrucția **JUMP**, în continuare, se va proceda la transferul succesiv al conținuturilor registrelor **IREL** și **IREH** în registrele **PCL**, respectiv **PCH**, în cadrul unui ciclu de lucru desfășurat exclusiv la nivelul procesorului. Numim acest ciclu “*execuție 3*”.

Cronogramele ciclurilor *execuție 1*, *execuție 2* și *execuție 3* ale instrucției “**JUMP aaaa**” sunt redate în figurile 6.4.4_1, 6.4.4_2 și 6.4.4_3.

În concluzie, se punctează faptul că valoarea care se va găsi înscrisă în registrul **PC** la sfârșitul execuției instrucției “**JUMP 0100H**”, este 0100H.

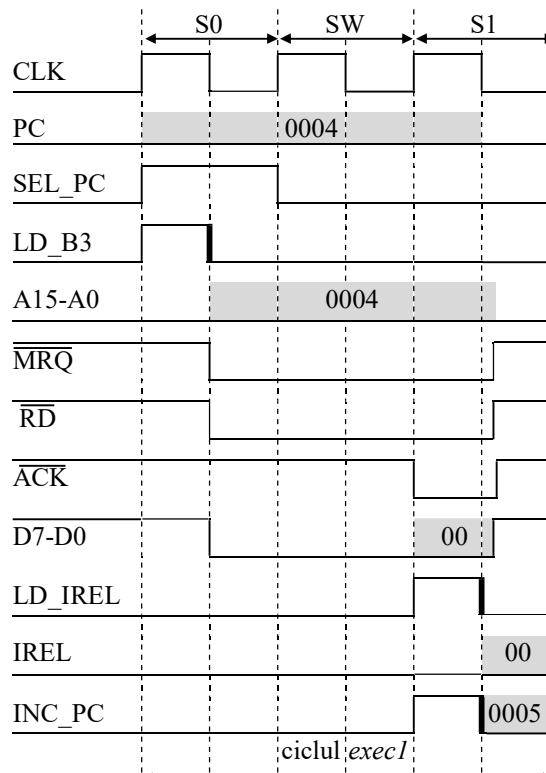


Fig. 6.4.4_1. Cronograma ciclului *execuție 1* al instrucției "JUMP aaaa"
cu aaaa = 0100H

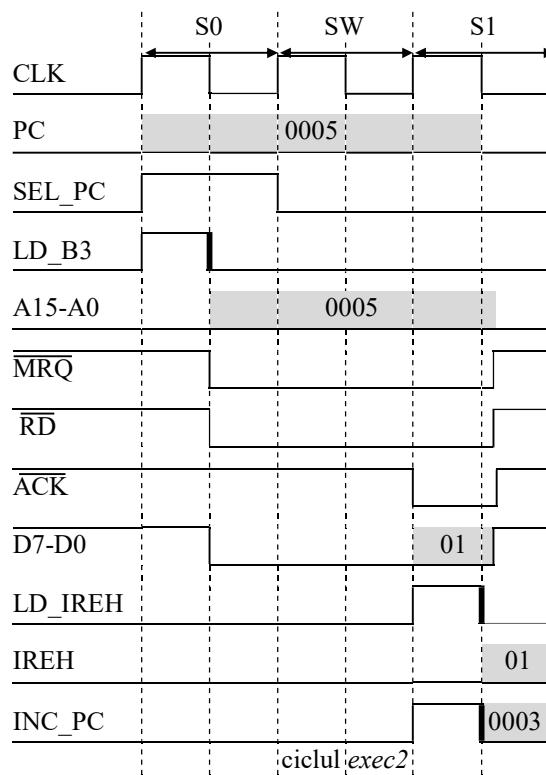


Fig. 6.4.4_2. Cronograma ciclului *execuție 2* al instrucției "JUMP aaaa"
cu aaaa = 0100H.

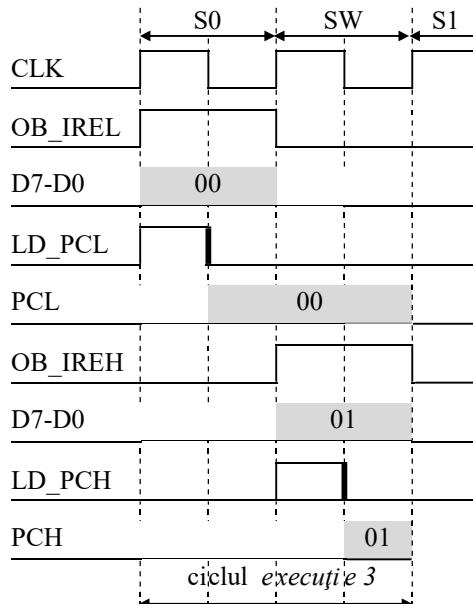


Fig. 6.4.4_3. Cronograma ciclului *execuție 3* al instrucției “**JUMP aaaa**” cu $aaaa = 0100H$.

6.4.5. Implementarea instrucției “ADD A, @X”

Să presupunem, în continuare, că în zona de memorie $0100H$ - $0105H$ avem situația surprinsă de figura 6.4.5_1. Mai presupunem că în X avem adresa $1234H$, că în locația de adresă $1234H$ avem valoarea $56H$, iar în A, valoarea $78H$.

Adresa [în hexa]	Conținuturile locațiilor cu adresele date de col (1) [în hexa]	Semnificația informației din col (2)	Instrucțiile aflate în locațiile cu adresele date de col (1)
(1)	(2)	(3)	(4)
0100	44	octet 1	ADD A, @X
0101	1E	octet 1	
0102	00	octet 2	CALL 1000H
0103	10	octet 3	
0104	E6	octet 1	OUT @21H, A
0105	21	octet 2	

Fig. 6.4.5_1. Conținutul zonei de memorie de adrese $0100H$ - $0105H$.

Așa cum s-a văzut în subcapitolul precedent, “programul” nostru a ajuns la rularea instrucției de la adresa 0100H în urma execuției instrucției **“JP 0100H”**. Așadar, suntem imediat după ciclul *execuție 3* al acestei instrucții.

În continuare, se va executa un ciclu *fetch* prin care, de la adresa 0100H, în registrul **IR** va fi adus octetul 44H, adică: codul instrucției **“ADD A, @X”**.

Această instrucție este pe 1 octet. Execuția ei presupune, mai întâi, transferul operandului 1, aflat în registrul **A**, în registrul tampon **B1**, lucru care se face cu ajutorul semnalelor **OB_A** și **LD_B1**, iar apoi citirea operandului 2 din memorie, de la adresa din **X**, aducerea lui la intrarea a doua a unității aritmetico-logice, comanda efectuării de către unitatea aritmetico-logică a unei operații de adunare în timp ce informația citită din memorie este stabilă pe liniile de date –adică: în timp ce semnalul **ACK** este activ-, preluarea rezultatului adunării în registrul **B2** și, în final, transferul său din **B2** în **A**.

Întrucât prima parte a operației de citire din memorie nu necesită magistrala de date internă a procesorului, transferul operandului 1 din **A** în **B1** se poate face în paralel cu ea. Se mai face precizarea că deodată cu încărcarea rezultatului în registrul **B2**, se efectuează și încărcarea informațiilor despre rezultat în registrul **F**, informații furnizate automat de către unitatea aritmetico-logică.

Figura 6.4.5_1 și 6.4.5_2 ilustrează la nivel de cronogramă ceea ce s-a prezentat în rândurile de mai sus.

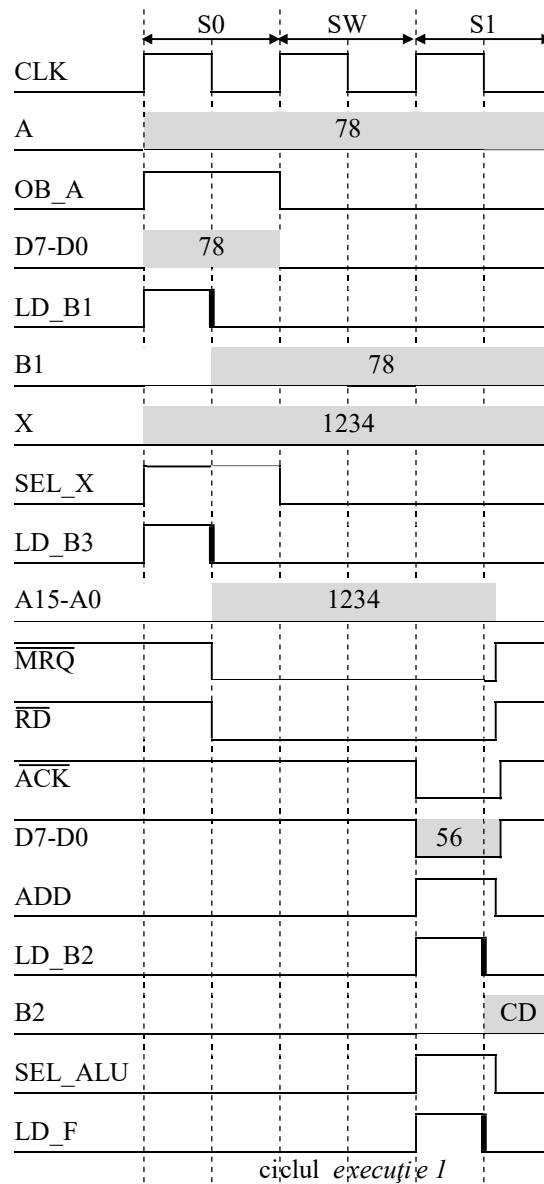


Fig. 6.4.5_2. Cronograma ciclului de execuție 1 al instrucției “ADD A, @X”, cu A inițial = 78H, X = 1234H și @1234H = 56H.

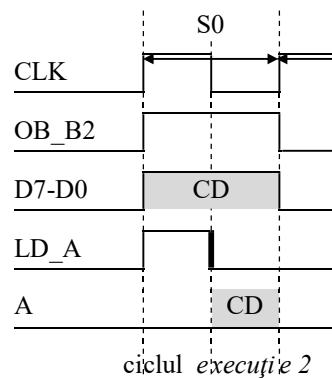


Fig. 6.4.5_3. Cronograma ciclului de execuție 2 al instrucției “ADD A, @X”, cu A inițial = 78H, X = 1234H și @1234H = 56H.

6.4.6. Implementarea instrucției “CALL aaaa”

L-a întocmirea figurii 6.4.5_1, s-a presupus că în zona de memorie considerată, imediat după instrucția **“ADD A, @X”** –adică: începând de la adresa 0101H- se află instrucția **“CALL 1000H”**. Drept urmare, după execuția instrucției **“ADD A, @X”**, “programul” evoluează după cum urmează.

Se execută, mai întâi, un ciclu *fetch* cu referire la adresa 0101H. Prin acesta, în registrul **IR** se aduce octetul 1EH, adică: codul instrucției **“CALL aaaa”**.

Instrucția **“CALL aaaa”** fiind una pe 3 octeți, vor urma alte 2 cicluri de citire, primul dintre ele identic cu ciclul *execuție 1* al instrucției **“JUMP aaaa”**, iar al doilea aproape identic cu ciclul *execuție 2* al instrucției **“JUMP aaaa”**; “aproape identic” și nu “identic” deoarece în ciclul *execuție 2* al instrucției **“CALL aaaa”** se execută, în plus față de ceea ce se întâmplă în ciclul *execuție 2* al instrucției **“JUMP aaaa”**, decrementarea registrului **SP**, în vederea iminentei campanii de scriere în stivă, sub asistența sa, a conținutului registrului **PC**, înainte de încărcarea acestui registru cu adresa subrutinei la care se face apelul.

Salvarea în stivă a lui **PC** se face în 2 cicluri: mai întâi, se va salva **PCH**, iar apoi, după o nouă decrementare a lui **SP**, realizată la finele primului ciclu, se va salva **PCL**. Cele 2 cicluri se vor numi “*execuție 3*”, respectiv “*execuție 4*”.

Execuția instrucției **“CALL aaaa”** se încheie cu un ciclu “*execuție 5*” identic cu ciclul *execuție 3* al instrucției **“JUMP aaaa”**, adică: cu un ciclu intern de transfer al informației din registrele **IREL** și **IREH**, în registrele **PCL**, respectiv **PCH**.

Cronogramele ciclurilor *execuție 1*, *execuție 2*, *execuție 3*, *execuție 4* și *execuție 5* ale instrucției **“CALL aaaa”** sunt redate în figurile 6.4.6_1, 6.4.6_2, 6.4.6_3, 6.4.6_4 și 6.4.6_5.

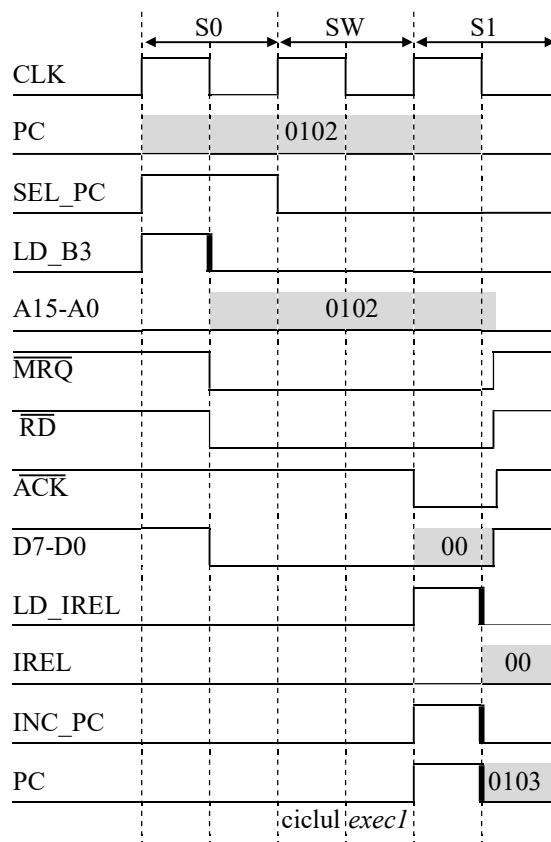


Fig. 6.4.6_1. Cronograma ciclului *execuție I* al instrucției “CALL aaaa”
cu aaaa = 1000H

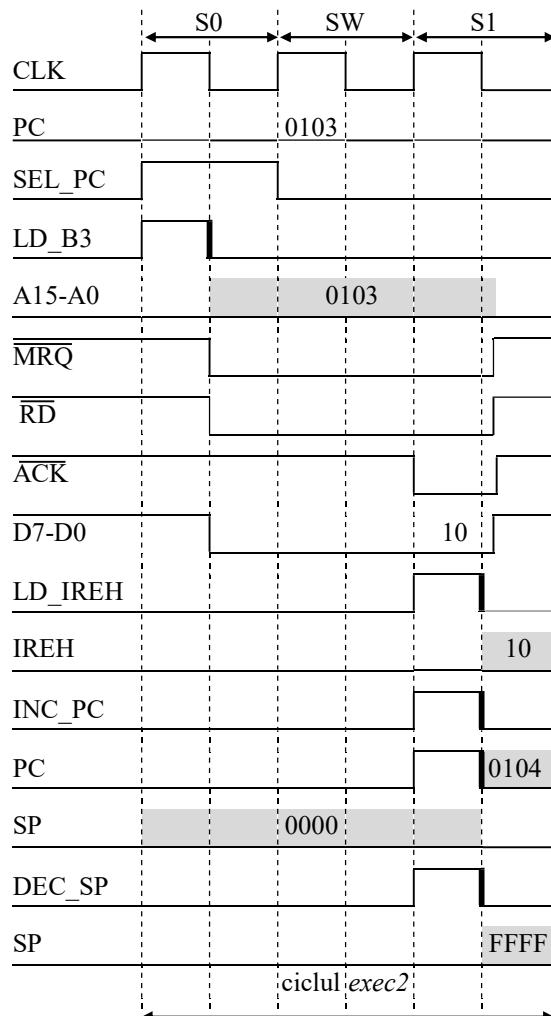


Fig. 6.4.6_2. Cronograma ciclului *execuție 2* al instrucțiiei “CALL aaaa”
cu aaaa = 1000H

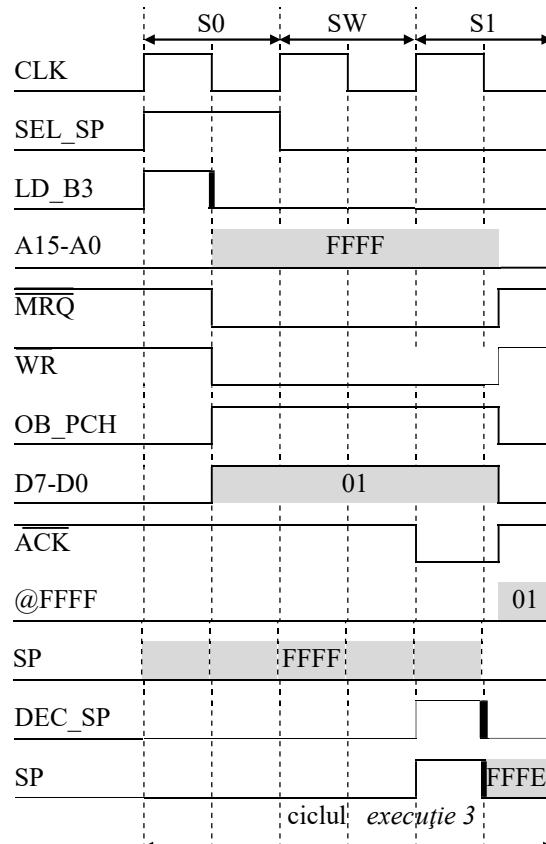


Fig. 6.4.6_3. Cronograma ciclului execuție 3 al instrucției "CALL aaaa" cu aaaa = 1000H

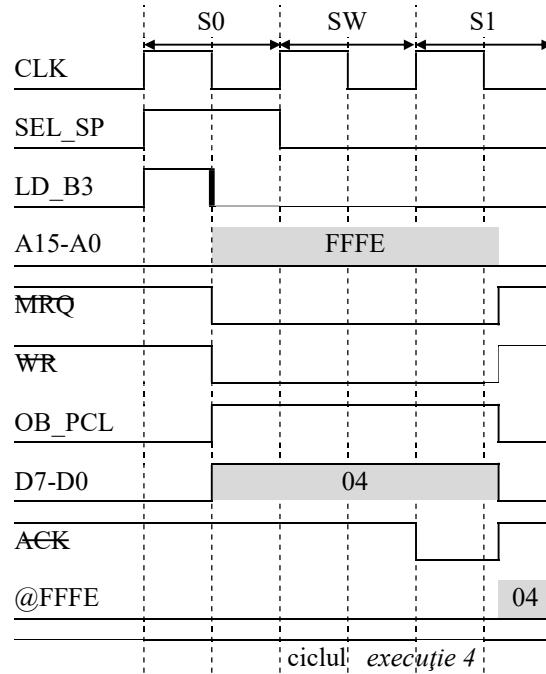


Fig. 6.4.6_4. Cronograma ciclului execuție 4 al instrucției "CALL aaaa" cu aaaa = 1000H

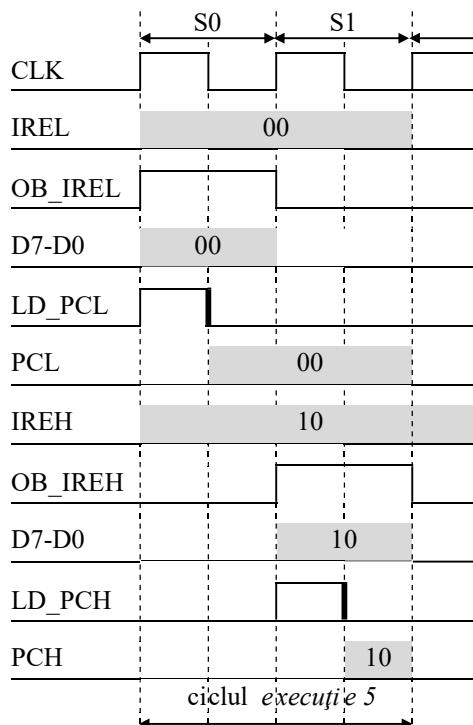


Fig. 6.4.6_5. Cronograma ciclului *execuție 5* al instrucției “**CALL aaaa**” cu $\text{aaaa} = 1000\text{H}$

În concluzie, se punctează faptul că valoarea care se va găsi înscrisă în registrul **PC** la sfârșitul execuției instrucției **“CALL 1000H”**, este 1000H , în timp ce în vîrful stivei se găsește valoarea 0104H , reprezentând adresa instrucției aflată topologic imediat după instrucția **CALL**, adică: adresa la care trebuie revenit din subrutină.

6.4.7. Implementarea instrucției “RET”

Așa cum s-a văzut, efectul execuției instrucției **“CALL 1000H”**, amplasată în memorie la adresele 0101H - 0103H , a fost, pe de o parte, salvarea în stivă a adesei 0104H , corespunzătoare instrucției aflată, din punct de vedere topologic, imediat după instrucția **CALL 1000H**, iar pe de altă parte, încărcarea registrului **PC** cu valoarea 1000H , reprezentând adresa primei instrucții din subrutina apelată. În cadrul subroutinei, la un moment, se va întâlni instrucția **RET**. Prezența acesteia este obligatorie pentru revenirea în secțiunea de program de unde subrutina a fost apelată. Să vedem cum se implementează această instrucție.

Instrucția **RET** este o instrucție pe 1 octet. După parcurgerea ciclului *fetch* cu referire la adresa unde ea se găsește, în registrul **IR** se va avea valoarea 0E7H. Execuția propriu-zisă a instrucției **RET** presupune efectuarea a două cicluri de citire din memorie din zona gestionată după principiul stivei –asta, bineînțeles, sub asistența registrului **SP**- și transferarea informației citite în registrul **PC**. Conform principiilor lucrului cu stiva, fiecare citire din stivă presupune o postincrementare a registrului **SP**, așa cum fiecare scriere în stivă presupune o predecrementare a registrului **SP**.



Cronogramele celor 2 cicluri de execuție ale instrucției **RET** sunt redate în figurile 6.4.7_1, respectiv 6.4.7_2.

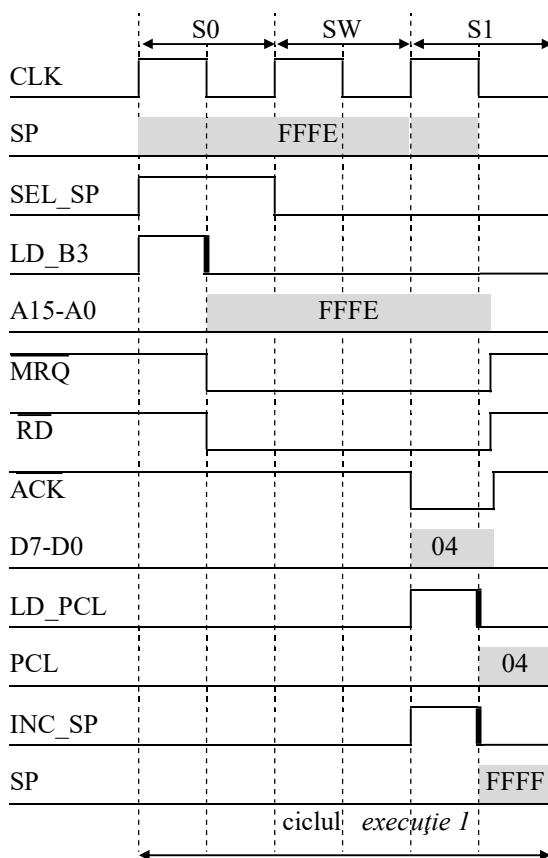


Fig. 6.4.7_1. Cronograma ciclului *execuție I* al instrucției “RET”.

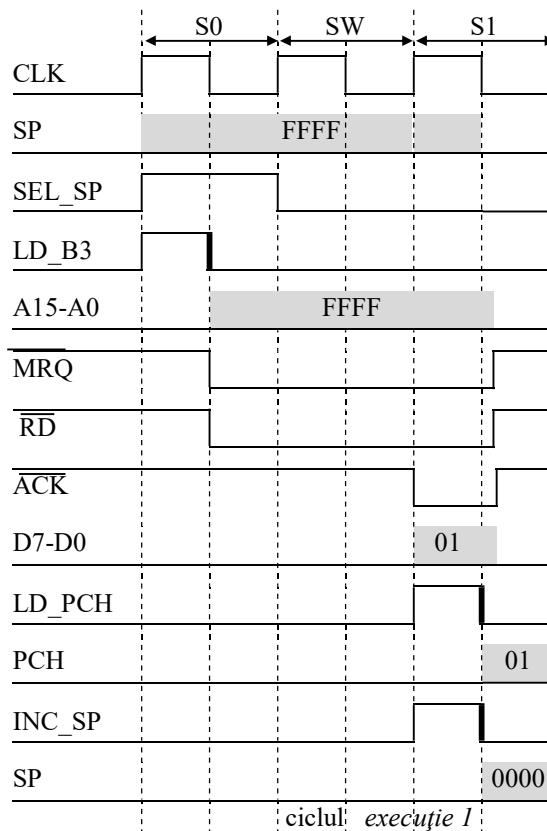


Fig. 6.4.7_2. Cronograma ciclului *execuție 2* al instrucției “RET”.

În concluzie, se punctează faptul că valoarea care se va găsi înscrisă în registrul **PC** la sfârșitul execuției instrucției **“RET”** în subrutina apelată prin instrucția **“CALL 1000H”** amplasată în memorie la adresele 0101H - 0103H este 0104H.

6.4.8. Implementarea instrucției “OUT @aa, A”

În figura 6.4.5_1, s-a presupus că “programul” considerat are de rulat, după subrutina de la adresa 1000H, o instrucție de tip **OUT**, și anume: **“OUT 21H, A”**.

Instrucția **“OUT 21H, A”** este o instrucție pe 2 octeți. După efectuarea ciclului *fetch* cu referire la adresa 0104H, în registrul **IR** se va găsi valoarea E6H, reprezentând octetul 1 al unei instanțe a acestei instrucții, ce vizează portul de ieșire de adresă 21H. Execuția instrucției **“OUT 21H, A”** presupune, pe de o parte, efectuarea unui ciclu de citire din memorie sub asistența lui **PC**, în scopul aflării adresei portului vizat, cuprinsă în octetul 2 al instrucției, iar pe de altă parte, efectuarea unui ciclu de scriere în portul

cu adresa respectivă, a conținutului registrului A. Notăm cu “*execuție 1*”, respectiv “*execuție 2*” cele 2 cicluri.

Dacă până acum, în toate operațiile de citire sau scriere, a fost activat semnalul \overline{MRQ} , la scrierea într-un port –lucru valabil, evident, și pentru citirea dintr-un port-, locul acestui semnal va fi luat de semnalul \overline{IORQ} .

Cronogramele ciclurilor *execuție 1* și *execuție 2* sunt redate în figura 6.4.8_1, respectiv 6.4.8_2.

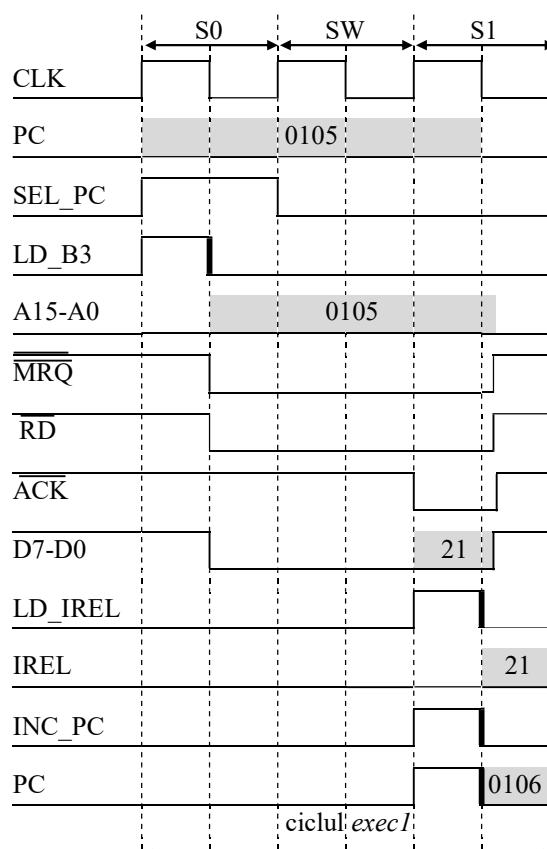


Fig. 6.4.8_1. Cronograma ciclului *execuție 1* al instrucției “OUT aa” cu aa = 21H

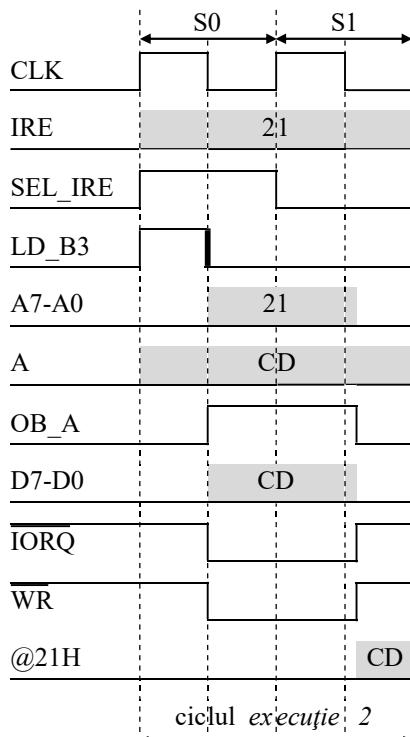


Fig. 6.4.8_2. Cronograma ciclului *execuție 2* al instrucției “**OUT aa**” cu aa = 21H și presupunând că în A se are valoarea hexazecimală CD

Este de remarcat faptul că din cronograma ciclului *execuție 2* lipsește semnalul $\overline{\text{ACK}}$. Explicația rezidă în aceea că porturile au timpi de răspuns foarte reduși, fiind posibilă conlucrarea dintre ele și procesor fără introducerea de stări de așteptare și, în consecință, fără a se mai folosi semnalul $\overline{\text{ACK}}$. Negenerarea semnalului $\overline{\text{ACK}}$ de către unitatea de porturi face necesară aplicarea automată la intrarea cu același nume a procesorului a semnalului $\overline{\text{IORQ}}$ însuși.

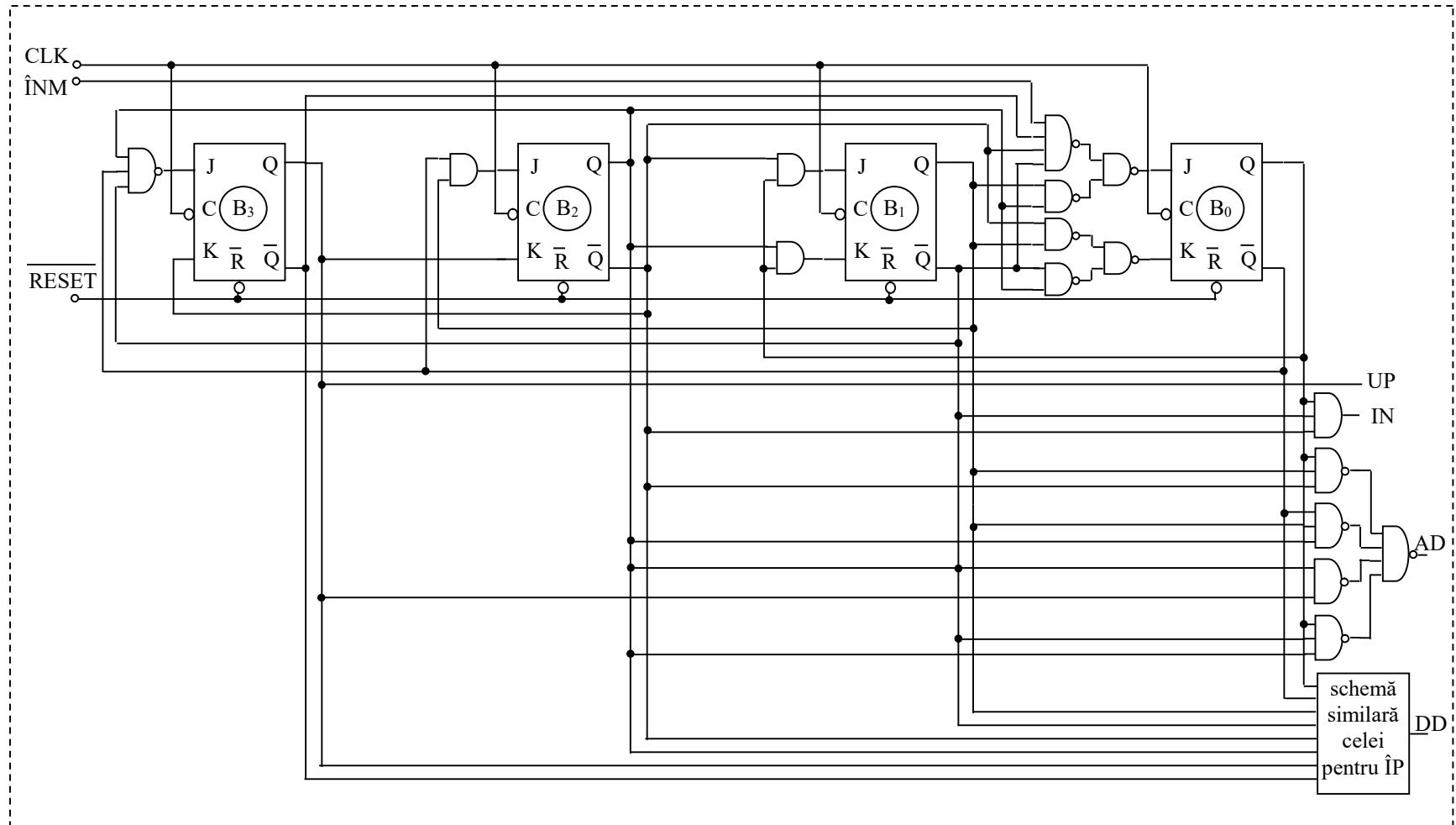


Fig. 4.3.2_20. Schema blocului de comandă al înmulțitorului în semn-mărime.

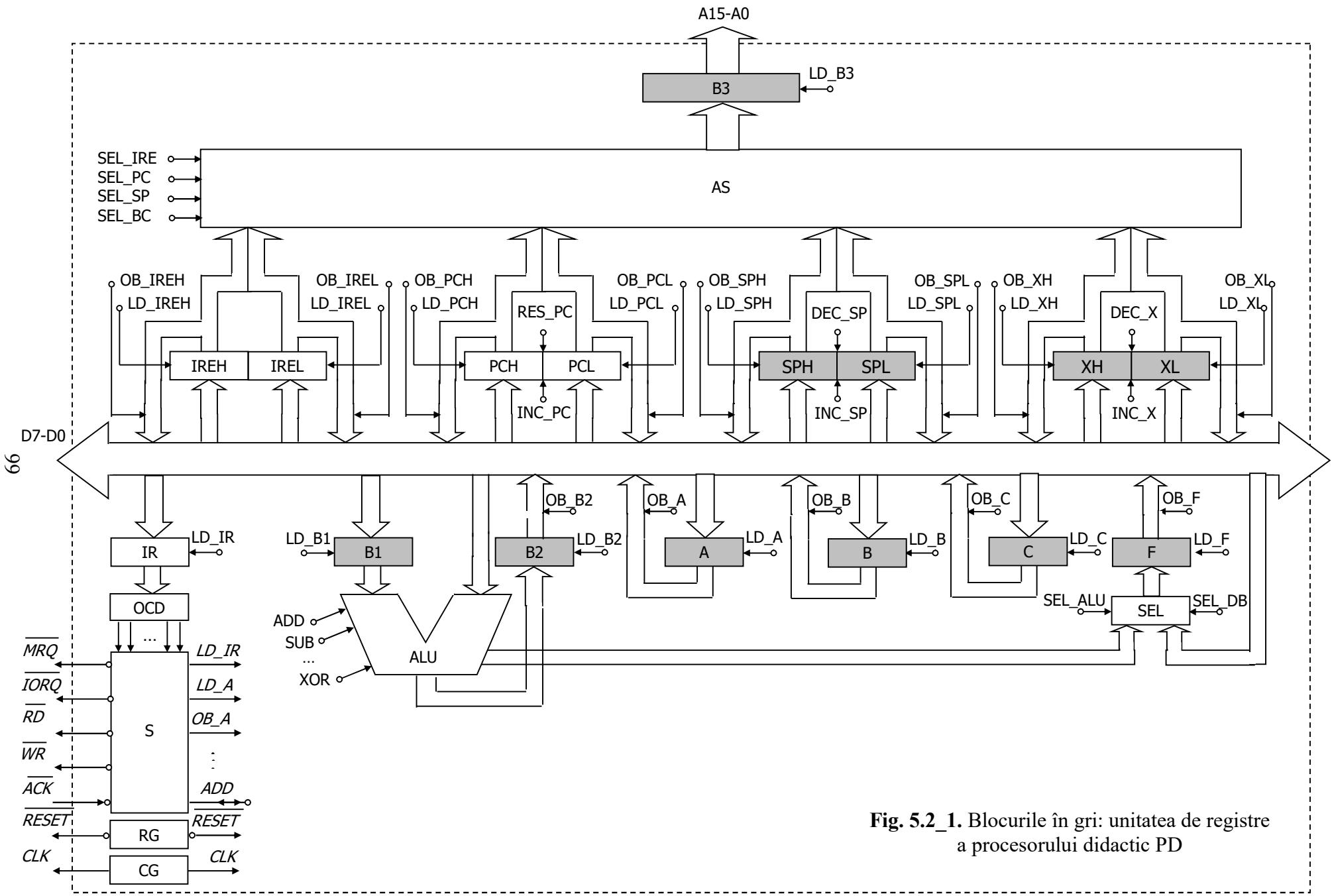


Fig. 5.2_1. Blocurile în gri: unitatea de registre a procesorului didactic PD

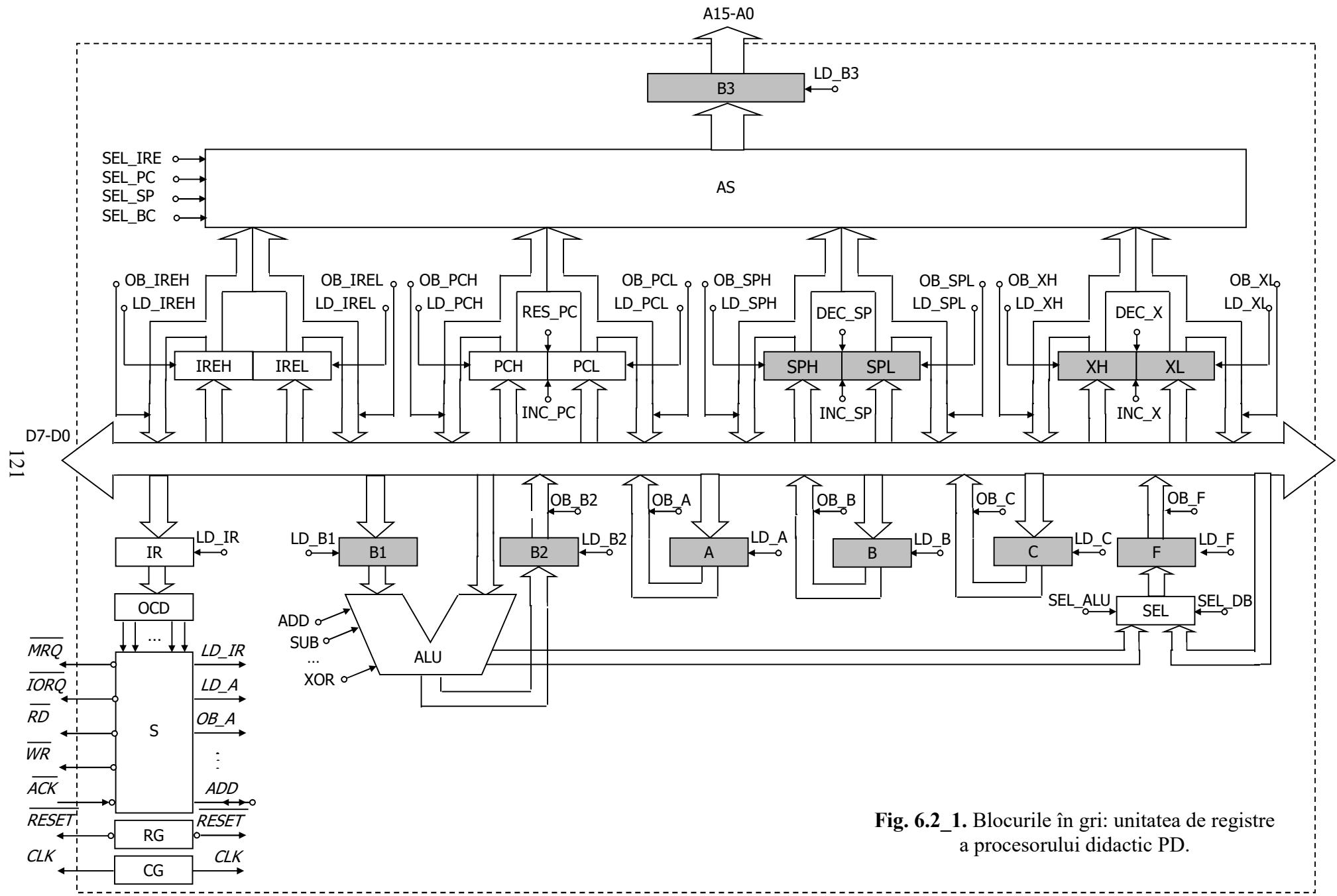


Fig. 6.2_1. Blocurile în gri: unitatea de registre a procesorului didactic PD.