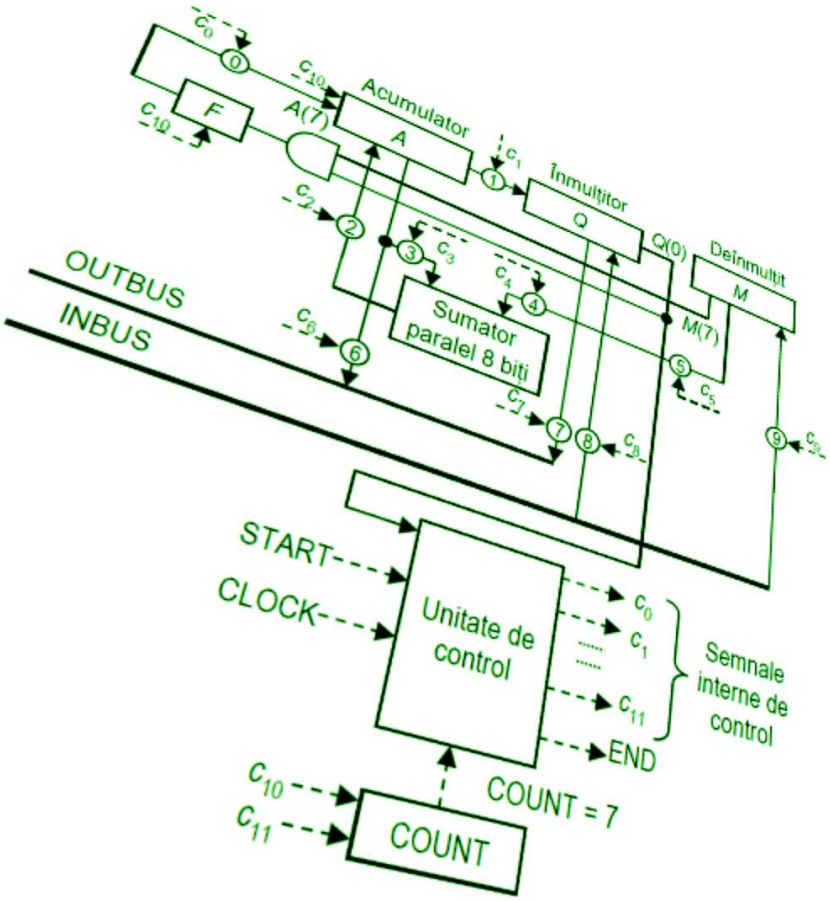
Mircea-Petru Ursu
Daniela E. Popescu



STRUCTURA ŞI ORGANIZAREA CALCULATOARELOR

EDITURA UNIVERSITĂȚII DIN ORADEA

Referenți:
Prof. univ. em. dr. ing. Mircea Vlăduțiu, Universitatea Politehnică Timișoara
Prof. univ. dr. ing. Ștefan Vari-Kakas, Universitatea din Oradea

ISBN 978-606-10-2148-2

Tehnoredactare computerizată: ș. l. fiz. dr. Mircea-Petru Ursu

Coperta: ș. l. fiz. dr. Mircea-Petru Ursu

Mircea-Petru Ursu
Daniela E. Popescu

STRUCTURA ŞI ORGANIZAREA CALCULATOARELOR

Îndrumător de laborator

pentru uzul studenților

Editura Universității din Oradea
2021

Prefață

STRUCTURA ȘI ORGANIZAREA CALCULATOARELOR – îndrumător de lucrări de laborator cuprinde un număr de 10 lucrări de laborator orientate spre însușirea conceptelor de proiectare a sistemelor numerice, în general, și a celor de calcul, în special. Pentru exemplificarea metodelor de proiectare se utilizează un exemplu de sistem de înmulțire a două numere binare și un sistem de calcul cu structură bazată pe acumulator. Proiectarea ei se face atât în variantă cablată, cât și în variantă microprogramată.

Lucrările de laborator, realizate pe baza combinării resurselor bibliografice cu studiile, experiența și ideile proprii ale autorilor, au următoarele titluri: *Reprezentarea informației în sistemele de calcul, definiții, algoritmi* (capitol introductiv); *Metode de descriere a comportării sistemelor utilizate în proiectare*; *Tehnici de proiectare a unității de procesare a datelor într-un sistem de calcul*; *Metoda tabelului de stare pentru proiectarea unității de control cablate a sistemelor digitale*; *Metodă de proiectare a unității de control cablate bazată pe utilizarea elementelor de întârziere pentru secvențierea semnalelor de comandă*; *Metodă de proiectare a unității de control bazată pe utilizarea numărătoarelor pentru secvențierea în timp a semnalelor de control*; *Proiectarea unității de control cablate a unității centrale de prelucrare a unui calculator cu structură bazată pe acumulator*; *Proiectarea unității de procesare a datelor pentru o unitate centrală de prelucrare a unui calculator cu structură bazată pe acumulator*; *Proiectarea unității de control microprogramate pentru un dispozitiv de înmulțire a două numere în reprezentare cu virgulă fixă*; *Proiectarea unității de control microprogramate pentru unitatea centrală de prelucrare a unui calculator cu structură bazată pe acumulator*; *Proiectarea unității de procesare a datelor pentru unitatea centrală de prelucrare a unui calculator microprogramat cu structură bazată pe acumulator*.

Datorită exemplelor care însoțesc lucrările de laborator propuse, cartea are un puternic caracter aplicativ, practic și ingineresc, fiind deosebit de utilă studenților de la Departamentul de Calculatoare și Tehnologie a Informației, dar și tuturor acelora care doresc să-și aprofundeze cunoștințele legate de structura, organizarea și arhitectura sistemelor de calcul. Imaginile sunt realizate de autori cu ajutorul facilităților grafice ale aplicației Microsoft Word și cu aplicația online CircuitVerse.

CUPRINS

Reprezentarea informației în sistemele de calcul, definiții, algoritmi

Scopul acestui capitol constă în familiarizarea studenților cu modul de reprezentare a informației în sistemele de calcul, insistându-se asupra operațiilor de conversie între formatele de reprezentare. Pentru fiecare format numeric în parte se va urmări modul în care se realizează operațiile de adunare și de scădere la nivelul sistemului. De asemenea, se va defini sistemul de calcul, se va descrie structura calculatorul IAS și vor fi ilustrați câțiva algoritmi bazați pe această structură.

Reprezentarea informației în sistemele de calcul

În sistemele de calcul informația este reprezentată sub formă de secvențe binare, organizate de obicei sub formă de cuvinte. Un *cuvânt* este o unitate de informație de lungime fixă *n*, unde *n* este determinat de considerente de cost hardware.

În prezent, pentru reprezentarea caracterelor este răspândită utilizarea secvențelor binare de 8 cifre binare, numite *octeți*, care permit reprezentarea codificată a literelor mari, a literelor mici, a altor caractere speciale, precum și reprezentarea eficientă a numerelor în format BCD (într-un câmp de un octet pot fi memorați doi digiți BCD – ***B**inary **C**oded **D**ecimal*). Calculatoarele moderne au lungimea cuvintelor multiplu de 8, uzual fiind întâlnite cuvinte de 8, 16, 24, 32 sau 64 cifre binare.

Două coduri standard de 8 biți/caracter cunosc o largă utilizare:

- EBCDIC (***E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode*), dezvoltat de IBM;
- ASCII (***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*), dezvoltat sub auspiciile unui comitet al Asociației Americane a Standardelor (***A**merican **S**tandards **A**ssociation* – ASA), devenite între timp Institutul Național American al Standardelor (***A**merican **N**ational **S**tandards **I**nstitute* – ANSI).

În continuare sunt reprezentate schematic tipurile de informație din sistemele de calcul:

<table>
<tr><td rowspan="6">INFORMAȚII</td><td colspan="4">Instrucțiuni</td></tr>
<tr><td rowspan="5">Date</td><td colspan="3">Nenumerice</td></tr>
<tr><td rowspan="4">Numerice</td><td rowspan="2">În virgulă fixă</td><td>Binare</td></tr>
<tr><td>Zecimale</td></tr>
<tr><td rowspan="2">În virgulă flotantă</td><td>Binare</td></tr>
<tr><td>Zecimale</td></tr>
</table>

Fundamental, informațiile se împart în *instrucțiuni* și *date*. Datele pot fi atât *numerice*, cât și *nenumerice*. Datorită importanței calculului numeric, s-a acordat o atenție deosebită dezvoltării codurilor numerice și au fost dezvoltate două formate majore: formatul *în virgulă fixă* și formatul *în virgulă flotantă*. Formatul în virgulă fixă pe *n* biți are forma:

$$b_{n-1}\ldots b_3 b_2 b_1 b_0, \text{ unde } b_i \text{ este fie 0, fie 1}$$

și o virgulă (imaginară!) care separă partea întreagă de partea fracționară, fixată într-o poziție dată și implicită. Bitul cel mai din stânga este numit *bitul cel mai semnificativ* (**MSB** – ***M**ost **S**ignificant **B**it*), iar bitul cel mai din dreapta este numit *bitul cel mai nesemnificativ* (**LSB** – ***L**east **S**ignificant **B**it*).

Formatul de reprezentare în virgulă flotantă este alcătuit dintr-o pereche de numere în virgulă fixă

$$M, E$$

care reprezintă numărul $M \times B^E$, unde B este o bază predeterminată. Acest format de reprezentare corespunde notației științifice.

Pentru reprezentarea numerelor în virgulă fixă sunt utilizate diverse codificări. Acestea pot fi clasificate în *codificări binare* (de exemplu, codul complement față de 2) și *codificări zecimale* (de exemplu, codul BCD). Sistemele de calcul mari utilizează în mod frecvent mai multe formate diferite în virgulă fixă și în virgulă flotantă.

Datele nenumerice se reprezintă sub forma unor șiruri de caractere de lungime variabilă, codificate în cod ASCII sau EBCDIC.

Formatul numerelor

Pentru alegerea formatului numerelor care vor fi reprezentate într-un sistem de calcul se iau în considerare următorii factori:

- ✓ *tipul numerelor* care vor fi reprezentate, adică numere întregi, numere reale, numere complexe etc.;
- ✓ *domeniul de valori* al numerelor cu care se va opera;
- ✓ *precizia* cerută numerelor, adică acuratețea maximă de reprezentare;
- ✓ *costul hardware-ului* implicat pentru memorarea și procesarea numerelor.

Există două formate principale de reprezentare: în *virgulă fixă* sau în *virgulă flotantă*. Formatele de reprezentare în virgulă fixă permit un domeniu limitat de valori și au cerințe hardware relativ simple. Pe de altă parte, numerele în virgulă flotantă permit un domeniu de valori mult mai larg, dar implică un hardware mai costisitor pentru procesare.

Numere în virgulă fixă

Formatul de reprezentare în virgulă fixă este derivat direct din reprezentarea zecimală a numerelor, fiind format dintr-o secvență de biți separați printr-o virgulă. Biții din partea stângă a virgulei reprezintă *partea întreagă* și biții din partea dreaptă reprezintă *partea fracționară* a numărului. Aceasta este o notație *pozițională*, în cadrul căreia fiecare bit are o pondere fixată, în funcție de poziția sa față de virgulă. Sistemele de calcul utilizează o notație pozițională având baza 2. Secvența binară de forma următoare, unde $P + F = n$,

$$b_{P-1} \dots b_3 b_2 b_1 b_0 b_{-1} b_{-2} b_{-3} b_{-4} \dots b_{-F}$$

reprezintă numărul:

$$\sum_{i=-F}^{P-1} b_i \cdot 2^i$$

Acest format este utilizat pentru reprezentarea *numerelor fără semn*, considerate pozitive. Pentru reprezentarea *numerelor negative* există mai multe formate distincte, care vor fi discutate mai târziu.

Numărul de valori distincte care pot fi reprezentate este 2^n. Valoarea maximă se obține trecând pe 1 toți biții numărului, iar precizia se obține anulând toți biții numărului cu excepția celui mai nesemnificativ. Astfel, valoarea maximă care poate fi reprezentată este

$$2^{n-F} - 2^{-F}$$

iar precizia de reprezentare („pasul”) este

$$2^{-F}$$

Virgula **NU** este reprezentată în mod explicit, ea având o poziție fixă și implicită în cadrul cuvântului. Această poziție a virgulei nu este deosebit de importantă din punct de vedere a proiectării.

De exemplu, în cazul specificațiilor $n = 8$ și $F = 3$, formatul numerelor în virgulă fixă este:

$$b_4 b_3 b_2 b_1 b_0 b_{-1} b_{-2} b_{-3}$$

Acest format asigură $2^8 = 256$ valori distincte cuprinse între 0 (toți biții anulați) și $2^5 - 2^{-3} = 31.875$ (toți biții trecuți pe 1), cu precizia reprezentării $2^{-3} = 0.125$. Dacă nu se alocă biți pentru partea fracționară, deci se reprezintă numai numere întregi, atunci specificațiile devin $n = 8$ și $F = 0$. Formatul asigură tot $2^8 = 256$ valori distincte, dar care sunt acum cuprinse între 0 și $2^8 - 2^0 = 255$ cu precizia $2^0 = 1$.

Atunci când se procesează numai numere întregi, virgula se află în dreapta bitului cel mai puțin semnificativ b_0, obținându-se formatul întreg de reprezentare a numerelor în virgulă fixă. Conform formulelor de mai sus, cu acest format întreg de *n* biți se pot reprezenta toți întregii *N* din domeniul $0 \leq N \leq 2^n - 1$.

Un alt format larg utilizat tratează formatul figurat ca o fracție cu virgula poziționată între b_0 și b_{-1}, unde b_0 este bitul cel mai semnificativ (MSB). Acest format fracție pe *n* biți permite reprezentarea numerelor *X* din domeniul de valori:

$$0 \leq X \leq 2 - 2^{-n+1}$$

Reprezentarea numerelor negative

Presupunem că numerele în virgulă fixă sunt reprezentate sub forma unui cuvânt de *n* biți, deci un astfel de număr este $X = x_{n-1}x_{n-2}\ldots x_2x_1x_0$. Cea mai naturală reprezentare a numerelor negative constă în utilizarea bitului x_{n-1} ca *bit de semn*, atribuindu-i-se valoarea 0 pentru semnul plus și 1 pentru semnul minus, ceilalți biți păstrându-și rolurile deja cunoscute, fiind denumiți *biți de mărime*. Acest format este denumit ***semn-mărime***. Totuși, pe lângă avantajul simplității reprezentării, acest format are dezavantaje, cum sunt dificultatea implementării operațiilor aritmetice, reprezentarea diferită a numărului 0 în funcție de semn etc.

Au fost dezvoltate și alte coduri, toate având aceeași reprezentare pentru numerele pozitive ca formatul semn-mărime, dar cu alte reprezentări pentru numerele negative.

Ne vom referi la același număr *X* de mai sus, pozitiv cu *n* biți. În ***codul complement față de 1,*** $-X$ este definit ca $\overline{X}$, respectiv complementul bit cu bit al numărului *X*. În ***codul complement față de 2***, $-X$ este definit ca fiind $\overline{X} + 1$. În ambele coduri complement x_{n-1} rămâne *bitul de semn*, dar în cazul numerelor negative biții de mărime nu mai formează un simplu cod pozițional. Aceste coduri au mai rezolvat din dificultățile sus-menționate, dar codul complement față de 2 a rezolvat inclusiv reprezentarea corectă a numărului 0. Astfel, la ora actuală singurul cod recunoscut pentru reprezentarea numerelor binare negative este ***codul complement față de 2***. Principalul avantaj este acela că scăderea implică doar complementarea scăzătorului și adunarea lui la descăzut. Astfel, scăderea lui *X* din *Y* calculează astfel: $\overline{X} + 1 + Y$.

Există două metode pentru obținerea reprezentării lui în cod complement față de 2 a unui număr pozitiv X:

Metoda I

pas 1: Se inversează fiecare bit a lui X pentru a forma $\overline{X}$ (cod complement față de 1).

pas 2: Se adună 1 la bitul cel mai puțin semnificativ al lui $\overline{X}$.

Metoda II

Se copiază de la dreapta spre stânga toți biții numărului pozitiv X până la întâlnirea primului 1 inclusiv, după care toți biții numărului X se inversează.

ex : $76_{10} = 01001100_2$

$-76_{10} = 10110100_2$

Observație: În calcul, bitul de semn se tratează exact ca și ceilalți biți.

În **Fig. 1** este arătată reprezentarea întregilor utilizând cele trei codificări pentru cazul când $n = 4$.

Reprezentare zecimală	Semn-mărime	Complement față de 1	Complement față de 2
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	0000
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001

Fig. 1. Reprezentări ale numerelor întregi pe 4 biți.

Aceste coduri sunt numite *coduri binare*. De observat faptul că, în cazul codului complement față de 2, complementul lui 0000 este tot 0000, ceea ce este un avantaj pentru implementarea instrucțiunilor care fac testarea lui 0.

Numere zecimale

Toate informațiile sunt reprezentate în sistemele de calcul în cod binar. Astfel, toate numerele introduse în sistemele de calcul trebuie convertite din format zecimal în format binar și, evident, trebuie convertite din format binar în format zecimal pentru procesul de extragere a rezultatelor. Astfel, numărul acestor conversii tinde să reprezintă o fracțiune importantă din totalul operațiilor elementare realizate de sistem, deci, pentru performanțele sistemului, este important ca aceste conversii să se facă cât mai rapid. Codurile binare prezentate mai sus nu duc la ușurarea și creșterea în viteză a procesului de conversie.

Pentru realizarea mai rapidă a conversiilor binar-zecimale și invers se utilizează așa zisele *coduri zecimale*, care realizează codificarea separată a fiecărui digit zecimal printr-o secvență de biți. În reprezentarea BCD („***B**inary **C**oded **D**ecimal*") a unui număr zecimal fiecare digit d_i este reprezentat prin 4 biți $b_{i,3}b_{i,2}b_{i,1}b_{i,0}$. BCD este un cod numeric ponderat fiindcă fiecare bit $b_{i,j}$ are ponderea $10^i \cdot 2^j$. Numerele BCD sunt, în general, în formă semn-mărime.

Codurile de 8 biți, EBCDIC și ASCII, reprezintă cei 4 digiți zecimali într-un câmp de 4 biți, la fel ca și codul BCD. În **Fig. 2** sunt arătate și alte două coduri de importanță moderată, „*exces 3*" și „*doi din cinci*", care pot fi studiate în literatura de specialitate.

Digit zecimal	BCD	EBCDIC	ASCII	Exces trei	Doi din cinci
0	0000	11110000	00110000	0011	11000
1	0001	11110001	00110001	0100	00011
2	0010	11110010	00110010	0101	00101
3	0011	11110011	00110011	0110	00110
4	0100	11110100	00110100	0111	01001
5	0101	11110101	00110101	1000	01010
6	0110	11110110	00110110	1001	01100
7	0111	11110111	00110111	1010	10001
8	1000	11111000	00111000	1011	10010
9	1001	11111001	00111001	1100	10100

Fig. 2. Reprezentări ale numerelor întregi 0-9 în coduri diferite.

Principalul avantaj al codurilor zecimale este dat de ușurința conversiei între reprezentarea internă a sistemului de calcul, care permite numai simboluri 0 și 1, și reprezentarea externă a numerelor, care utilizează cele 10 simboluri: 0, 1, 2, 3, 4, 5, 6, 7, 8 și 9.

Codurile zecimale au două dezavantaje:

1. Utilizează mai mulți biți pentru reprezentarea numerelor decât codurile binare, necesitând astfel mai mult spațiu de memorare. Dacă se utilizează codurile binare, un cuvânt de *n* biți poate reprezenta 2^n numere, pe când dacă se utilizează un cod zecimal pe 4 biți (BCD sau exces trei) se pot reprezenta $10^{n/4} = 2^{0.830 \cdot n}$ numere.
2. Circuitele cerute pentru operarea aritmetică a doi operanzi în cod zecimal sunt mai complexe decât cele cerute de operarea aritmetică binară. De exemplu, pentru adunarea a două numere bit cu bit nu este posibilă o metodă de propagare uniformă a transporturilor între poziții adiacente, întrucât ponderile w_i și w_{i+1} ale biților adiacenți nu diferă printr-un factor constant.

Numerele în virgulă flotantă

Pentru multe aplicații științifice, unde se operează frecvent cu numere foarte mari și/sau foarte mici, domeniul de valori acoperit de formatul cu virgulă fixă nu este suficient. Notațiile științifice permit o reprezentare pentru asemenea numere cu un număr relativ mic de digiți. De exemplu, masa planetei Pământ, $M_{Terra} = 5.972 \cdot 10^{24}$ kg, se reprezintă mult mai ușor în acest format decât în virgulă fixă. Codurile de reprezentare în virgulă flotantă, care se utilizează în procesoarele digitale, sunt versiuni binare ale reprezentării de mai sus.

Unui număr reprezentat în *virgulă flotantă* i se asociază trei numere: *mantisa* (M), *exponentul* (E) și *baza* (B). Împreună, aceste trei componente reprezintă numărul $\boldsymbol{M \times B^E}$. Pentru exemplul de mai sus 5.972 este mantisa, 24 este exponentul și baza este 10. Pentru implementarea reprezentării în sistemele de calcul, mantisa și exponentul se codifică sub formă de numere binare sau zecimale în virgulă fixă, iar baza este fie 2, fie 10. Întrucât baza este constantă, ea nu trebuie inclusă în codul numărului, ținându-se cont de ea prin circuitele care procesează numerele. Astfel, masa planetei Pământ, $M_{Terra} = 5.972 \cdot 10^{24}$ kg, se reprezintă sub forma perechii de numere (5.972, 24).

Deci, un număr în virgulă flotantă se memorează ca o pereche de numere în virgulă fixă:

- mantisa *M* care în general este o fracție, sau un întreg;
- exponentul *E* care este un întreg.

Precizia lui $M \times B^E$ este dată, în primul rând, de numărul de cifre binare utilizate pentru reprezentarea mantisei. *B* și *E* determină domeniul de reprezentare. Formatele în virgulă flotantă sunt utilizate pentru reprezentarea numerelor reale aflate într-un interval continuu $(-R, +R)$.

Deoarece doar un număr finit de numere din interval pot fi reprezentate de format (cel mult 2^n, unde n este lungimea cuvântului în virgulă flotantă) aceste numere sunt distribuite rar în intervalul $(-R, +R)$. Crescându-se *B*, se crește domeniul de reprezentare a numerelor, dar se rarefiază distribuția lor peste domeniu.

Sistemul de calcul – definiție, descriere

Un *sistem de calcul* se definește ca fiind o colecție de obiecte, numite *componente*, interconectate pentru a forma o entitate coerentă cu o funcție bine definită. Un exemplu de sistem de calcul este constituit de computerele (calculatoarele) folosite astăzi în aproape orice activitate.

Proprietățile principale ale unui sistem de calcul sunt *structura* și *comportarea* sa. *Descrierea structurală* se referă la componența sistemului, iar *descrierea funcțională* la comportarea acestuia. Familiarizarea cu un sistem de calcul necesită cunoașterea ambelor descrieri, ele fiind complementare.

Se definește *structura unui sistem* ca fiind o imagine abstractă care constă din diagramele sale bloc fără informație funcțională. Cu alte cuvinte, descrierea structurală numește componentele sistemului și definește interconexiunile dintre ele.

Comportarea sistemului poate fi reprezentată în diferite feluri. În cazul circuitelor combinaționale descrierea comportării se face prin *tabele de adevăr*. În cazul sistemelor secvențiale ea se poate face sub *formă narativă* sau sub forma *tabelelor de stare* sau a *organigramelor*.

Funcția realizată de sistem este determinată de funcțiile realizate de componentele sale și de interconexiunile acestora. Pe scurt, sistemele de calcul transformă un set *A* de termeni de informație de intrare (de exemplu, un program și seturile sale de date), în informație de ieșire *B* (de exemplu, rezultatele calculate de programul care acționează asupra seturilor sale de date). Această transformare poate fi exprimată printr-o mapare *f* de la *A* la *B*, notată

$$f: A \to B$$

Având definite structura și comportarea unui sistem, se poate defini, în termeni generali, problema cu care se confruntă un proiectant de sistem, și anume: dându-se un domeniu de comportare dorit și un set de componente disponibile (adică *specificațiile sistemului*), trebuie determinată o structură formată din aceste componente, care să realizeze comportarea dorită, la un cost acceptabil.

Un astfel de sistem de calcul este computerul IAS (**Fig. 3**) [1]. Acesta a fost construit la *Institute of Advanced Studies* din Princeton, New Jersey, între anii 1946-1951 sub conducerea celebrului profesor de matematică John von Neumann, cu scopul de a face mai ușoară proiectarea computerelor digitale.

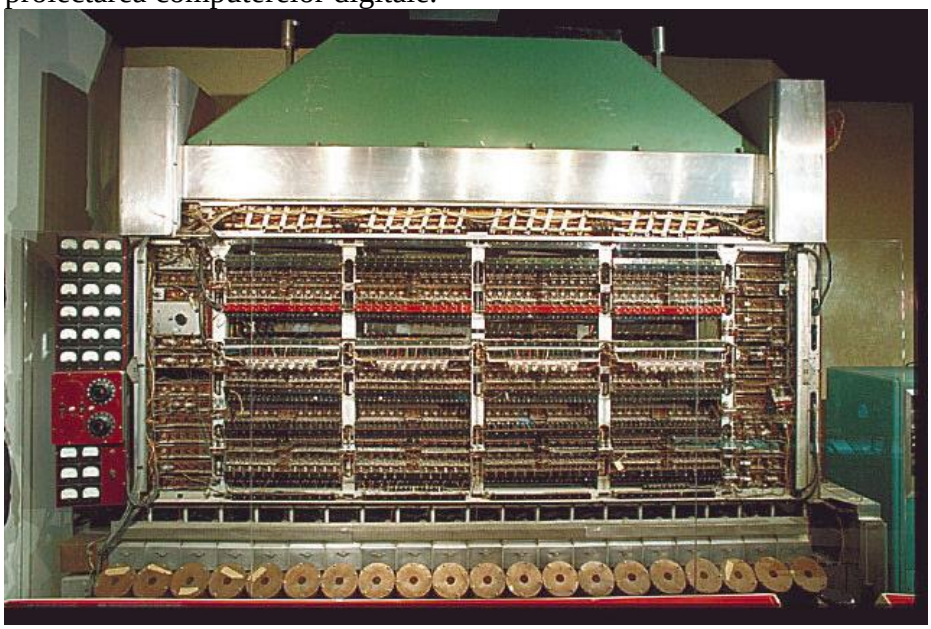


Fig. 3. Computerul IAS. [1]

Conform specificațiilor [2], acest computer are un registru de memorare *M*, un registru de deplasare *Q* și un registru acumulator *A*, o unitate logică asincronă de control, o memorie electrostatică de 1024 de cuvinte, diverse dispozitive magnetice de stocare și dispozitive de intrare-ieșire de tip mașină de scris electronică Teletype. Computerul cunoștea complementul față de 2 pentru reprezentarea numerelor negative și putea lucra cu cuvinte de câte 40 de biți. Comenzile erau codificate în cuvinte de câte 20 de biți, care puteau fi plasate în adresele memoriei două câte două. Computerul era constituit din circa 1700 de tuburi electronice, avea dimensiunile 299.72 cm × 320.04 cm × 83.82 cm și a costat câteva sute de mii de dolari. Acest computer a fost donat celebrului muzeu Smithsonian, unde se află expus în zilele noastre [1].

Algoritmi – descriere, implementare

Ne putem folosi de structura computerului IAS pentru implementarea unor algoritmi simpli și pentru proiectarea unor sisteme de calcul ipotetice care să funcționeze conform acestora. Pentru exemplificare vom utiliza câțiva algoritmi care implică introducerea în sistem a două numere în format binar și prelucrarea acestora prin înmulțire și prin împărțire. Proiectarea structurilor acestor sisteme de calcul constituie obiectul lucrărilor de laborator care urmează.

Pe scurt, înmulțirea înseamnă adunarea repetată a deînmulțitului conform valorii înmulțitorului, iar împărțirea înseamnă scăderea repetată a împărțitorului din deîmpărțit conform valorii acestui factor. Astfel, ambele operații necesită un număr fix de iterații, conform numărului de biți ai operanzilor. Valorile de prelucrat și rezultatele parțiale trebuie memorate în registre, iar pașii algoritmilor se execută în timpul transferurilor între aceste registre. Numărul care rămâne nemodificat (deînmulțitul, respectiv împărțitorul) se memorează în registrul *M*, celălalt număr (înmulțitorul, respectiv deîmpărțitul) se memorează în registrul de deplasare *Q*, iar rezultatele parțiale sunt memorate în registrul acumulator *A*. Registrul *COUNT* numără iterațiile. Datele de prelucrat sunt aduse printr-o magistrală de intrare *INBUS*, iar rezultatele sunt extrase printr-o magistrală de ieșire *OUTBUS*.

Începem cu algoritmul înmulțirii a două numere fără semn pe 8 biți, pe care îl exprimăm atât în *pseudocod*, cât și sub formă de *schemă logică* sau *organigramă* (**Fig. 4**):

```
            Declare registers A[8], M[8], Q[8], COUNT[3];
            Declare buses INBUS[8], OUTBUS[8];
START:      A ← 0, COUNT ← 0, M ← INBUS; [deînmulțit]
            Q ← INBUS; [înmulțitor]
TEST:       if Q(0) = 0 then go to RSHIFT;
ADD:        A ← A + M;
RSHIFT:     AQ ← RIGHTSHIFT (AQ), COUNT ← COUNT + 1;
            if COUNT = 8 then go to FINISH;
            go to TEST;
FINISH:     OUTBUS ← A; [biții cei mai semnificativi]
            OUTBUS ← Q; [biții cei mai puțin semnificativi]
            STOP;
```

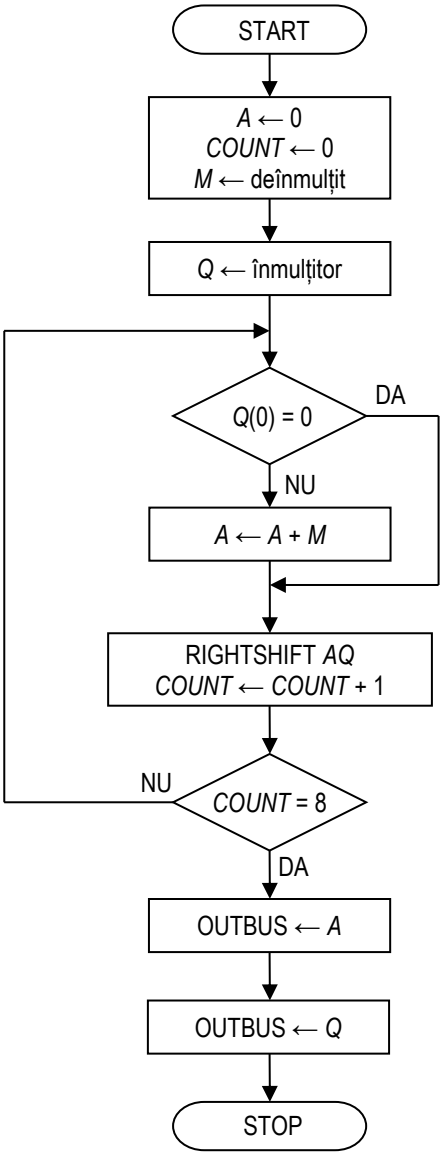


Fig. 4. Organigrama algoritmului înmulțirii a două numere fără semn.

În fiecare iterație, adunarea deînmulțitului (stocat în registrul *M*) peste rezultatul parțial (stocat în registrul *A*) se efectuează numai dacă bitul curent al înmulțitorului (stocat în registrul *Q*) are valoarea 1. Indiferent dacă adunarea parțială s-a efectuat sau nu, conținuturile registrelor *A* și *Q* se translatează spre dreapta (*RIGHTSHIFT*). Astfel, bitul curent al înmulțitorului se va afla totdeauna pe poziția *Q*(0), iar biții deja utilizați ai acestuia sunt eliminați din registrul *Q*.

La terminarea operației, rezultatul obținut va fi pe 16 biți, cu biții cei mai semnificativi în registrul *A* și restul biților în registrul *Q*. Din aceste două registre va fi extras rezultatul pe magistrala *OUTBUS*.

Continuăm cu împărțirea a două numere binare. Ca regulă generală, împărțirea numerelor se realizează prin scăderea repetată a împărțitorului, deplasat spre dreapta cu un rang, din restul de la scăderea precedentă, până la obținerea unui rest egal cu zero sau mai mic decât împărțitorul; ca prim rest se consideră deîmpărțitul [3]. Sunt cunoscute trei metode pentru realizarea împărțirii:

- *metoda comparării*: deîmpărțitul este comparat succesiv cu multipli ai împărțitorului, iar procesul de comparație se oprește la cel mai mare multiplu care este mai mic decât deîmpărțitul. Acesta se scade din deîmpărțit, iar ordinul multiplului devine valoarea câtului. O variantă a acestei metode presupune scăderea repetată a împărțitorului din deîmpărțit, procesul oprindu-se atunci când împărțitorul a devenit mai mare decât restul parțial. În aceste condiții numărul de scăderi devine valoarea câtului. [4]
- *metoda refacerii ultimului rest pozitiv*;
- *metoda fără refacerea ultimului rest pozitiv*.

Prezentăm aici un algoritm simplificat pentru împărțirea a două numere binare pe 8 biți fără semn, bazat pe metoda comparării. Împărțitorul este memorat în registrul *M* și deînmulțitul în registrul *Q*. Pentru operativitate, vom presupune că împărțitorul este nenul (evitarea împărțirii la 0) și mai mic decât deîmpărțitul. Algoritmul este exprimat în pseudocod și sub formă de schemă logică sau organigramă (**Fig. 5**).

```
                Declare registers A[8], M[8], Q[8], COUNT[3];
                Declare buses INBUS[8], OUTBUS[8];
START:          A ← 0, COUNT ← 0, M ← INBUS; [împărțitor]
                Q  ← INBUS; [deîmpărțit]
LSHIFT:         AQ ← LEFTSHIFT (AQ), COUNT ← COUNT + 1;
                if A < M then Q(0) ← 0, goto TEST;
                A ← A – M, Q(0) ← 1;
TEST:           if COUNT = 8 then go to FINISH;
                go to LSHIFT;
FINISH:         OUTBUS ← A; [rest]
                OUTBUS ← Q; [cât]
                STOP;
```

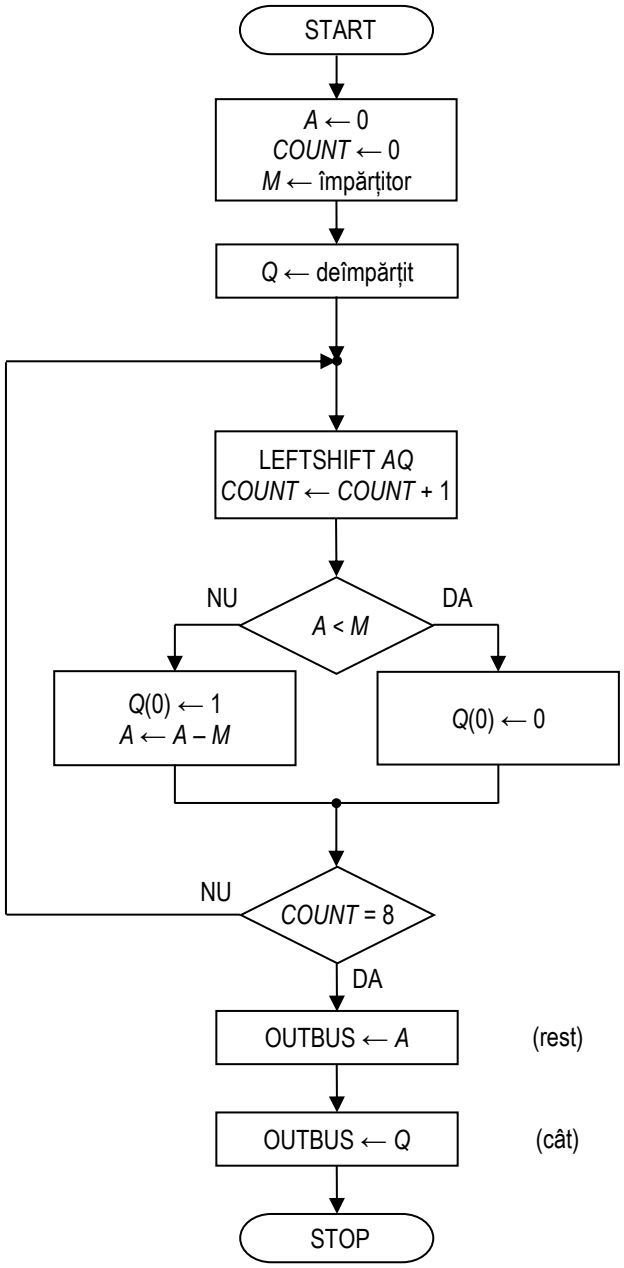


Fig. 5. Organigrama algoritmului simplificat al împărțirii cu rest.

La fiecare iterație se execută o deplasare la stânga a conținutului registrelor *A* și *Q* și se compară valoarea registrului *A* cu cea a registrului *M*. Dacă restul parțial (memorat în registrul *A*) este mai mic decât împărțitorul (memorat în *M*), atunci pe ultima poziție a registrului *Q*

(unde începe să se formeze câtul) se înscrie valoarea 0 și se trece mai departe. Dacă restul parțial este mai mare sau egal cu împărțitorul, atunci se efectuează scăderea $A \leftarrow A - M$ și pe ultima poziție a registrului Q se înscrie valoarea 1, după care se trece mai departe. Se vor realiza 8 iterații deoarece împărțitorul are 8 biți. După terminarea iterațiilor, deîmpărțitul a fost complet distrus, în registrul Q se găsește acum câtul și în registrul A se găsește restul, cele două rezultate fiind extrase din aceste două registre pe magistrala *OUTBUS*.

Evident, cei doi algoritmi prezentați aici reprezintă cazurile particulare în care ambii factori sunt fără semn, respectiv toți biții acestora contribuie la valorile numerelor prin ponderile pe care le au. În cazul prelucrării numerelor binare cu semn, când biții cei mai semnificativi preiau rolurile de *biți de semn*, se complică atât operațiile, cât și structurile sistemelor de calcul care le vor executa. Numerele negative vor fi exprimate în codul complement față de 2, ceea ce ușurează implementarea operațiilor aritmetice.

Lucrarea 1. Metode de descriere a comportării sistemelor utilizate în proiectare

Scopul lucrării

Comportarea unui sistem poate fi descrisă formal prin *organigrame, limbaje de descriere* și *diagrame de stări*. Scopul acestei lucrări este familiarizarea cu descrierea funcționării unui sistem cu ajutorul organigramelor de funcționare.

Considerații teoretice

Pentru a descrie structura unui sistem, se folosesc *diagramele bloc*. O componentă, sau un bloc într-o diagramă bloc, are forma generală ilustrată în **Fig. 1.1**.

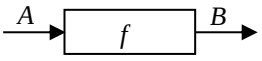


Fig. 1.1. Diagramă bloc a unui sistem de calcul.

Componenta apare de fapt ca o „cutie neagră", în sensul că nu este specificată structura internă a componentei, ci este descrisă în parte sau în întregime doar comportarea ei. Componentele care prezintă interes sunt mașinile cu număr finit de stări și cu comportare determinată.

Totalul intrărilor de stare A ale unei asemenea mașini, care îi determină comportarea, poate fi divizat în două părți:

1) semnale primare de intrare X, aplicate de o sursă externă;
2) stări interne Y, care cuprind informația stocată intern în mașină.

Similar, totalul ieșirilor de stare B ale mașinii este format din:

1) partea Z de ieșiri primare;
2) partea Y dată de noua stare internă a mașinii.

Deci funcția f, care descrie funcționarea componentei în discuție, are forma generală:

$$f: X \times Y \to Y \times Z$$

unde, dacă f mapează $(x, y) \in X \times Y$ în $(y', z) \in Y \times Z$, se notează

$$y', z \leftarrow f(x, y)$$

Această mapare este reprezentată explicit de un tabel de funcționare, numit *tabel de stări*, ale cărui linii și coloane reprezintă *Y* și *X* și ale cărui valori sunt date de totalul ieșirilor de stare $Y \times Z$.

Pe de altă parte, se obișnuiește să se reprezinte informația din tabelul de stări cu ajutorul unui graf, numit *diagramă de stări*. Nodurile diagramei de stări reprezintă stări interne, iar arcele reprezintă tranziții între stări. Spre deosebire de o diagramă bloc, o diagramă de stări nu oferă informații structurale, ci numai o descriere funcțională.

O altă modalitate de reprezentare a comportării stărilor este descrierea cu ajutorul unui set de ecuații (*state transitions*) de forma:

$$y_1, z_1 \leftarrow f(x_0, y_0)$$
$$y_2, z_2 \leftarrow f(x_0, y_1)$$
$$\dots\dots\dots$$
$$y_p, z_p \leftarrow f(x_r, y_s)$$

Fiecare tranziție poate fi văzută ca o declarație despre mașină, care poate fi interpretată ca:

- un *indicativ*, adică o declarare a ceea ce mașina face;
- un *imperativ*, adică o declarare a ceea ce mașina trebuie să facă.

În ultimul caz, tranziția de stări este văzută ca ordin sau instrucțiune. O secvență de tranziții de acest tip, care specifică o anumită funcționare dorită, este un program.

Asemenea descrieri (gen program) ale sistemelor digitale au devenit populare în ultimii ani; au apărut o varietate de limbaje speciale având diferite denumiri: limbaje de proiectare, limbaje de descriere hardware, sau limbaje de descriere a transferurilor între registre.

Dispozitiv de înmulțire (multiplicator) pentru numere binare cu virgulă fixă, în cod complement față de 2

În **Fig. 1.2** avem structura unui asemenea dispozitiv de înmulțire, respectiv *descrierea structurală* a acestuia. Acest multiplicator este proiectat pentru a realiza înmulțirea a două numere *X* și *Y*, ambele în reprezentare cu semn, respectiv în cod complement față de 2 pentru valorile negative. Fiecare număr este un cuvânt de 8 biți. Bitul cel mai semnificativ (**MSB** – ***M**ost **S**ignificant **B**it*) este bitul cel mai din stânga b_7. Numărul se presupune a fi întreg, respectiv cu virgula în dreapta bitului cel mai puțin semnificativ (*LSB* – ***L**east **S**ignificant **B**it*) b_0.

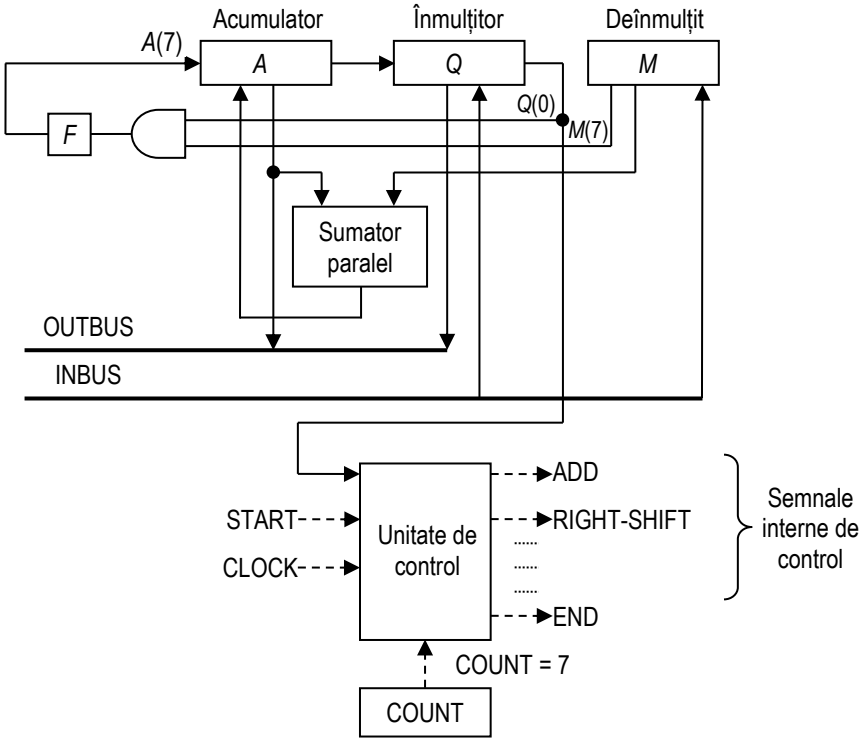


Fig. 1.2. Diagrama bloc pentru multiplicatorul a două numere cu virgulă fixă în reprezentare cod complement față de 2.

Registrul *M* („*Memory*”/„*Multiplicand*”), de 8 biți, memorează deînmulțitul plasat în el la începutul înmulțirii. Registrul *Q* („*Quotient*”), de 8 biți, memorează înmulțitorul plasat în el la începutul înmulțirii. Registrul *A* („*Accumulator*”), de 8 biți, este folosit ca registru de lucru împreună cu registrul *Q* pentru memorarea rezultatelor intermediare.

Jumătatea mai semnificativă a rezultatului este memorată în *A,* de unde poate fi transferată pe magistrala de ieșire. Registrele *A* și *Q* pot fi conectate fizic, pentru a forma un singur registru de deplasare *AQ* care își deplasează conținutul cu o poziție la dreapta la primirea semnalului de control *RIGHT-SHIFT* de la unitatea de control.

Considerăm că multiplicatorul are un controller extern care rulează programul de control al unității centrale de procesare (CPU – „***C**entral **P**rocessing **U**nit*”), care este sursa datelor de intrare și care trimite un semnal *START* pentru pornirea procesului de înmulțire.

Unitatea de control a multiplicatorului controlează iterațiile procesului. În fiecare iterație, unitatea de control examinează bitul *Q*(0) (cel mai din dreapta) al registrului *Q* unde a fost memorat înmulțitorul:

- ✓ Dacă $Q(0) = 1$, atunci conținuturile registrelor *A* și *M* sunt adunate și rezultatul este plasat în *A*;
- ✓ Dacă $Q(0) = 0$, pasul de adunare este sărit.

În ambele cazuri, registrul compus *AQ* este deplasat spre dreapta, aducând un nou bit în poziția $Q(0)$. Procesul se termină după efectuarea tuturor iterațiilor, după care înmulțitorul a fost distrus complet și rezultatul de lungime dublă se află în registrul compus *AQ*. Pe urmă produsul poate fi transferat pe magistrala de ieșire. Contorizarea iterațiilor se face cu un registru numărător *COUNT* de 3 biți.

După cum s-a mai arătat, *descrierea funcțională* se poate realiza în mai multe moduri: narativ (povestire), pseudocod, limbaje de descriere, diagrame de stări, organigrame etc. Descriem în continuare una din variantele algoritmului de înmulțire a două numere binare cu semn, cunoscută sub numele de ***algoritm Robertson***.

Fie operația $Y \times X$, unde $Y = y_7y_6y_5y_4y_3y_2y_1y_0$ este deînmulțitul și $X = x_7x_6x_5x_4x_3x_2x_1x_0$ este înmulțitorul, ambele cu semn (complement față de 2 unde este cazul). Avem 4 cazuri posibile, în funcție de semnele *X* și *Y*:

1. $x_7 = y_7 = 0$ (*X* și *Y* sunt pozitive)
Înmulțirea acestor numere este similară cu cea a numerelor fără semn. Cu alte cuvinte, produsul *P* este obținut printr-o serie de deplasări spre dreapta sub forma $P_i \leftarrow P_i + x_j \cdot Y$. Toate produsele parțiale sunt pozitive, deci la deplasarea spre dreapta (RIGHT-SHIFT) se adaugă în stânga 0.

2. $x_7 = 0,\ y_7 = 1$ (*X* este pozitiv, *Y* este negativ)
Produsul parțial este pozitiv, deci se adaugă 0 la stânga în timpul deplasărilor spre dreapta *până când este întâlnit primul* 1 *din X*. Înmulțirea lui Y cu acest 1 și adunarea sa duce la trecerea produsului parțial de la pozitiv la negativ, iar din acel moment se adaugă la stânga 1 în timpul deplasării spre dreapta.

3. $x_7 = 1,\ y_7 = 0$ (*X* este negativ, *Y* este pozitiv)
Produsul parțial este întotdeauna pozitiv, deci la deplasarea spre dreapta se adaugă 0 la stânga. După terminarea iterațiilor trebuie efectuată o scădere de corecție $P \leftarrow P - Y$.

4. $x_7 = y_7 = 1$ (*X* și *Y* sunt negative)
La fel ca în cazul 2, se adaugă 0 la stânga în timpul deplasărilor spre dreapta *până când este întâlnit primul* 1 *din X*, apoi 1. De asemenea, deoarece *X* este negativ, pasul cu scăderea de corecție trebuie efectuat.

Evident, se execută $n - 1 = 7$ iterații, deoarece bitul x_7 reprezintă *semnul înmulțitorului* și **NU** participă la realizarea produselor parțiale.

Modificările față de înmulțirea numerelor fără semn constă în tratarea specială a bitului de semn $A(7)$ al acumulatorului pentru tratarea produselor parțiale negative și în pasul de corecție finală atunci când înmulțitorul este negativ. Bitul de semn al produsului parțial $A(7)$ este dat de un bistabil F („***F**lag*") setat inițial la 0 și care devine 1 conform descrierii narative de mai sus. Astfel, F este setat conform formulei

$$F \leftarrow F \vee [Q(0) \wedge M(7)]$$

și va determina o nouă valoare pentru $A(7)$, care va fi adăugată la stânga produsului parțial după efectuarea deplasării la dreapta.

În continuare, algoritmul Robertson este exprimat atât în pseudocod, cât și sub formă de schemă logică sau organigramă (**Fig. 1.3**).

```
            Declare registers A[8], M[8], Q[8], F[1], COUNT[3];
            Declare buses INBUS[8], OUTBUS[8];
START:      A ← 0, F ← 0, COUNT ← 0, M ← INBUS; [deînmulțit]
            Q ← INBUS; [înmulțitor]
TEST1:      if Q(0) = 0 then go to RSHIFT;
            A ← A + M, F ← F ∨ [Q(0) ∧ M(7)];
RSHIFT:     A(6:0),Q ← A,Q(7:1), A(7) ← F, COUNT ← COUNT+1;
            if COUNT = 7 then go to TEST2;
            go to TEST1;
TEST2:      if Q(0) = 0 then go to FINISH;
            A ← A – M;
FINISH:     OUTBUS ← A; [biții cei mai semnificativi]
            OUTBUS ← Q(7:1); [biții cei mai puțin semnificativi]
            STOP;
```

La descrierea prin *organigramă*, un set de operații simultane se prezintă printr-o listă de declarații într-un dreptunghi. Operațiile consecutive sunt indicate prin săgeți între dreptunghiuri. Salturile condiționate sunt reprezentate printr-un romb, în interiorul căruia se scrie condiția care se testează. Avantajul reprezentării comportării mașinii prin organigrame este că ele explicitează aspecte funcționale mai puțin vizibile într-o modelare prin limbaj de descriere (ex. ciclurile).

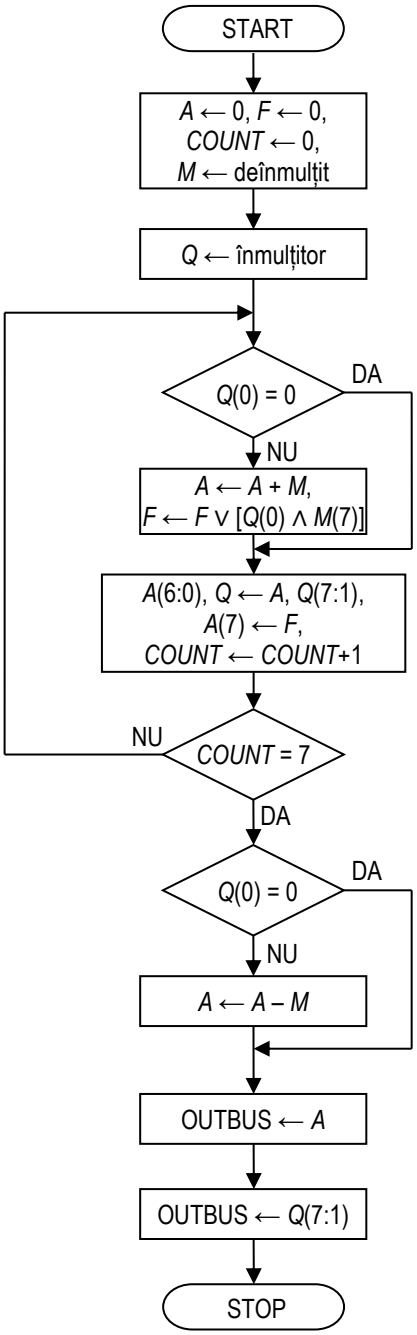


Fig. 1.3. Organigrama pentru înmulțirea numerelor în cod complement față de 2.

Mersul lucrării

a) Să se exemplifice operațiile aritmetice de:
   - înmulțire
   - împărțire

   a două numere cu semn reprezentate în complement față de 2.
b) Să se implementeze procesul de înmulțire a două numere în reprezentare complement față de 2.
c) Să se descrie algoritmul de împărțire cu refacerea restului, atât prin organigramă, cât și în pseudocod.
d) Să se descrie algoritmul de împărțire fără refacerea restului, atât prin organigramă, cât și în pseudocod.

Lucrarea 2. Tehnici de proiectare a unității de procesare a datelor într-un sistem de calcul

Scopul lucrării

Lucrarea își propune prezentarea unei proceduri euristice pentru proiectarea unui sistem, insistând asupra proiectării unității de procesare a datelor, proiectarea unității de control fiind obiectul altor lucrări.

Considerații teoretice

Un sistem digital poate fi divizat în două părți:

- ✓ o *unitate de procesare a datelor*;
- ✓ o *unitate de control*.

Pentru cazul unui sistem de calcul a cărui divizare este prezentată în **Fig. 2.1**, unitatea principală de procesare CPU (***C**entral **P**rocessing **U**nit*) reprezintă partea de procesare a datelor, în timp ce memoria, care cuprinde programul care se execută, reprezintă partea de control (pentru simplitate, au fost neglijate echipamentele de intrare/ieșire).

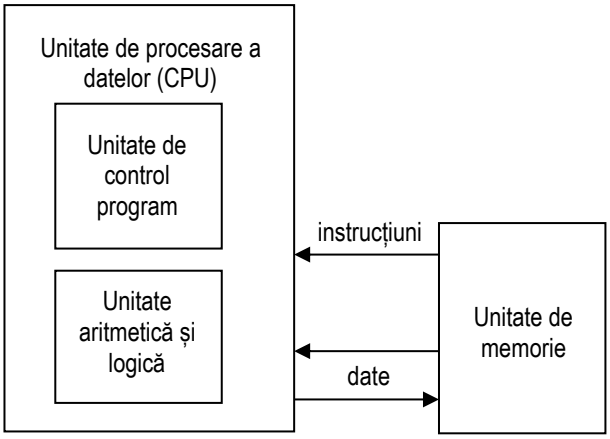


Fig. 2.1. Sistem de calcul.

La rândul ei, unitatea principală de procesare (CPU) este compusă din *unitatea aritmetică și logică* (UAL), care se ocupă cu procesarea datelor, și o *unitate de control a programului*. Este de remarcat că distincția între unitățile de control și unitățile de date se bazează pe faptul că sistemul recunoaște implicit o submulțime a informației procesate ca reprezentând date.

Componentele și căile de date traversate de către date constituie *unitatea de procesare* a datelor. Restul mașinii constituie *unitatea de control*. Căile traversate de date, precum și operațiile realizate de unitatea de procesare a datelor în fiecare moment de timp, sunt specificate de către unitatea de control.

Distincția între partea de control și de date poate fi modelată folosind definiția de funcționare a mașinii $f: A \to B$ și descompunând A și B în partea de control și de date, indicate prin indicii c și d. Astfel, fie $A = A_C \times A_d$ și $B = B_C \times B_d$. Se poate considera f ca fiind compusă dintr-o funcție de control $f_c: A_c \to B_c$ și o funcție de date $f_d: (A_d \times A_c) \to B_d$. În acest sens, diagrama bloc din **Fig. 2.2** reprezintă structura sistemului considerat, dacă se face implementarea separată a părții de control și a celei de date.

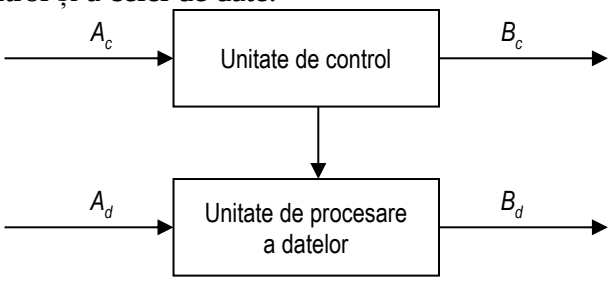


Fig. 2.2. Sistem de calcul, format din unitate de control și unitate de procesare a datelor.

În practică, distincția între partea de procesare a datelor și partea de control este mai puțin clară. De cele mai multe ori, partea de procesare a datelor este influențată de partea de control, dar sunt cazuri când partea de procesare a datelor influențează partea de control. De exemplu, derularea programului în unitatea de control poate fi influențată de posibilitatea apariției unei depășiri în cazul unei operații aritmetice efectuate în cadrul CPU-ului. Astfel, așa cum este arătat în **Fig. 2.3**, funcția de control f_C devine dependentă atât de A_C cât și de A_d. De aici apare o anumită ambiguitate în determinarea semnalelor de date și de control; aceleași semnale pot fi văzute în procesul de proiectare uneori ca semnale de control, alteori ca semnale de date.

Presupunem că un sistem este specificat printr-o diagramă bloc. Dacă fiecare bloc este împărțit în parte de control și parte de procesare a datelor, rezultă o diagramă care poate fi privită ca fiind compusă din două subdiagrame: una constând numai din unități de control, cealaltă numai

din unități de procesare a datelor. Astfel, sistemul apare ca fiind format din două subsisteme interconectate.

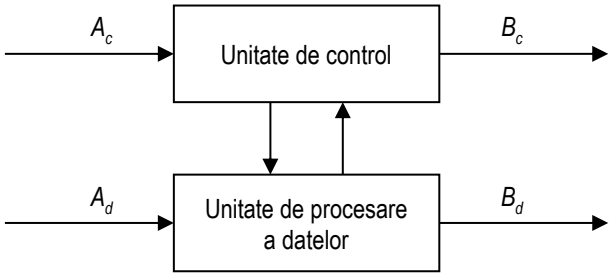


Fig. 2.3. Unitatea de control și unitatea de procesare a datelor.

Această împărțire este importantă, întrucât proiectarea unității de control și a celei de procesare a datelor se face separat și în mod diferit. Mai mult, se poate face o clasificare a sistemelor în funcție de modul în care sunt implementate funcțiile sale majore de control:

- Dacă unitatea de control se implementează cu circuite logice fixe, astfel încât f_C este permanentă, se spune că sistemul este *cablat* (*hardwired*).
- Dacă f_C este implementat prin memorarea informațiilor de control într-o memorie de comandă (funcțiile de control se bazează mai mult pe software decât pe hardware), se spune că sistemul este *microprogramat*.

Nu există notații standard pentru distingerea unităților și a semnalelor de date de cele de control. Totuși, așa cum se arată în **Fig. 2.4**, pot fi folosite *linii întrerupte* pentru figurarea semnalelor de control și *linii continue* pentru figurarea semnalelor de date. Menționăm că liniile de control sunt indicate, în general, prin etichetarea liniilor, mai degrabă decât a blocurilor pe care le controlează.

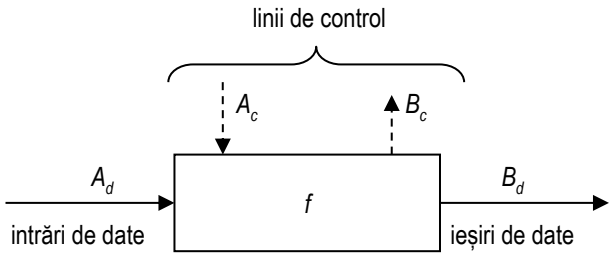


Fig. 2.4. Reprezentarea liniilor de date și a liniilor de control.

Diagramele bloc sunt cele mai simple ilustrări ale unei structuri de mașini, de unde rezultă registrele și componentele funcționale ale părții de procesare a datelor pentru mașină.

Considerăm exemplul din **Fig. 2.5**, unde R_1 și R_2 sunt registre, iar unitatea funcțională poate fi un dispozitiv de adunare/scădere sau orice alt dispozitiv. Liniile din diagramă sunt *căile de date* ale mașinii, care de cele mai multe ori sunt legături logice și nu fizice. Când mai multe căi de date intră sau ies dintr-o unitate, este necesară introducerea unor *puncte* și *semnale de control* pentru selectarea și validarea acestor căi de date, ceea ce este indicat în **Fig. 2.6**.

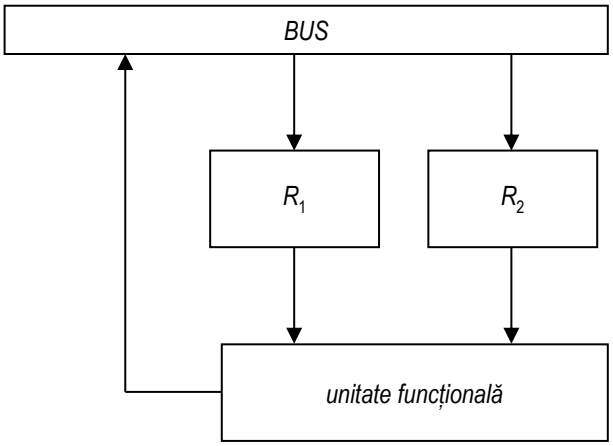


Fig. 2.5. Sistem de calcul – diagramă bloc.

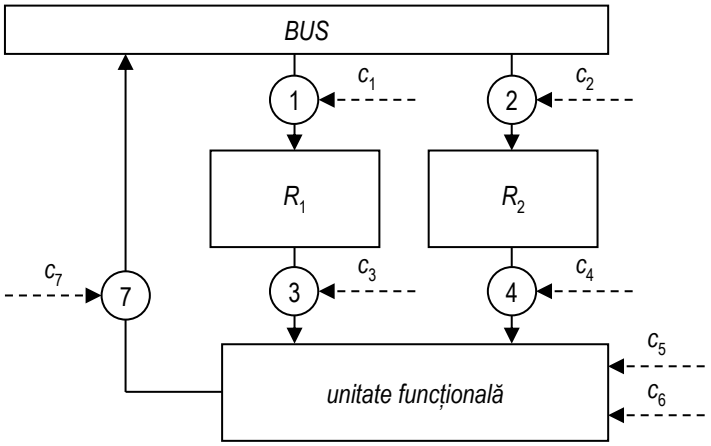


Fig. 2.6. Puncte de control în diagrama bloc.

Un ***punct de control*** este un *switch* (comutator) pe o linie de date care, atunci când este activat de un semnal de control, validează transferul datelor pe calea de date, respectiv blochează calea de date atunci când nu este activat. Deci, punctul de control este un dispozitiv de ajutor conceptual în proiectarea și descrierea comportării mașinii.

Punctele de control se implementează cu circuite logice a căror natură depinde de caracteristicile dispozitivelor conectate la căile de date. Toate *semnalele de control* (c_1, c_2,…, c_7), care activează căile de date în punctele de control, sunt generate de unitatea de control a mașinii.

Tehnici de proiectare pentru proiectarea unui sistem la nivel de registru

Problema proiectării unui astfel de circuit poate fi formulată astfel: dându-se un set de algoritmi sau instrucțiuni, să se proiecteze circuitul folosind un număr specificat de registre, care să realizeze funcțiile dorite și să satisfacă anumite criterii de cost și de performanță.

Nu există metode care să impună o anumită structură matematică a circuitului, așa cum este cazul proiectării circuitelor combinaționale, unde este vorba de algebra booleană. Deci, proiectarea acestor circuite tinde să fie *ad-hoc* și depinde în mare măsură de experiența proiectantului. Cu toate acestea, este posibilă identificarea unei proceduri de abordare generală, de forma următoare:

Pas 1. Se definește funcționarea dorită printr-un set de operații de transfer *S* între registre, astfel încât fiecare operație să poată fi implementată direct folosind componentele de proiectare disponibile.

Pas 2. Se analizează *S* pentru a determina tipul componentelor și numărul fiecărui tip cerut pentru procesarea datelor.

Pas 3. Se construiește o diagramă bloc *D* pentru unitatea de procesare a datelor, folosind componentele identificate în pasul 2. Se interconectează componentele, astfel încât să fie prevăzute toate căile implicate de *S* și să fie satisfăcute condițiile impuse de cost și de performanță;

Pas 4. Se analizează *S* și *D*, pentru a se stabili punctele de control necesare și se introduce logica necesară pentru implementarea acestora.

Pas 5. Se proiectează unitatea de control a sistemului astfel încât să se genereze semnalele de control necesare în ordinea specificată de *S*.

Pas 6. Se elimină redundanțele și se încearcă, pe cât posibil, simplificarea circuitului.

Pentru implementarea setului dat de algoritm, primul pas al procedurii implică un proces de translație, analog cu scrierea unui program în limbaj de asamblare; *S* reflectă îndemânarea proiectantului.

Identificarea componentelor pentru procesarea datelor, din pasul 2, este directă. O declarație de forma:

X : A ← A + B

implică existența a două registre pentru memorarea lui A și B, respectiv a unui sumator la care A și B trebuie să fie conectate prin căile de date. Atunci când există posibilitatea partajării componentelor, pot apărea complicații. De exemplu, declarația:

X : A ← A + B, C ← C + D

definește două operații de adunare. Dacă sunt prevăzute două sumatoare independente, ele pot fi executate în paralel, întrucât cele două adunări nu implică aceiași operanzi. Totuși, costul circuitului poate fi redus prin utilizarea unui singur sumator și realizarea secvențială a celor două adunări, conform declarațiilor:

$X(t_0)$: A ← A + B;

$X(t_1)$: C ← C + D;

Identificarea paralelismului inerent unui algoritm poate fi extrem de dificilă. Construcția diagramei, în pasul 3, cere definirea unei structuri de magistrală potrivite.

Performanța mașinii în executarea unui anumit algoritm *A* din *S* este proporțională cu totalul întârzierilor de pe toate căile de date și a celor datorate traversării componentelor pentru execuția lui *A*.

Costul circuitului este

$$\sum_{i=1}^{n} n_i \cdot c_i$$

unde c_i este costul componentei de tip *i* și n_i este numărul componentelor de tip *i* utilizate.

Dacă o anumită proiectare nu satisface un anumit criteriu de performanță (de exemplu: un algoritm este realizat prea încet, sau maximul de cost este depășit), este necesară reîntoarcerea la pasul 1 și reproiectarea lui *S*. Deci, există o interacțiune între pași de proiectare diferiți. De multe ori pentru a se realiza criteriile impuse apare necesitatea de a modifica repetat anumite decizii luate în pași anteriori.

Proiectarea unității de control (pasul 5) necesită analiză atentă de timp a algoritmilor din *S*.

Ultimul pas al procedurii evidențiază natura ei euristică. În practică, îmbunătățirile sunt făcute în fiecare pas al proiectării.

Pentru exemplificarea procedurii se discută proiectarea dispozitivului de înmulțire a două numere cu virgulă fixă reprezentate în

cod complement față de 2. În Lucrarea 1 s-a stabilit funcționarea circuitului în termenii secvențelor de transfer între registre.

Deci, sunt necesare trei registre de 8 caractere binare: *A*, *M* și *Q*. Întrucât conținutul lui *A* și *Q* trebuie deplasat la dreapta, *A* și *Q* vor fi registre de deplasare. Considerații de viteză implică cerința ca toate cele 3 registre să aibă capacitate de încărcare/descărcare paralelă.

Declarația $A \leftarrow A + M$ cere utilizarea unui sumator paralel de 8 caractere binare. Scăderea, specificată de $A \leftarrow A - M$, va fi implementată prin formarea complementului față de 2 a lui *M*, după care se execută transferul de registre: $A \leftarrow A + (-M)$.

Căile de date implicate de algoritm sunt ușor de identificat din organigrama de descriere a funcționării. De exemplu, declarația

$$A \leftarrow A + M$$

cere ca atât *A*, cât și *M*, să fie conectate la intrările sumatorului prin căi de date de 8 biți. O altă cale de date leagă ieșirea sumatorului la registrul *A*. În **Fig. 2.7** este prezentată structura mașinii rezultate.

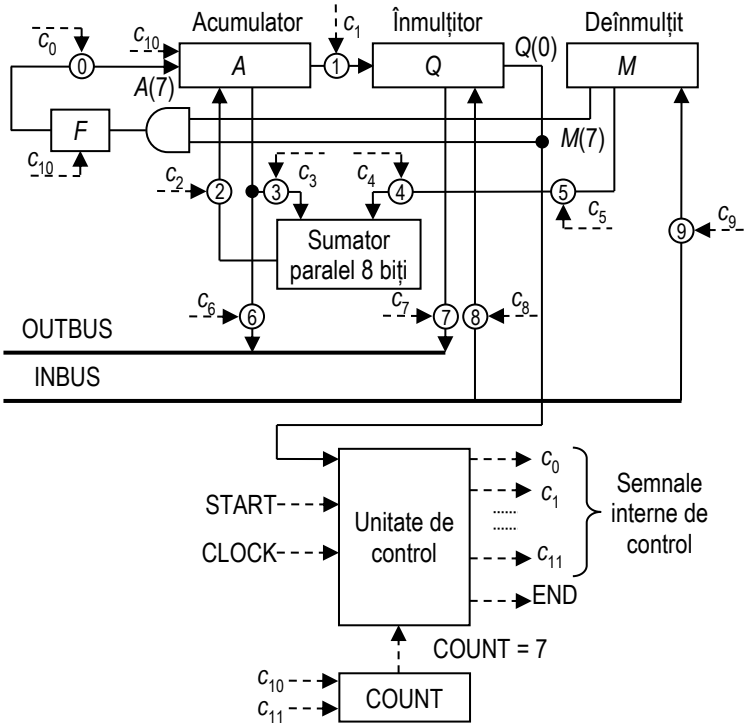


Fig. 2.7. Diagrama bloc a dispozitivului de înmulțire în virgulă fixă pentru două numere în complement față de 2.

Pe diferitele căi de date din figură au fost introduse puncte de control simbolice (c_i). Setul de semnale $\{c_i\}$, care activează aceste puncte de control, este descris în **Fig. 2.8**.

Pentru a completa proiectarea părții de procesare a datelor pentru dispozitivul de înmulțire analizat, este necesară implementarea punctelor de control. În **Fig. 2.9** se arată cum se implementează punctele de control de pe calea de date de la registrele *A* și *M* la sumator.

Semnalul de control	Operație controlată
c_0	$A(7) \leftarrow F$
c_1	$A(6:0), Q \leftarrow A, Q(7:1)$ (deplasare dreapta *A* și *Q*)
c_2	Transferă suma în *A*
c_3	Transferă *A* la prima intrare a sumatorului
c_4	Transferă +/– *M* la a doua intrare a sumatorului
c_5	Transformă valoarea lui *M* în –*M*
c_6	OUTBUS $\leftarrow A$
c_7	OUTBUS $\leftarrow Q(7:1)$
c_8	$Q \leftarrow$ INBUS
c_9	$M \leftarrow$ INBUS
c_{10}	$A \leftarrow 0$, *COUNT* $\leftarrow 0$, $F \leftarrow 0$ (resetare)
c_{11}	incrementează *COUNT*

Fig. 2.8. Semnale de control pentru multiplicatorul în complement față de 2.

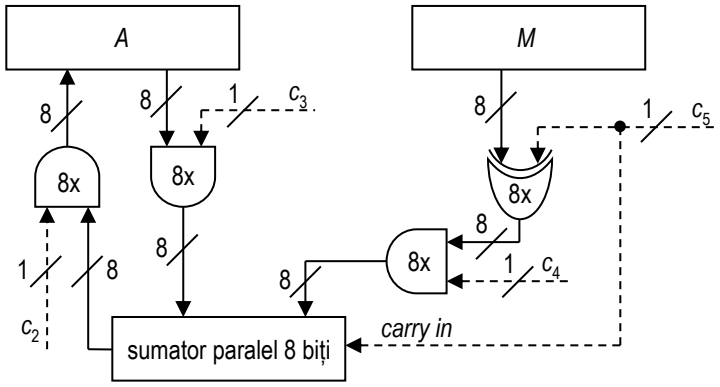


Fig. 2.9. Implementarea unor puncte de control din **Fig. 2.7**.

Porțile SAU-EXCLUSIV din **Fig. 2.9** sunt folosite pentru a schimba *M* în $\overline{M}$ atunci când c_5 se modifică de la 0L la 1L. Pentru a aduna o unitate la suma formată, c_5 este aplicat și terminalului *CARRY IN* al sumatorului. Astfel:

- ✓ când c_3, c_4 și c_5 sunt toate 1L, multiplicatorul face adunarea $A + \bar{M} + 1$, ceea ce este același lucru cu $A - M$;
- ✓ când c_3, c_4 sunt 1L și c_5 este 0L, multiplicatorul realizează adunarea $A + M$.

Proiectarea unității de control a multiplicatorului este obiectul altor lucrări de laborator.

Menționăm faptul că proiectarea anterioară nu este decât una din posibilitățile de implementare pentru multiplicatorul a două numere în complement față de 2.

Mersul lucrării

1. Pe baza celor prezentate în lucrare, să se facă proiectarea completă a unității de procesare a datelor pentru dispozitivul de înmulțire în virgulă fixă a două numere în complement față de 2.
2. Să se implementeze unitatea de procesare a datelor a unui dispozitiv de împărțire, care folosește pentru împărțire algoritmul de împărțire cu refacerea restului.
3. Să se implementeze unitatea de procesare a datelor pentru o UAL (***U**nitate **A**ritmetică și **L**ogică*) care să realizeze operațiile de:
   - adunare;
   - scădere;
   - înmulțire;
   - împărțire.

   a două numere în virgulă fixă în reprezentare cod complement față de 2.

Lucrarea 3. Metoda tabelului de stare pentru proiectarea unității de control cablate a sistemelor digitale

Scopul lucrării

Lucrările 3, 4 și 5 își propun însușirea de către studenți a unei metodologii de proiectare a unității de control cablate a unui sistem de calcul digital.

Considerații teoretice

Pentru proiectarea unităților de control cablate există 3 metodologii posibile de abordare:

Metoda I – metodă bazată pe utilizarea tabelului de stare;
Metoda II – metodă bazată pe utilizarea elementelor de întârziere pentru temporizarea semnalelor de control;
Metoda III – metodă bazată pe utilizarea numărătoarelor pentru secvențierea în timp a semnalelor de control.

Metoda I, cea mai formală dintre cele trei, încorporează tehnici sistematice pentru minimizarea numărului de porți și bistabile.

Metodele II și III, mai puțin formale, își propun derivarea circuitului logic direct din descrierea originală a funcționării circuitului, respectiv se bazează pe organigrame de funcționare.

Proiectul se obține cu efort mai mic cu metodele II și III, dar el nu va conține numărul minim posibil de porți și bistabile. Aceste proiecte sunt în general mai ușor de înțeles, cu mai puține posibilități de eroare și mai ușor de întreținut. În practică, unitățile de control sunt atât de complexe încât niciuna din metode, considerată singură, nu duce la rezultate satisfăcătoare atât din punct de vedere al circuitului, cât și al costului. Ca un rezultat, de multe ori se utilizează metode *ad-hoc* de proiectare, care nu pot fi formalizate.

Metoda tabelelor de stare

Pentru descrierea funcționării unității de control, această metodă utilizează *tabelele de stare* de tipul celui din **Fig. 3.1**.

Stări	I_1	I_2	• • •	I_j	• • •	I_m
S_1	$S_{1,1}$, $z_{1,1}$	$S_{1,2}$, $z_{1,2}$	• • •	• • •	• • •	$S_{1,m}$, $z_{1,m}$
S_2	$S_{2,1}$, $z_{2,1}$	$S_{2,2}$, $z_{2,2}$	• • •	• • •	• • •	$S_{2,m}$, $z_{2,m}$
• • •	• • •	• • •	• • •	• • •	• • •	• • •
S_i	• • •	• • •	• • •	$S_{i,j}$, $z_{i,j}$	• • •	• • •
• • •	• • •	• • •	• • •	• • •	• • •	• • •
S_n	$S_{n,1}$, $z_{n,1}$	$S_{n,2}$, $z_{n,2}$	• • •	• • •	• • •	$S_{n,m}$, $z_{n,m}$

Fig. 3.1. Tabel de stare – formă generală.

Fie C_{in} și C_{out} mulțimile variabilelor de intrare și de ieșire ale unității de control. Rândurile tabelului de stări corespund setului de stări interne ale mașinii $\{S_i\}$. O *stare internă* este determinată de informația memorată în unitate la momente discrete de timp, determinate de perioada impulsului de tact. Coloanele corespund unui set de semnale externe ale unității de control, deci lui C_{in}. La incidența rândului S_i și coloanei I_j avem elemente de forma $S_{i,j}$, $z_{i,j}$ unde:

- $S_{i,j}$ indică starea următoare a unității de control;
- $z_{i,j}$ indică setul de semnale de ieșire din C_{out} care este activat prin aplicarea lui I_j la intrarea unității de control aflată în starea S_i.

Descrierea prin tabele de stare poate fi utilizată ca punct de plecare la implementarea unităților de control mici. În continuare schițăm pașii acestei metode, pe care o vom exemplifica mai târziu:

- ✓ pas 1 – construirea tabelului de stări pe baza specificațiilor de funcționare;
- ✓ pas 2 – reducerea tabelului de stări la formă minimă, utilizând tehnicile de minimizare cunoscute;
- ✓ pas 3 – codificarea stărilor;
- ✓ pas 4 – construirea unui tabel de tranziții utilizând tipurile de bistabile date, tabel care va defini funcționarea circuitului; acesta este un tabel de adevăr pentru ieșirile primare $\{z_i\}$ și intrările de excitație ale bistabilelor $\{Y_i\}$;
- ✓ pas 5 – proiectarea circuitului cu o tehnică adecvată (Quine-McCluskey, Karnaugh).

Cu toate că această metodă poate fi utilizată pentru proiectarea unității de control din cadrul oricărui proiect, dezavantajele sale sunt:

1) Numărul stărilor și al combinațiilor de intrare poate fi atât de mare, încât dimensiunea tabelului și a calculului devine excesivă.
2) Tabelul de stări tinde să ascundă informații utile, legate de comportarea circuitului (de exemplu, existența ciclurilor). Circuitele de control proiectate în această manieră tind să aibă o structură stufoasă și complexă, ceea ce face foarte dificilă verificarea proiectului și elaborarea unor acțiuni de mentenanță pentru circuit.

Unitatea de control a unui dispozitiv de înmulțire în virgulă fixă pentru două numere reprezentate în complement față de 2

În lucrarea anterioară s-a realizat o diagramă bloc a sistemului proiectat, arătându-se în detaliu unitatea de procesare a datelor și punctele de control. În **Fig. 3.2** sunt prezentate conexiunile de intrare și de ieșire (I/E) ale unității de control.

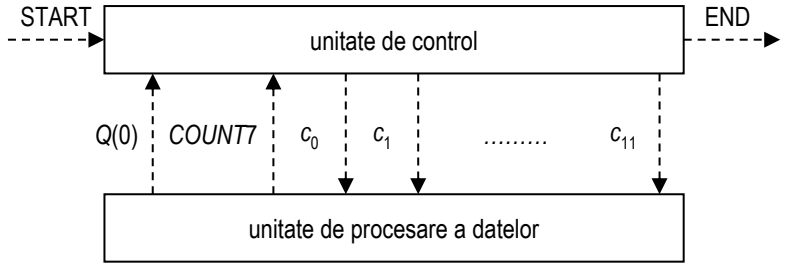


Fig. 3.2. Liniile de intrare/ieșire ale unității de control a multiplicatorului în complement față de 2.

Observație: Numărătorul pentru iterare *COUNT* este alocat unității de procesare a datelor.

Funcțiile semnalelor de control c_0, c_1, …, c_{11}, transmise unității de procesare a datelor, au fost studiate în lucrarea anterioară. Q(0) este bitul cel mai din dreapta (LSB) al registrului *Q*. *COUNT*7 este un semnal derivat din numărătorul pentru iterare, care primește valoarea 1L când *COUNT* = 7 și 0L în celelalte cazuri.

Fig. 3.3 redă algoritmul cu semnalele de control care sunt activate în timpul execuției fiecărei microoperații. Pe baza acestuia se va urmări modul în care sunt generate semnalele de control prin metoda prezentată.

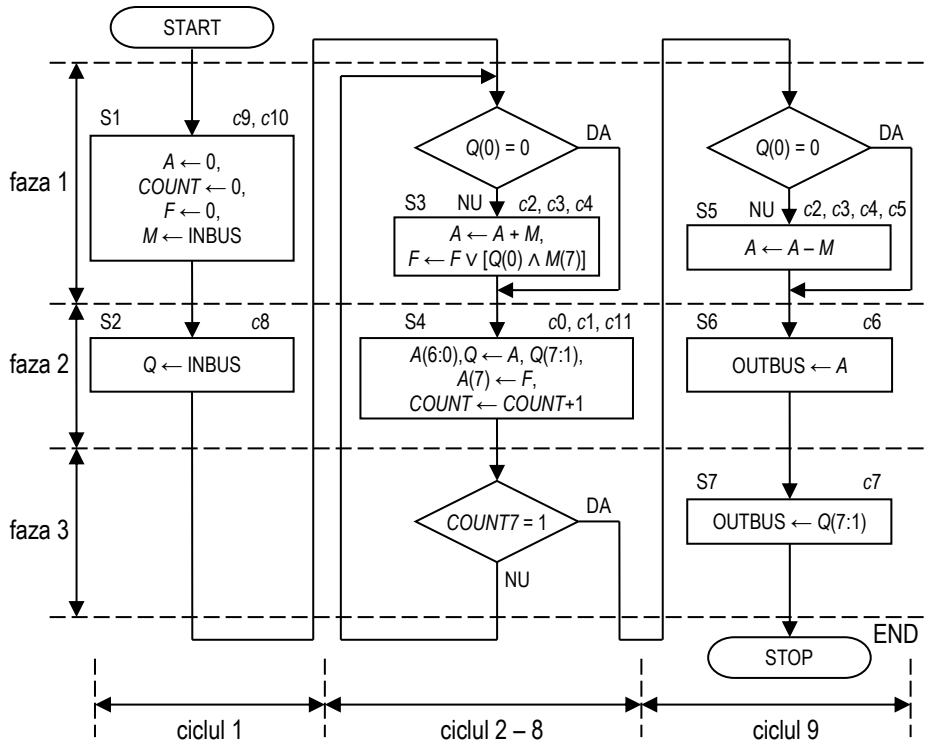


Fig. 3.3. Organigrama pentru înmulțirea în complement față de 2.

Este necesară construirea unui *tabel de stări* pentru unitatea de control. Se asociază câte o stare S_i fiecărui bloc de microoperații din **Fig. 3.3**, obținându-se 7 stări notate S_1 la S_7. Asociem starea S_0 pentru starea adițională, de așteptare, a unității de control.

Unitatea de control are 3 intrări externe: *START*, *Q*(0) și *COUNT7*, deci sunt 8 combinații posibile ale informațiilor de intrare.

Fig. 3.4 arată tabelul de stări a unității de control, unde simbolul „ϕ" înseamnă că nu se activează nicio linie de control în acea stare. Rubricile goale corespund intrărilor din tabel nespecificate, adică situația când în stările respective nu apar combinațiile specificate de intrări externe. De exemplu:

- *START* nu este 1L decât în S_0;
- *COUNT7* (care devine 1L când *COUNT = 7*) nu poate fi 1L în starea S_2, întrucât în starea precedentă S_1 numărătorul *COUNT* este resetat și astfel *COUNT7* este setat pe 0L etc.

Stare	Combinații de intrare (*START*, *Q*(0), *COUNT7*)							
	000	001	010	011	100	101	110	111
S_0	S_0,ϕ	S_0,ϕ	S_0,ϕ	S_0,ϕ	S_1,ϕ	S_1,ϕ	S_1,ϕ	S_1,ϕ
S_1	S_2,c_9,c_{10}		S_2,c_9,c_{10}					
S_2	S_4,c_8		S_3,c_8					
S_3			S_4,c_2,c_3,c_4					
S_4	S_4,c_0,c_1,c_{11}	S_6,c_0,c_1,c_{11}	S_3,c_0,c_1,c_{11}	S_5,c_0,c_1,c_{11}				
S_5				S_6,c_2,c_3,c_4,c_5				
S_6		S_7,c_6		S_7,c_6				
S_7		S_0,c_7,END		S_0,c_7,END				

Fig. 3.4. Tabelul de stări a unității de control.

Stări	Variabila de stare		
	x_1	x_2	x_3
S_0	0	0	0
S_1	0	0	1
S_2	0	1	0
S_3	0	1	1
S_4	1	0	0
S_5	1	0	1
S_6	1	1	0
S_7	1	1	1

Fig. 3.5. Stările sistemului și codificarea acestora.

Pentru simplificarea proiectării logice a unității se pot folosi intrări nespecificate. Deoarece avem 8 stări ale sistemului, sunt necesare $[log_2 8] = 3$ bistabile pentru unitatea de control. Urmează alegerea tipului de bistabile și asigurarea celor 8 combinații posibile ale celor 3 variabile de stare pentru cele 8 stări care au fost identificate. Se preferă bistabilele JK deoarece toleranța lor pentru valorile nespecificate la intrările J și K simplifică sinteza circuitului.

Întrucât unitatea de control este sincronă, alocarea stărilor se poate face arbitrar. De notat faptul că, în cazul circuitelor asincrone, o asemenea alocare arbitrară poate duce la apariția hazardului sau a curselor critice, fiind necesară realizarea unei codificări adiacente (schimbarea unui singur bit) pentru stările între care se fac tranziții. Pe de altă parte, în literatură sunt indicate metode de alocare și pentru circuitele sincrone, al căror scop este minimizarea numărului de porți implicate pentru sinteza acestora. Pentru simplificarea prezentării, s-a

preferat alocarea arbitrară din **Fig. 3.5**, optându-se pentru utilizarea bistabilelor JK pentru memorarea variabilelor de stare. În aceste condiții, unitatea de control are forma generală din **Fig. 3.6**.

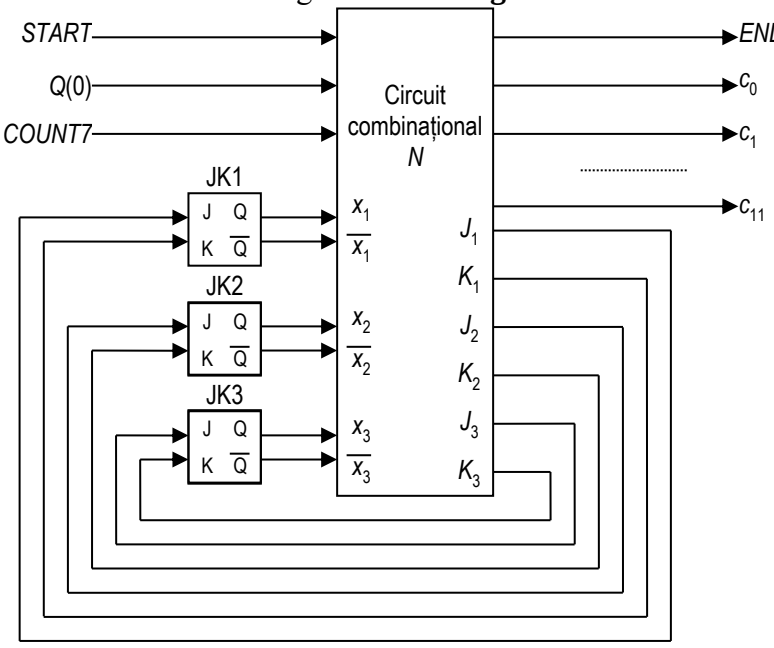


Fig. 3.6. Unitatea de control a multiplicatorului în complement față de 2, obținută pe baza metodei tabelului de stare.

Rămâne de proiectat circuitul combinațional *N*, cu cele 6 intrări și 19 ieșiri ale sale. Metoda standard este de a construi *un tabel de tranziții* având formatul din **Fig. 3.7**, care poate fi privită ca un tabel de adevăr unde ieșirile lui *N* sunt reprezentate ca funcții de variabilele de stare și celelalte intrări ale lui *N*. Intrările în partea de ieșire a tabelului de tranziții sunt determinate din tabelul de stări, din alocarea stărilor și din ecuațiile de definire ale bistabilelor.

Considerăm, de exemplu, rândul marcat cu „ * ” în **Fig. 3.7**. Această intrare corespunde coloanei 001 și rândului S_4 din tabelul din **Fig. 3.4**, întrucât lui S_4 i s-a alocat valoarea $x_1x_2x_3 = 100$. Se vede din tabelul de stări că starea următoare este $S_6 = 011$ și că vor fi activate variabilele de ieșire c_0, c_1 și c_{11}. Valorile de ieșire se introduc direct în tabelul de tranziții. Tranziția din starea S_4 în starea S_6 implică trecerea bistabilului *JK*1 din 1L ($x_1 = 1$L) în 0L ($x_1 = 0$L) și trecerea bistabilelor *JK*2 și *JK*3 din 0L ($x_2 = x_3 = 0$L) în 1L ($x_2 = x_3 = 1$L). Cunoscând funcționarea bistabilelor JK, se pot determina direct valorile cerute de

intrările bistabilelor $\{J_i, K_i\}$ pentru realizarea acestor schimbări de stare: J_1 = *nd* și K_1 = 1 resetează *JK*1, în timp ce $J_2 = J_3 = 1$ și $K_2 = K_3$ = *nd* activează atât *JK*2, cât și *JK*3. Aceste valori sunt pe urmă introduse în tabelul de tranziții. Odată completat tabelul de tranziții, el constituie un tabel de adevăr pentru circuitul combinațional *N*, care poate fi realizat utilizând circuite combinaționale sau PLA-uri (**P**rogrammable **L**ogic **A**rray).

Intrări						Ieșiri										
START	Q(0)	COUNT7	x_1	x_2	x_3	END	c_0	c_1	...	c_{11}	J_1	K_1	J_2	K_2	J_3	K_3
0	0	0	0	0	0	0	0	0	...	0	0	nd	0	nd	0	nd
0	0	0	0	0	1	0	0	0	...	0	0	nd	1	nd	nd	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
*0	0	1	1	0	0	0	1	1	...	1	nd	1	1	nd	1	nd
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Fig. 3.7. Tabelul de tranziții.

Sinteza circuitului combinațional *N* poate fi simplificată prin divizarea în două etape, obținându-se astfel două dispozitive electronice care se interconectează. Primul dispozitiv gestionează cele trei variabile de stare, iar al doilea generează semnalele de control conform acestora.

În prima etapă se construiește un tabel de tranziții (de adevăr) numai pentru semnalele de comandă *J* și *K* ale celor trei bistabile care gestionează variabilele de stare. Din acest tabel se deduc ecuațiile celor șase semnale de comandă și se construiește schema electronică a circuitului combinațional CLC_1 pentru acestea, având ca semnale de intrare *START*, *Q*(0), *COUNT*7, x_1, x_2 și x_3, respectiv ca semnale de ieșire J_1, K_1, J_2, K_2, J_3 și K_3. Aceste semnale de ieșire sunt livrate la intrările bistabilelor care oferă la ieșiri semnalele corespunzătoare variabilelor de stare x_1, x_2 și x_3. Astfel, schema electronică a acestui dispozitiv secvențial va conține și cele trei bistabile JK, deci intrările sale sunt *START*, *Q*(0) și *COUNT*7, iar ieșirile sale sunt x_1, x_2 și x_3 (**Fig. 3.8**).

În etapa a doua se construiește dispozitivul care va genera semnalele de control conform stărilor sistemului. Conform schemei logice din **Fig. 3.3** și alocării stărilor din **Fig. 3.5** se scriu ecuațiile

semnalelor de control în funcție de variabilele de stare. Pe baza ecuațiilor se construiește circuitul combinațional CLC_2 cu intrările x_1, x_2 și x_3 și ieșirile c_0, c_1, …, c_{11} și *END* (**Fig. 3.9**).

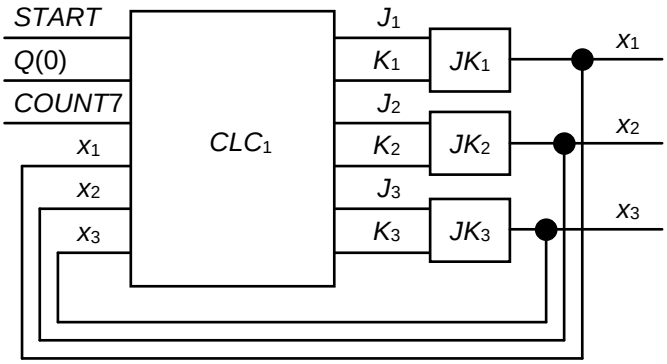


Fig. 3.8. Dispozitivul electronic secvențial care gestionează variabilele de stare.

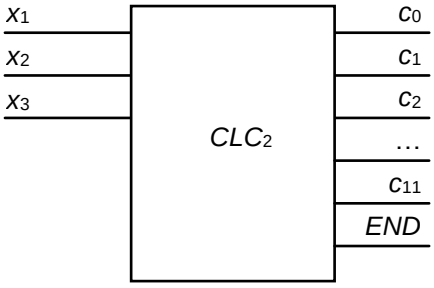


Fig. 3.9. Dispozitivul electronic combinațional care generează semnalele de control în funcție de variabilele de stare.

În acest mod se poate obține unitatea de control din două module de complexitate mai redusă, unul secvențial și unul combinațional, în locul unui singur dispozitiv electronic de complexitate ridicată. Evident, semnalele de intrare ale unității de control complete sunt *START*, *Q*(0) și *COUNT*7, iar ieșirile sale sunt c_0, c_1, …, c_{11} și *END* (**Fig. 3.2**).

Mersul lucrării

1. Să se însușească și să se discute metoda de proiectare a unității de control pe baza tabelului de stări.
2. Să se finalizeze implementarea unității de comandă a multiplicatorului prin metoda tabelului de stări.

Lucrarea 4. Metodă de proiectare a unității de control cablate bazată pe utilizarea elementelor de întârziere pentru secvențierea semnalelor de comandă

Scopul lucrării

Lucrările 3, 4 și 5 își propun însușirea de către studenți a unei metodologii de proiectare a unității de control cablate a unui sistem digital. Scopul acestei lucrări este realizarea unității de comandă pentru dispozitivul de înmulțire a două numere în complement față de 2 prin metoda elementului de întârziere.

Considerații teoretice

Metoda elementelor de întârziere

Considerăm problema generării la momentele de timp t_1, t_2, ... t_n a următoarelor semnale de control, utilizând o unitate de control cablată:

t_1: Activează $\{C_{1,j}\}$;
t_2: Activează $\{C_{2,j}\}$;
. . .
t_n: Activează $\{C_{n,j}\}$;

Presupunem că la momentul t_1 este disponibil un semnal de inițiere numit *START*(t_1). Acesta poate fi legat cu $\{C_{1,j}\}$ pentru a realiza prima microoperație. Dacă *START*(t_1) este intrare pentru un *element de întârziere* cu întârzierea $t_2 - t_1$, atunci ieșirea circuitului, denumită *START*(t_2), poate fi utilizată pentru a activa $\{C_{2,j}\}$. Similar, un alt element de întârziere cu întârzierea $t_3 - t_2$, cu intrarea *START*(t_2), poate fi utilizat să activeze $\{C_{3,j}\}$ ș.a.m.d. Deci, în această manieră, se poate utiliza o secvență de *elemente de întârziere* care să genereze direct semnalele de control.

O unitate de control care utilizează *elementele de întârziere* poate fi construită direct din organigrama de funcționare, care specifică secvența de semnale de control necesară. Ca o consecință a faptului că

circuitul oglindește fluxul semnalelor de control din organigramă, circuitul astfel format are o structură esențial asemănătoare cu cea a organigramei de funcționare.

În **Fig. 4.1** sunt ilustrate câteva reguli simple care indică modul în care circuitul de control este derivat din organigrama de funcționare.

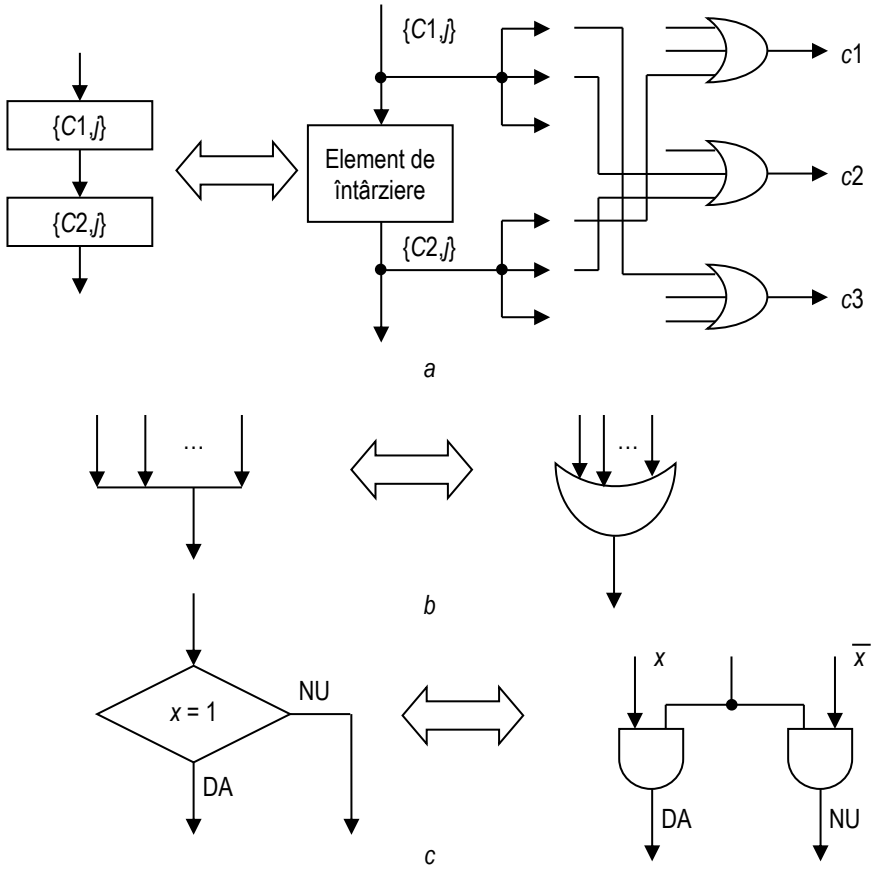


Fig. 4.1. Reguli de transformare a unei organigrame într-un circuit de control, cu ajutorul elementelor de întârziere.

Fiecare secvență de două microoperații succesive necesită câte un element de întârziere. Semnalele care activează liniile de control sunt preluate direct de la intrarea și ieșirea elementului de întârziere, așa cum se arată în **Fig. 4.1.a**. Semnalele care se intenționează să activeze aceeași linie de control c_i sunt duse la intrările unei porți OR a cărei ieșire este c_i. Această linie poate să fie apoi conectată la punctul de control pe care îl activează.

K linii din organigrama de funcționare care se leagă la o linie comună sunt transformate într-o poartă OR cu *k* intrări, ca în **Fig. 4.1.b**.

Un bloc de decizie, care indică un salt condiționat în organigramă, poate fi implementat de două porți AND, așa cum se arată în **Fig. 4.1.c**. Acest circuit AND formează un simplu demultiplexor de 1 bit, controlat de variabila *x* care se testează.

Observație: *x* poate fi înlocuit de o funcție booleană *f*(*x*), astfel încât, pentru determinarea fluxului controlului, se pot utiliza condiții de test de o anumită complexitate.

Fig. 4.2 arată o porțiune dintr-o organigramă tipică de funcționare, care indică semnalele de control $\{C_{i,j}\}$ care trebuie activate în fiecare pas.

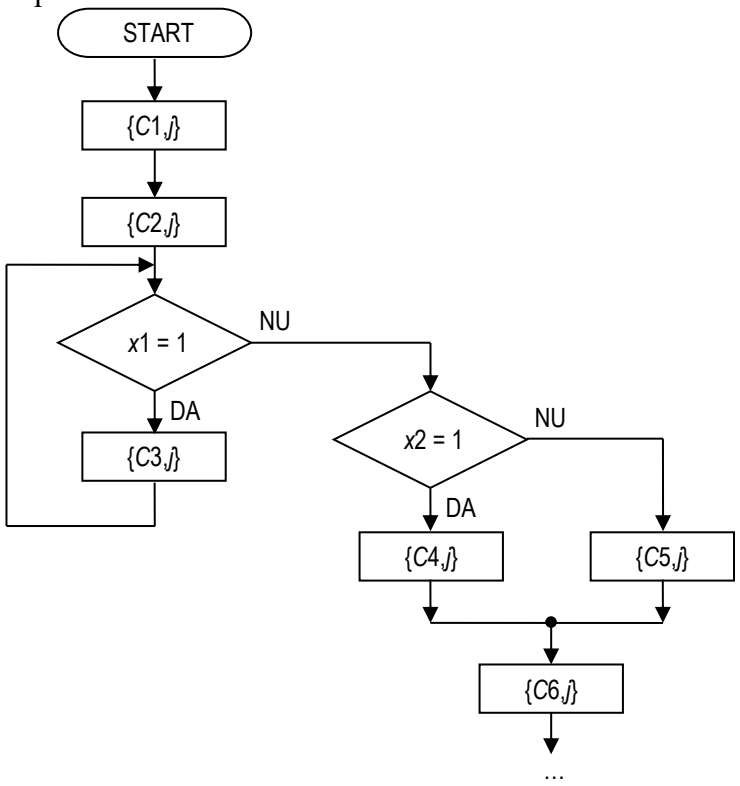


Fig. 4.2. Porțiune dintr-o organigramă care arată activarea semnalelor de control.

Fig.4.3 arată circuitul de control care se obține utilizând aceste reguli de transformare. De remarcat că cele două porți AND, derivate din cele două blocuri de decizie, au fost combinate într-un mod favorabil.

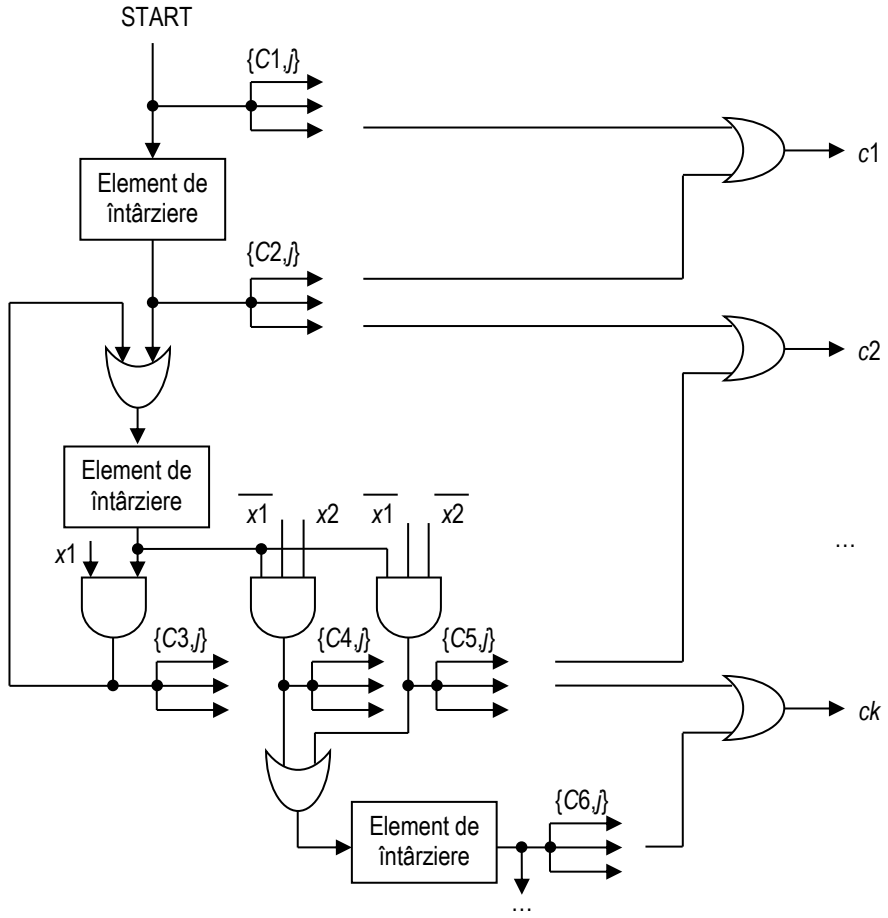


Fig. 4.3. Unitate de control cu elemente de întârziere, care corespunde organigramei din **Fig. 4.2**.

Elementul de întârziere

Elementul de întârziere cerut de astfel de circuite de control trebuie să asigure la ieșire impulsuri de anumită amplitudine și de durată precisă, sincronizate cu ceasul principal al sistemului. Foarte important, acest element NU este o simplă linie pasivă de întârziere.

Dacă toate întârzierile au durata unui impuls de ceas, atunci se folosește ca element de întârziere un bistabil *D* (**Fig. 4.4.a**). Impulsurile de control se presupun a fi de aceeași durată ca impulsurile de ceas. Dacă apare o defazare între impulsul de control de intrare și impulsul de ceas din cauza propagării întârzierilor prin elementele de întârziere, este necesară utilizarea unui circuit de întârziere mai complex (**Fig. 4.4.b**).

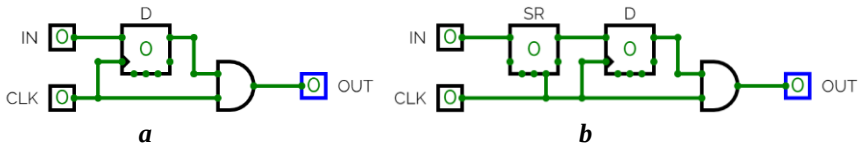


Fig. 4.4. Element de întârziere pentru circuite de control sincrone.

Dacă la intrarea *IN* apare un impuls în timp ce semnalul de tact *CLK* se află în 0L, atunci ieșirea normală a bistabilului SR trece în 1L. La următorul front crescător al semnalului de tact, bistabilul D preia acest 1L și-l transferă la ieșirea sa normală, iar poarta AND îl validează doar atâta timp cât semnalul de tact se află în 1L. În același timp, bistabilul SR este resetat de nivelul 1L al semnalului de tact, pregătindu-se pentru sosirea unui alt impuls. Astfel, impulsul la ieșire este decalat față de cel de la intrare cu o perioadă a semnalului de tact.

În ciuda modului simplu în care se face proiectarea unităților de control, direct pe baza organigramelor lor de funcționare, metoda prezintă o serie de dezavantaje dintre care remarcăm faptul că numărul elementelor implicate este mare. Fiecare element de întârziere definește o stare a unității de control, deci numărul elementelor de întârziere este aproximativ egal cu numărul stărilor n_S. Mai mult, fiecare element de întârziere este un circuit secvențial de complexitate mai mare sau egală cu cea a unui bistabil.

Utilizând metoda de proiectare bazată pe utilizarea tabelelor de stări se poate proiecta un circuit secvențial cu n_S stări, cu mai puțin de $log_{2^n} S$ bistabile, pe când această metodă produce circuite de control scumpe, în care temporizarea este controlată de impulsuri care traversează elemente de întârziere în cascadă, numite „*timing chains*”. Sincronizarea unui număr mare de elemente de întârziere distribuite poate fi de asemenea dificilă.

Unitatea de control a unui multiplicator pentru două numere cu virgulă fixă reprezentate în complement față de 2

În Lucrarea 2 s-a realizat o diagramă bloc a sistemului proiectat, arătându-se în detaliu unitatea de procesare a datelor și punctele de control. În **Fig. 4.5** sunt prezentate conexiunile de intrare/ieșire ale unității de control. Funcțiile semnalelor de control $c_0 \ldots c_{11}$, transmise unității de procesare a datelor, s-au studiat în aceeași lucrare. *Q*(0) este bitul cel mai din dreapta al registrului *Q*. *COUNT7* este un semnal derivat din numărătorul pentru iterare, care este 1L când *COUNT = 7* și 0L în celelalte cazuri.

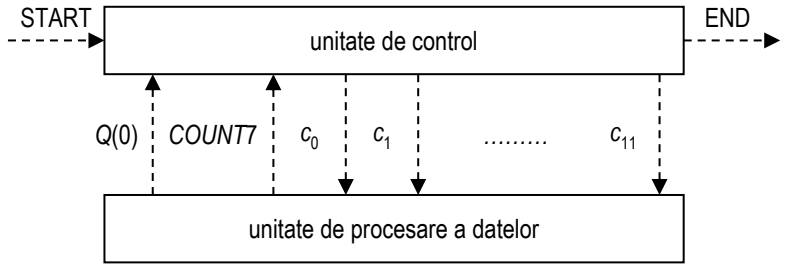


Fig. 4.5. Liniile de intrare/ieșire ale unității de control a multiplicatorului; numărătorul pentru iterare *COUNT* este alocat unității de procesare a datelor.

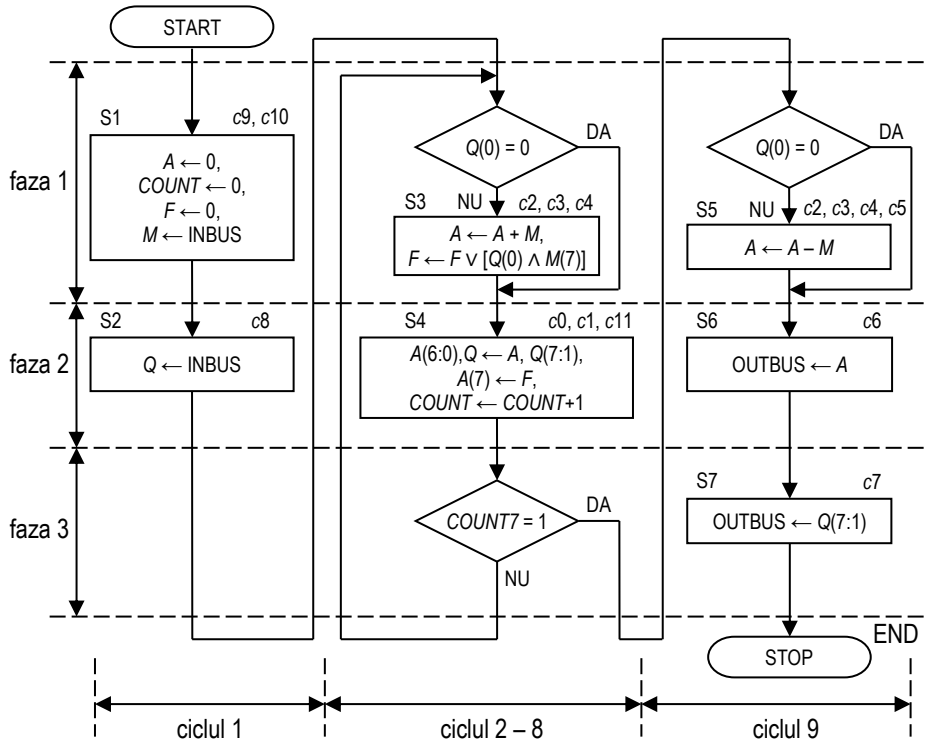


Fig. 4.6. Organigrama pentru înmulțirea în complement față de 2.

Fig. 4.6 redă algoritmul cu semnalele de control care sunt activate în timpul execuției fiecărei microoperații. Pe baza acestuia se va urmări modul de generare a semnalelor de control prin metoda prezentată. Pe baza **Fig. 4.6**, utilizând regulile de transformare prezentate în **Fig. 4.1**, se obține unitatea de control din **Fig. 4.7**. Pe baza unei corespondențe aproximativ 1 la 1 între stări și elementele de întârziere, rezultă un necesar de 7 elemente de întârziere.

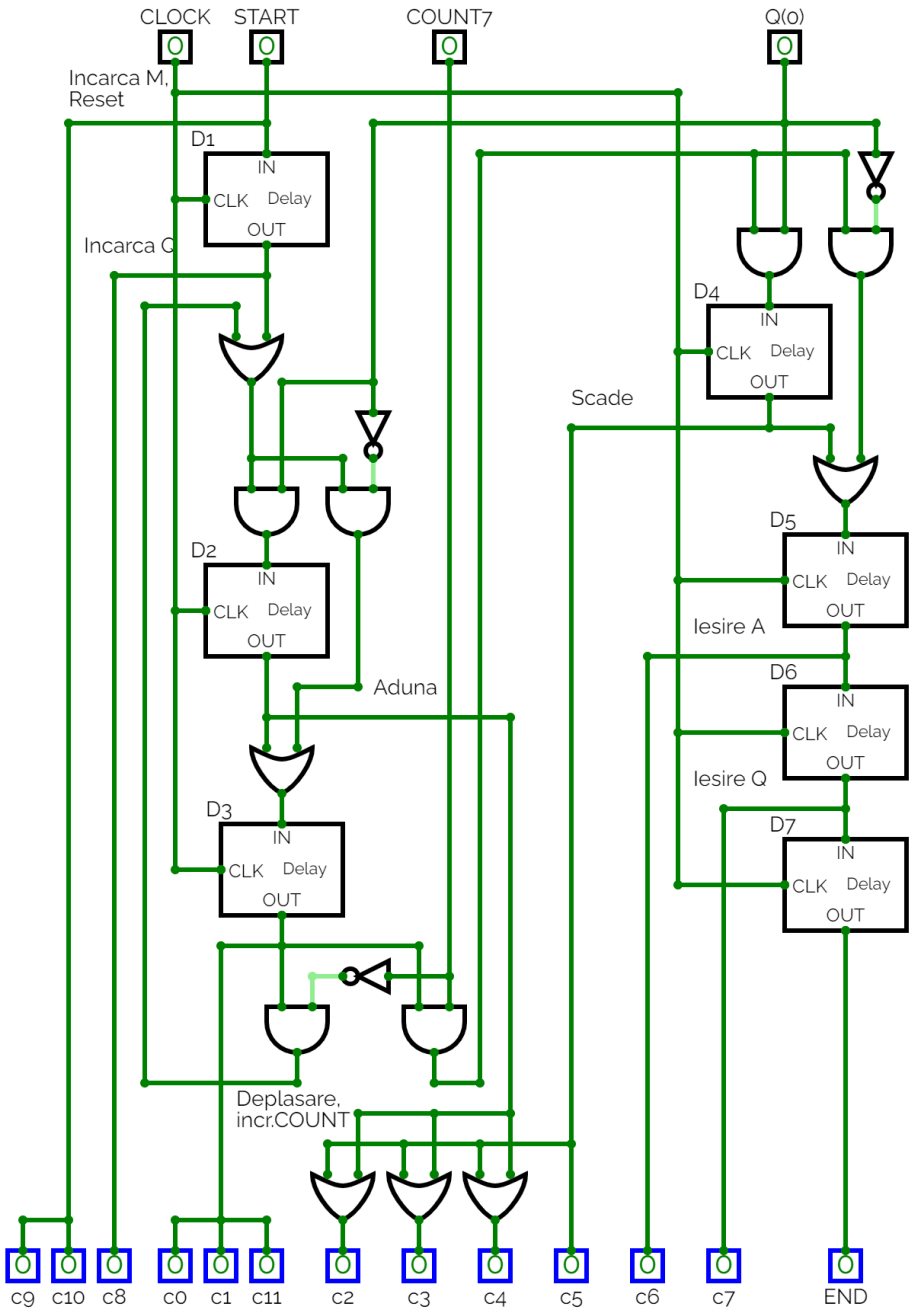


Fig. 4.7. Unitatea de control a multiplicatorului, cu elemente de întârziere.

Circuitul obținut are avantajul că reflectă structura organigramei de funcționare pe baza căreia a fost implementat, fapt care simplifică procesul de proiectare și de mentenanță a circuitului. De remarcat că sunt necesare mai puține circuite combinaționale decât în situația anterioară, deoarece nu este necesar un decodificator pentru identificarea stărilor unității de control.

Mersul lucrării

1. Să se însușească și să se discute metoda de proiectare a unității de control prin utilizarea elementelor de întârziere.
2. Să se realizeze proiectarea unității de comandă prin metoda elementului de întârziere pentru un dispozitiv de împărțire cu refacerea restului.

Lucrarea 5. Metodă de proiectare a unității de control bazată pe utilizarea numărătoarelor pentru secvențierea în timp a semnalelor de control

Scopul lucrării

Lucrarea își propune însușirea de către studenți a unei metodologii de proiectare a unității de control cablate a unui sistem digital prin metoda utilizării numărătoarelor de secvențiere.

Considerații teoretice

Se consideră dispozitivul din **Fig. 5.1**, care constă dintr-un circuit de bază numărător modulo *k*, ale cărui ieșiri sunt conectate la un decodificator 1/*k*.

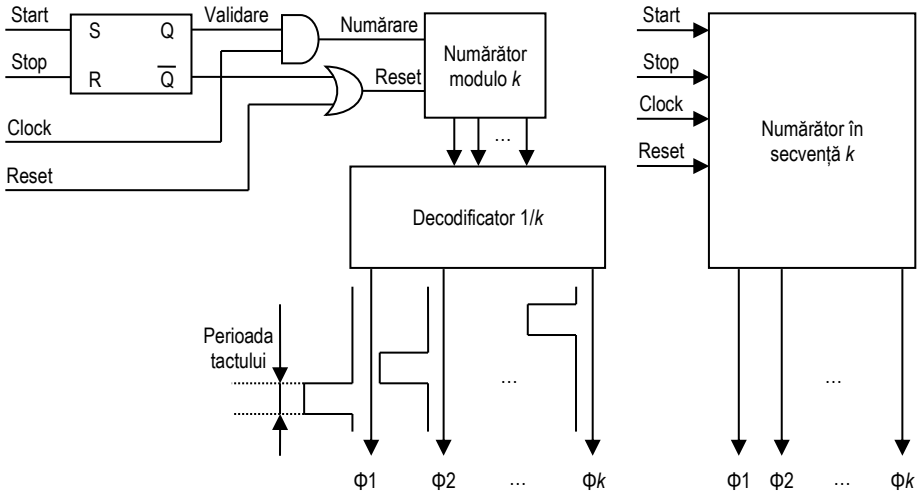


Fig. 5.1. Numărător în secvență modulo *k* cu ieșiri decodificate; ***a*** – diagrama logică, ***b*** – simbolul numărătorului.

Dacă intrarea de validare a numărătorului este conectată la o sursă de tact, numărătorul ciclează conținutul cele *k* stări ale sale. Decodificatorul generează pe liniile sale de ieșire *k* impulsuri de semnale $\{\phi_i\}$. Impulsurile consecutive sunt separate de câte o perioadă de tact, așa cum este arătat în **Fig. 5.1.a**. Semnalele $\{\phi_i\}$ împart efectiv timpul

necesar unui ciclu complet de numărare în *k* părți egale; din acest motiv, $\{\phi_i\}$ pot fi numite *semnale de fază*.

Pentru validarea sau invalidarea numărătorului sunt prevăzute două intrări adiționale și un bistabil. Un impuls pe linia de *START* pornește numărarea, intrarea de numărare a numărătorului fiind astfel conectată la sursa de semnal de tact *CLOCK*. Un impuls pe linia de *STOP* deconectează semnalul de tact și resetează numărătorul.

Dispozitivul ilustrat în **Fig. 5.1.a** se numește *numărător în secvență* și va fi reprezentat prin simbolul din **Fig. 5.1.b**. Utilitatea acestor numărătoare rezidă din faptul că multe circuite digitale sunt proiectate pentru a realiza un număr relativ mic de acțiuni repetate. Această comportare poate fi descrisă la un nivel înalt printr-o organigramă de funcționare care constă dintr-o singură buclă cu *k* pași.

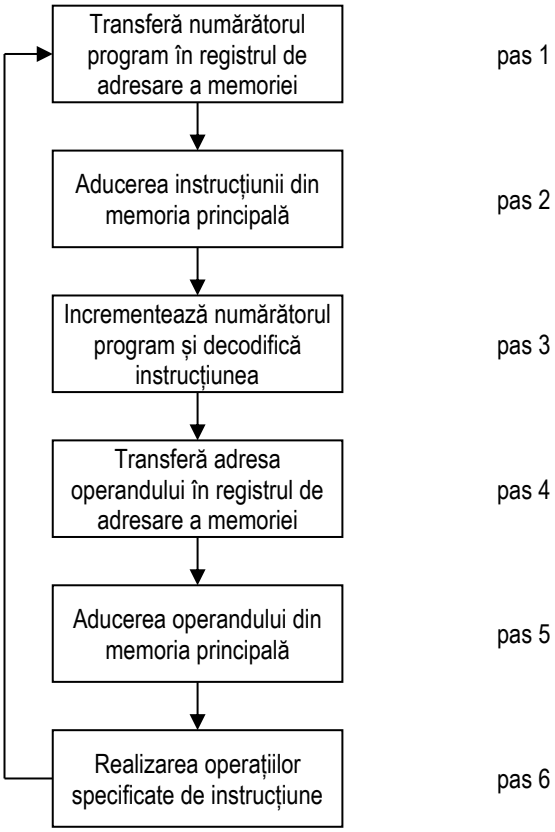


Fig. 5.2. Comportarea CPU-ului, reprezentată într-o singură buclă.

De exemplu, **Fig. 5.2** arată o organigramă de funcționare cu o buclă conținând 6 pași care descrie funcționarea unui CPU tipic (***C****entral*

***P**rocessing **U**nit*). Fiecare trecere prin buclă constituie un ciclu instrucțiune. Presupunând că fiecare pas poate fi realizat într-o perioadă de ceas, aleasă convenabil, pentru acest CPU se poate considera o unitate de control bazată pe un singur numărător în secvență modulo 6.

În pasul *i* al fiecărui ciclu de instrucțiune, fiecare semnal de fază ϕ_i activează un anumit set de linii de control. În general, este necesar să se poată varia operațiile realizate în pasul *i* în funcție de anumite semnale de control sau variabile de control aplicate unității de control. Acestea sunt reprezentate în **Fig. 5.3** de semnalele $C_{in} = \{C_{in}', C_{in}''\}$. După cum rezultă din **Fig. 5.3**, este necesar un circuit logic *N* care combină semnalele din mulțimea C_{in} cu semnalele de fază $\{\phi_i\}$, generate de numărătorul în secvență.

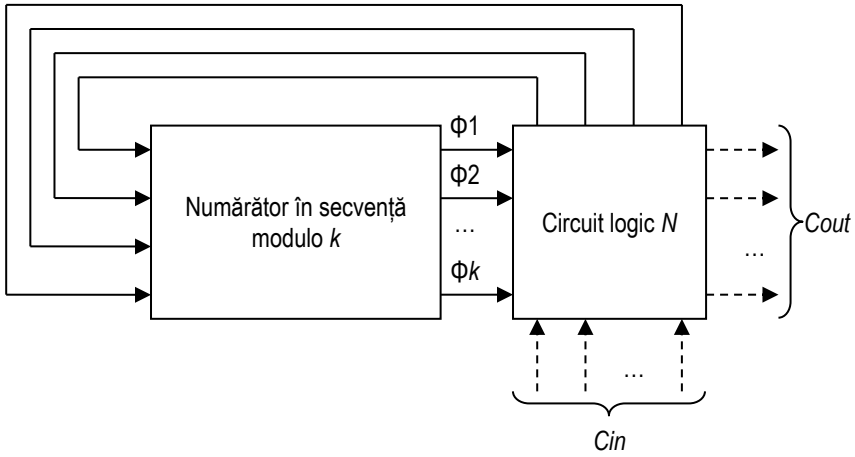


Fig. 5.3. Unitate de control bazată pe numărător în secvență.

Unitatea de control a unui multiplicator pentru două numere cu virgulă fixă reprezentate în complement față de 2

În Lucrarea nr. 2 s-a realizat o diagramă bloc a sistemului proiectat, arătându-se în detaliu unitatea de procesare a datelor și punctele sale de control. În **Fig. 5.4** sunt prezentate conexiunile de intrare/ieșire ale unității de control. Funcțiile semnalelor de control $c_0 \ldots c_{11}$, transmise unității de procesare a datelor, s-au studiat tot în Lucrarea nr. 2. *Q*(0) este bitul cel mai din dreapta (LSB) al registrului *Q*. *COUNT7* este un semnal derivat din numărătorul pentru iterare, care este 1L când *COUNT = 7* și 0L în celelalte cazuri.

Fig. 5.5 redă algoritmul cu semnalele de control care sunt activate în timpul execuției fiecărei microoperații. Pe baza acestuia se va urmări

modul de generare al semnalelor de control prin metoda utilizării numărătorului în secvență.

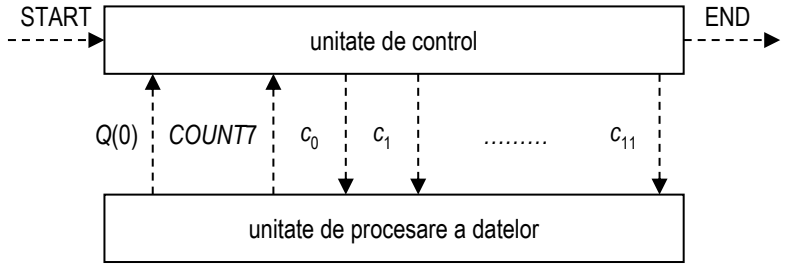


Fig. 5.4. Liniile de intrare/ieșire ale unității de control a multiplicatorului; numărătorul pentru iterare COUNT este alocat unității de procesare a datelor.

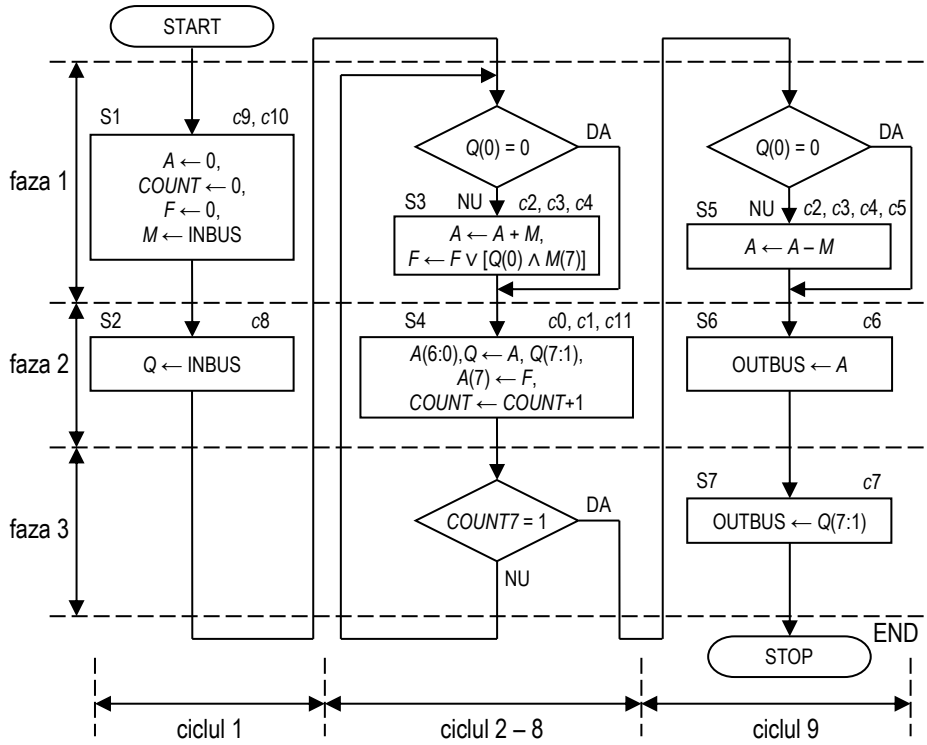


Fig. 5.5. Organigrama pentru înmulțirea în complement față de 2.

Organigrama de funcționare a multiplicatorului arată existența unei singure bucle închise, care cuprinde 3 pași: *adunare*, *deplasare* și *incrementare*. Astfel, proiectarea unității de control se bazează pe un numărător în secvență modulo 3. Pentru numerele de *n* biți bucla este parcursă de (*n*–1) ori, deci în cazul nostru bucla este parcursă de 7 ori.

Algoritmul implică anumiți pași care nu fac parte din bucla principală. La început sunt necesare două perioade *clock* pentru resetarea unității de control și încărcarea operanzilor de intrare. La sfârșitul algoritmului sunt necesare trei perioade *clock* pentru corectarea rezultatului când înmulțitorul este negativ și pentru transferarea produsului pe magistrala de ieșire. Acești pași de inițializare și terminare se realizează în două cicluri externe secvenței de numărare, după cum este indicat în **Fig. 5.5**. Deci execuția unei înmulțiri se realizează în nouă cicluri, astfel:

Etapa 1 (ciclul 1): inițializează unitatea de control și încarcă operanzii;
Etapa 2 (ciclurile 2 – 8): realizează produsul;
Etapa 3 (ciclul 9): corectează produsul, dacă este necesar, și descarcă rezultatul.

Pentru distingerea acestor trei etape, este necesar să se introducă bistabile care pot fi setate pentru identificarea acestora. În **Fig. 5.6** se arată o proiectare bazată pe aceste principii. Numărătorul în secvență modulo 3 asigură principalele semnale pentru temporizare.

Au fost introduse trei bistabile SR pentru identificarea celor trei etape ale algoritmului. Fiecare este activat la începutul etapei respective și resetat la sfârșit. Un set de porți AND identifică microoperațiile particulare care trebuie realizate în fiecare perioadă de ceas. Intrările acestor porți sunt conectate la trei surse:

- la numărătorul în secvență;
- la cele trei bistabile;
- la semnalele externe de control ale unității de control.

Intrările necesare fiecărei porți AND se determină ușor din **Fig. 5.5**. De exemplu, pentru încărcarea registrului *Q* de pe magistrala de intrare, unitatea de control trebuie să fie în etapa 1 cu ϕ_2 activ, deci poarta *AND*2 este conectată la ieșirea normală a bistabilului *SR*1 și la ϕ_2. Semnalele de control $c_0 \ldots c_{11}$ și *END* sunt derivate din ieșirile porților AND, care sunt reunite prin porți OR atunci când una sau mai multe microoperații distincte necesită activarea acelorași linii de control.

Terminarea fiecărei etape este semnalată de detectorul de front negativ format din porțile 14 și 15, care sesizează dezactivarea fazei ϕ_3. Semnalul END resetează întregul sistem. Din acest motiv, el trebuie emis în etapa 3 la dezactivarea fazei ϕ_3, după emisia semnalului c_7.

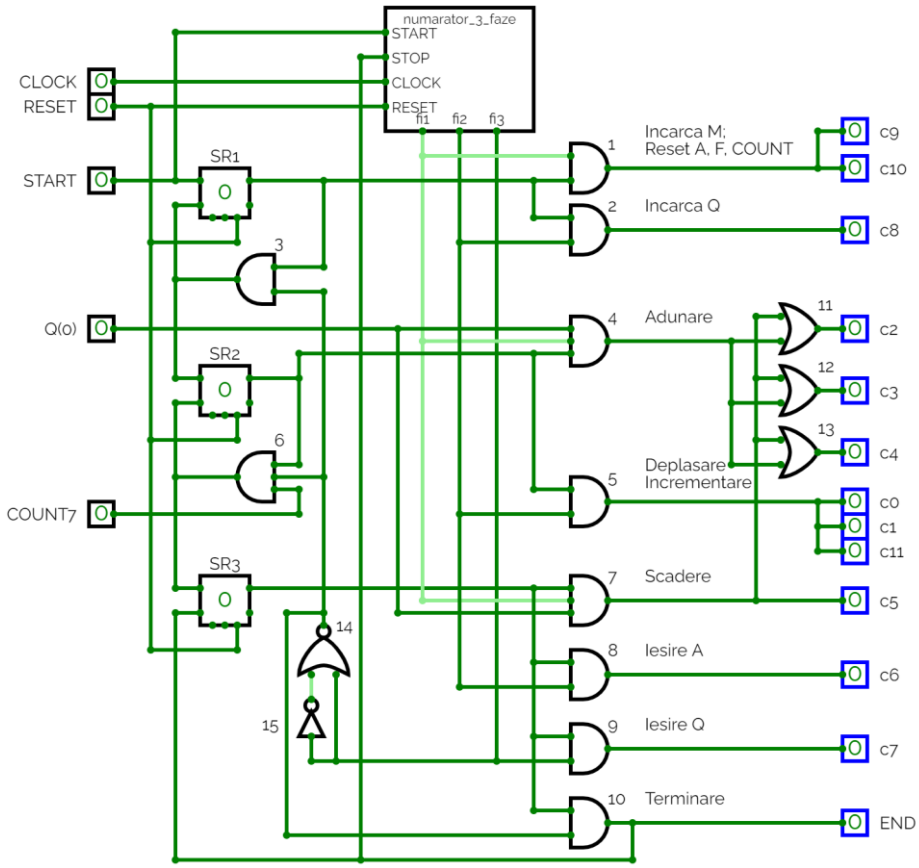


Fig. 5.6. Unitatea de control a multiplicatorului, utilizând numărător de faze (în secvență).

Mersul lucrării

1. Să se însușească și să se discute metoda de proiectare a unității de control prin utilizarea numărătoarelor în secvență.
2. Să se realizeze proiectarea unității de comandă pentru un dispozitiv de împărțire cu refacerea restului prin metoda utilizării numărătoarelor în secvență.

Lucrarea 6. Proiectarea unității de control cablate a unității centrale de prelucrare a unui calculator cu structură bazată pe acumulator / Lucrarea 7. Proiectarea unității de procesare a datelor pentru o unitate centrală de prelucrare a unui calculator cu structură bazată pe acumulator

Scopul lucrărilor

În cadrul Lucrării 6, pe baza metodelor de proiectare prezentate în lucrările anterioare, se dorește proiectarea în detaliu a unității de control a unității centrale de procesare (CPU) pentru un calculator pe 8 biți cu structura bazată pe acumulator. Unitatea de procesare a datelor pentru acest CPU va fi proiectată în cadrul Lucrării 7.

Considerații teoretice

O unitate centrală de procesare (*Central Processing Unit* – CPU) poate conține sute de linii de control, ceea ce face proiectarea sa destul de anevoioasă. În această lucrare se examinează sumar problemele de proiectare care apar, considerându-se pentru exemplificare cel mai simplu CPU posibil.

Descrierea CPU

Se consideră organizarea ipotetică a CPU-ului pe 8 biți prezentat în **Fig. 6.2**. Presupunem că este necesar să execute setul de 8 instrucțiuni (cu o adresă) prezentat în **Fig. 6.1**.

Mnemonic	Descriere
LOAD X	AC ← M(X) (se transferă conținutul locației de memorie X în acumulator)
STORE X	M(X) ← AC
ADD X	AC ← AC + M(X)
AND X	AC ← AC ∧ M(X) (ȘI logic)
JUMP X	PC ← X (salt necondiționat)
JUMPZ X	if AC = 0 then PC ← X (salt condiționat)
NOT	AC ← $\overline{AC}$ (inversează toți biții acumulatorul)
RSHIFT	deplasarea la dreapta a acumulatorului

Fig. 6.1. Instrucțiunile cunoscute de un CPU ipotetic.

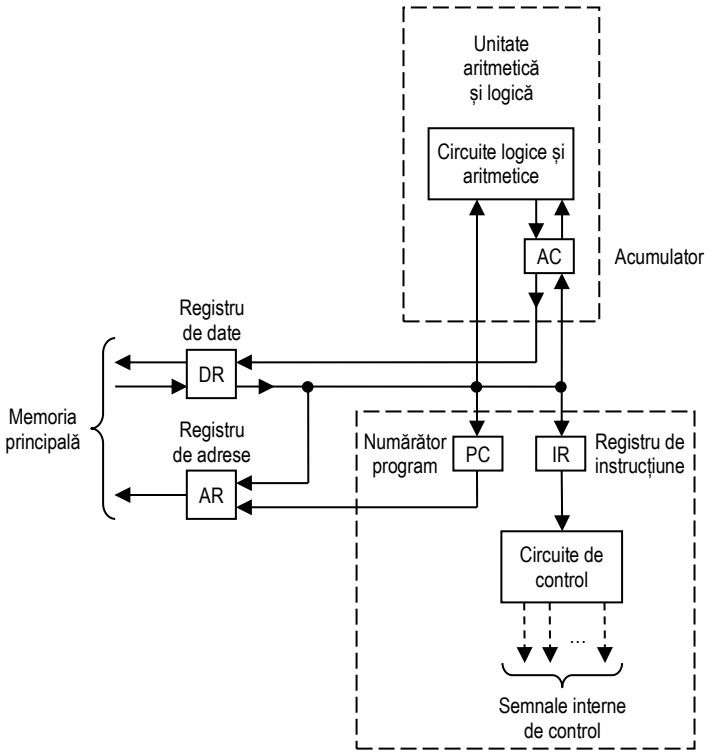


Fig. 6.2. Structura unui CPU simplu bazat pe acumulator.

Pentru structura hardware dată, algoritmii pentru implementarea fiecărei instrucțiuni sunt ușor de dedus. În **Fig. 6.3**, sub formă de organigramă, se prezintă ciclul de aducere, comun tuturor instrucțiunilor, precum și ciclurile de execuție, distincte, necesare executării fiecărei instrucțiuni. Microoperațiile din această organigramă determină semnalele de control și punctele de control necesare CPU-ului. Semnalul ACZ are valoarea 1L dacă AC = 0 și 0L dacă AC ≠ 0.

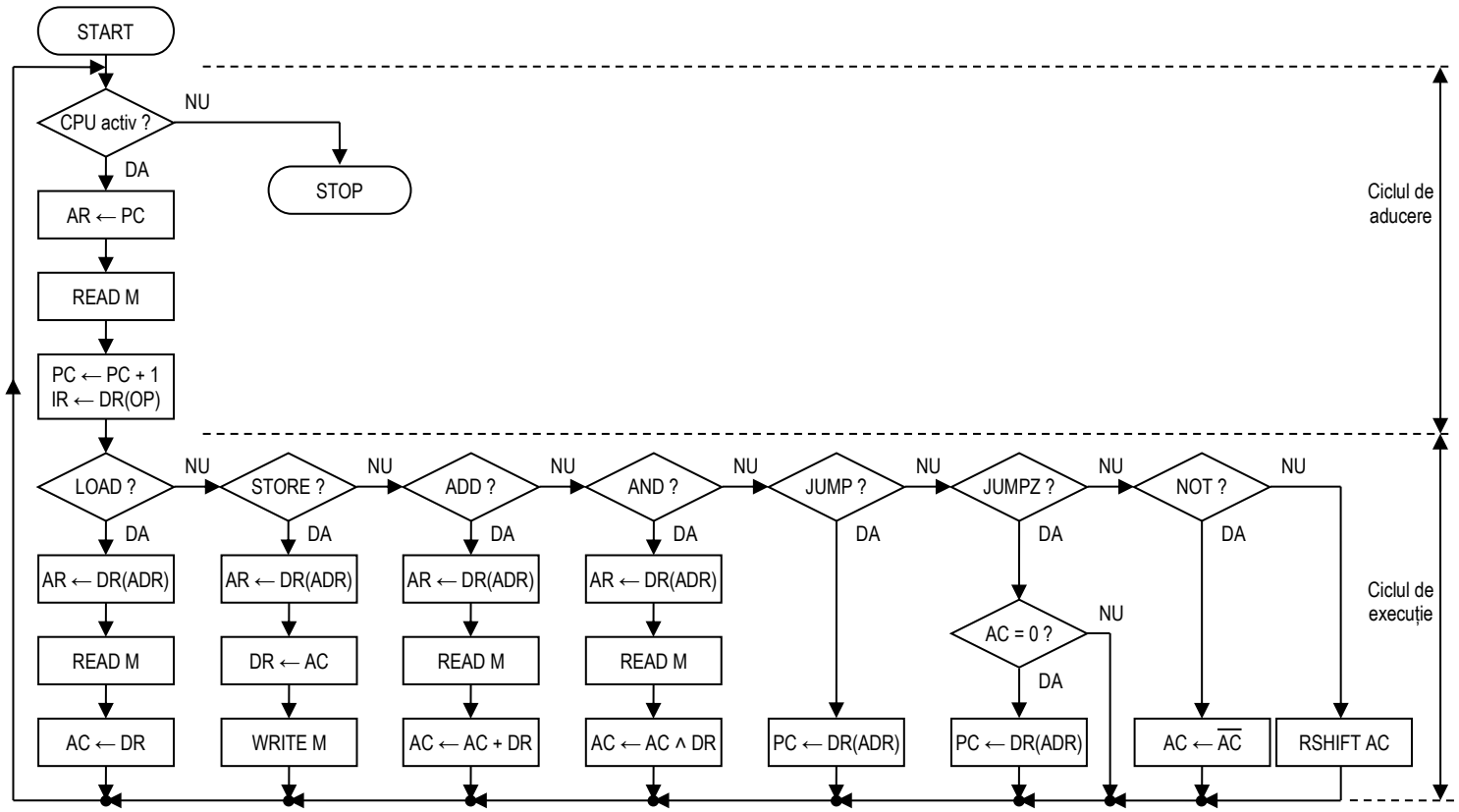


Fig. 6.3. Operarea CPU-ului cu 8 instrucțiuni.

Fig. 6.4 prezintă un set de semnale de control și funcțiile lor de control, în timp ce **Fig. 6.5** arată poziția aproximativă a liniilor de control corespondente în CPU.

Semnal de control	Microoperație controlată
c_0	AC ← AC + DR
c_1	AC ← AC ∧ DR
c_2	AC ← $\overline{AC}$
c_3	DR ← M(AR)
c_4	M(AR) ← DR
c_5	DR ← AC
c_6	AC ← DR
c_7	AR ← DR(ADR)
c_8	PC ← DR(ADR)
c_9	PC ← PC + 1
c_{10}	AR ← PC
c_{11}	IR ← DR(OP)
c_{12}	Deplasare la dreapta a lui AC (Right-shift)

Fig. 6.4. Semnalele de control pentru CPU.

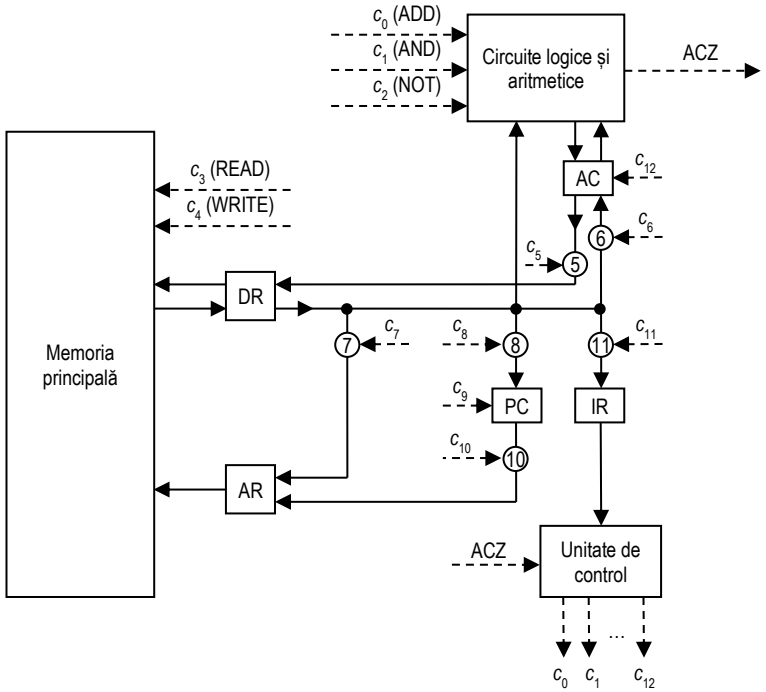


Fig. 6.5. Structura unui CPU simplu.

Implementare

Microoperațiile realizate de CPU apar ca o buclă formată din 6 pași (**Fig. 6.6**). Primii 3 pași formează ciclul de aducere și sunt identici pentru toate instrucțiunile. Ceilalți pași depind de instrucțiunea care se execută, ceea ce sugerează proiectarea eficientă a unității de control prin metoda numărătorului în secvență (este singura metodă pe care o luăm în considerare în cadrul prezentării teoretice).

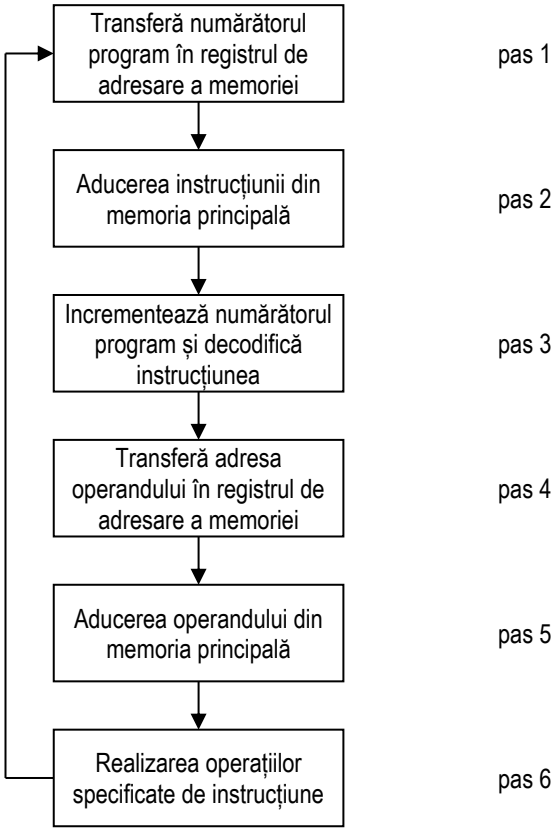


Fig. 6.6. Comportarea CPU reprezentată printr-o singură buclă.

Presupunem că toate instrucțiunile, cu excepția lui READ M și WRITE M se execută într-o singură unitate de timp, a cărei durată este aleasă în mod potrivit. Pe de altă parte, presupunem că READ M și WRITE M pot fi terminate în două unități de timp. Din analiza organigramei se observă că instrucțiunile lente (ex. ADD), necesită 8 unități de timp, care se împart egal între ciclul de aducere și cel de execuție, pe când instrucțiuni ca JUMP necesită doar 5 unități de timp, 4 în ciclul de aducere și una în ciclul de execuție. Deci, unitatea de

control poate fi proiectată eficient cu ajutorul unui numărător în secvență modulo 8, ale cărui impulsuri de numărare au o perioadă egală cu o unitate de timp. **Fig. 6.7** prezintă structura generală a unei unități de control cablate simple.

Din organigrama de funcționare a CPU-ului se determină care semnale de control se activează în fiecare moment al ciclului instrucțiune pentru fiecare instrucțiune. De exemplu, semnalul c_3, care validează execuția unei operații de citire, este activat când $\phi_2 = 1$ pentru a aduce o instrucțiune. Acesta este de asemenea activat pentru aducerea unui operand, când $\phi_6 = 1$ la instrucțiunile LOAD, ADD și AND (ieșirea decodificatorului de instrucțiuni indică execuția acestor instrucțiuni). Prin urmare, c_3 poate fi definit prin ecuația:

$$c_3 = \phi_2 + \phi_6 \times (\text{LOAD} + \text{ADD} + \text{AND})$$

care este implementată de circuitul combinațional *N* din **Fig. 6.7**.

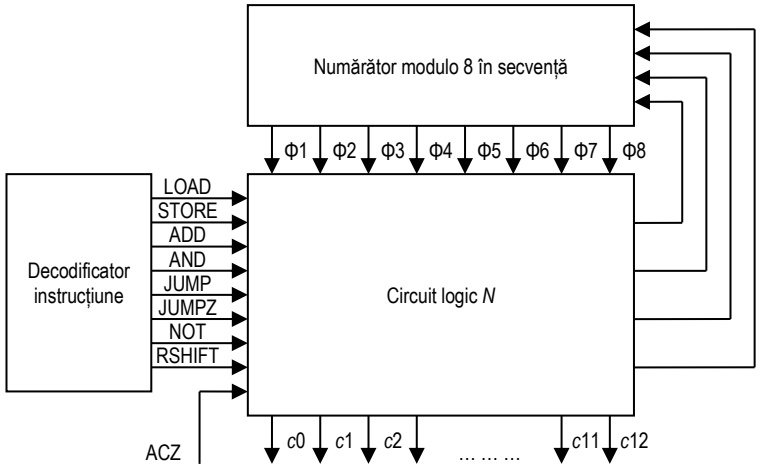


Fig. 6.7. Unitate de control pentru CPU, care utilizează un numărător în secvență.

În general, fiecare semnal de control c_i poate fi definit de o ecuație logică de forma:

$$c_i = \sum_i \left(\varphi_i \cdot \sum_m I_m \right)$$

unde I_m este o ieșire a decodificatorului de instrucțiuni.

În cazul unei instrucțiuni care necesită $j < k$ pași (unde k este modulul numărătorului), după pasul j trebuie resetat numărătorul în secvență.

Mersul lucrărilor

Pentru Lucrarea nr. 6:

1. Să se scrie ecuațiile logice pentru toate semnalele de comandă.
2. Să se proiecteze în detaliu circuitul combinațional *N* din **Fig. 6.7**.

Pentru Lucrarea nr. 7:

Să se proiecteze partea de procesare a datelor pentru CPU-ul considerat.

Lucrarea 8. Proiectarea unității de control microprogramate pentru un dispozitiv de înmulțire a două numere în reprezentare cu virgulă fixă

Scopul lucrării

Scopul Lucrării nr. 8 este însușirea unei strategii de proiectare a unei unități de control microprogramate.

Considerații teoretice

Noțiuni de bază ale microprogramării

Fiecare instrucțiune a unei CPU este implementată printr-o secvență de una sau mai multe seturi de *microoperații*. Fiecare microoperație este asociată cu un anumit set de linii de control care, când sunt activate, cauzează realizarea microoperației. Întrucât numărul instrucțiunilor și al liniilor de control este adeseori foarte mare, putând atinge ordinul sutelor, o unitate de control cablată, care selectează și secvențiază semnalele de control, poate fi extrem de complicată, fiind dificil și costisitor de proiectat. Mai mult, o asemenea unitate nu prezintă flexibilitate pentru schimbările cerute de reproiectarea unității, de exemplu, pentru situația corectării erorilor de proiectare sau a schimbării setului de instrucțiuni.

Microprogramarea este o metodă de proiectare a unității de control în care selectarea și secvențierea informației este memorată într-o memorie cu acces aleator, numită *memorie de control* (***C**ontrol **M**emory* – CM). Semnalele de control care se activează la un moment dat sunt specificate de un cuvânt numit *microinstrucțiune*, care este adus din CM în aceeași manieră în care se face aducerea unei instrucțiuni din memoria principală. De asemenea, fiecare microinstrucțiune specifică în mod explicit sau în mod implicit următoarea microinstrucțiune care se folosește, asigurând astfel informația necesară pentru secvențierea microinstrucțiunilor. Un set de microinstrucțiuni corelate, formează un *microprogram*.

Microprogramele, fiind realizate soft, pot fi modificate relativ ușor. Astfel, prin microprogramare se realizează unități de control mult mai flexibile decât cele realizate cablat. Această flexibilitate este atinsă în detrimentul costului hardware-ului care, datorită costului implicat de

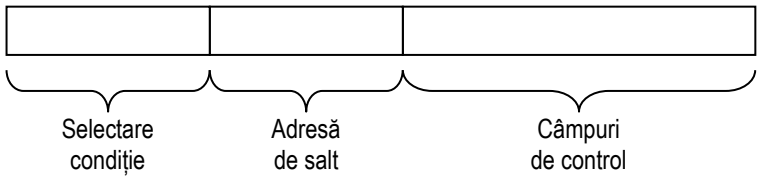
memoria de comandă și de circuitele sale de acces, devine mai mare comparativ cu situația unităților cablate. Pe de altă parte, datorită timpului implicat pentru accesarea microinstrucțiunilor din CM, apare și o scădere a performanțelor de viteză. Odată cu dezvoltarea în ultima perioadă a tehnologiilor de realizare a memoriilor de viteză mare și cost scăzut, potrivite pentru implementarea memoriilor de comandă, aceste dezavantaje au fost mult diminuate.

Într-un CPU microprogramat, fiecare instrucțiune mașină este executată de un microprogram care acționează pentru fiecare instrucțiune ca un interpretor în timp real. Un set de microprograme care interpretează un set particular de instrucțiuni, sau *limbaj L*, este numit *emulator pentru L*. O microinstrucțiune, în forma ei cea mai simplă, cea propusă de *Wilkes*, are două părți majore:

- un set de *câmpuri de control*, care indică liniile de control care trebuiesc activate;
- un *câmp de adresă*, care indică adresa din CM pentru următoarea microinstrucțiune care trebuie executată.

Unitatea de control a multiplicatorului

Se consideră dispozitivul de înmulțire în virgulă fixă a două numere în reprezentare complement față de 2, pentru care se va proiecta o unitate de control microprogramată. Punctul de plecare este constituit de organigrama de funcționare din **Fig. 8.1**. Pentru început se va utiliza formatul de microinstrucțiune:



care conține trei câmpuri:

- ✓ un câmp de selectare a condiției;
- ✓ un câmp al adresei de salt;
- ✓ un câmp de control.

Inițial nu se consideră nici un fel de codificare pentru câmpul de control. Deci sunt necesari 13 biți de control: unul pentru fiecare linie de control $c_0 \ldots c_{11}$ și unul pentru semnalul *END*.

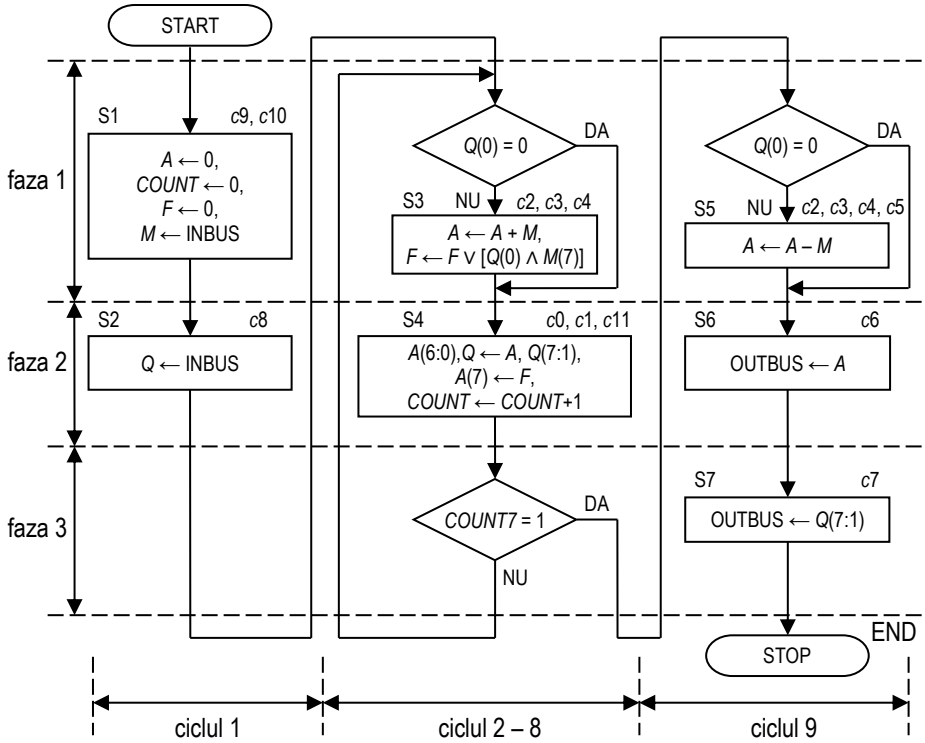


Fig. 8.1. Organigrama pentru înmulțirea în complement față de 2.

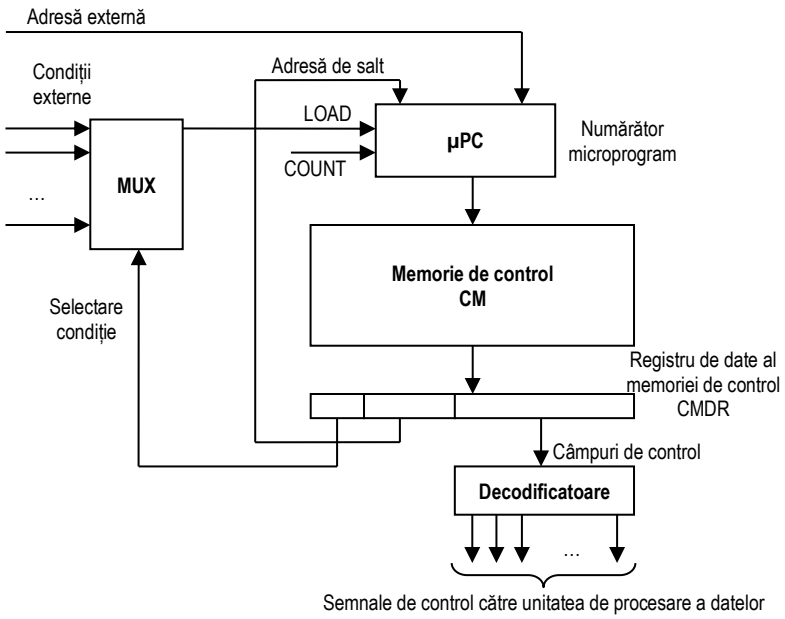


Fig. 8.2. Unitate de control tipică microprogramată.

Unitatea de control va avea organizarea generală din **Fig. 8.2**, care folosește un numărător microprogram μPC ca registru de adresare a memoriei de control. În timpul fiecărui ciclu microinstrucțiune, μPC este incrementat pentru a forma adresa următoarei microinstrucțiuni. În cazul unei instrucțiuni de salt se folosește ca adresă următoare adresa memorată în cuvântul microinstrucțiune. Necesitatea folosirii unei intrări pentru forțarea unei adrese externe este eliminată prin memorarea primei microinstrucțiuni la adresa „0" în CM și prin resetarea μPC la „0" la începutul fiecărei înmulțiri.

Este utilă redescrierea funcționării multiplicatorului cu ajutorul limbajului de descriere (pseudocod), întrucât această descriere reprezintă microprogramul unității scris într-o formă simbolică abstractă (**Fig. 8.3**).

```
            Declare registers A[8], M[8], Q[8], F[1], COUNT[3];
            Declare buses INBUS[8], OUTBUS[8];
START:      A ← 0, F ← 0, COUNT ← 0, M ← INBUS; [deînmulțit]
            Q ← INBUS; [înmulțitor]
TEST1:      if Q(0) = 0 then go to RSHIFT;
            A ← A + M, F ← F ∨ [Q(0) ∧ M(7)];
RSHIFT:     A(6:0),Q ← A,Q(7:1), A(7) ← F, COUNT ← COUNT+1;
            if COUNT = 7 then go to TEST2;
            go to TEST1;
TEST2:      if Q(0) = 0 then go to FINISH;
            A ← A – M;
FINISH:     OUTBUS ← A; [biții cei mai semnificativi]
            OUTBUS ← Q(7:1); [biții cei mai puțin semnificativi]
            STOP;
```

Fig. 8.3. Programul pentru multiplicatorul a două numere.

Fiecare declarație fără salt de mai sus, respectiv fiecare dreptunghi cu microoperații din **Fig. 8.1**, necesită o microinstrucțiune distinctă. În concluzie, este necesar un microprogram cu aproximativ 10 microinstrucțiuni. Este suficient un câmp de adresă de salt de 4 biți. Dacă se dorește, fiecare microinstrucțiune poate conține o adresă de salt; astfel se implementează salturile condiționate și necondiționate *go to*. Câmpul de selectare a condiției este necesar pentru a indica 4 condiții:

1. niciun salt
2. salt dacă $Q(0) = 0$
3. salt dacă *COUNT7* = 1
4. salt necondiționat

Deci, pentru acest câmp sunt suficienți 2 biți. În concluzie, se va folosi o microinstrucțiune cu format orizontal de 19 biți lungime (**Fig. 8.4**).

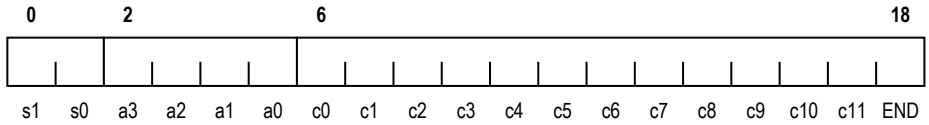


Fig. 8.4. Format de microinstrucțiune necodificat pentru înmulțirea în complement față de 2.

Microprogramul care implementează înmulțirea este ușor de scris. Se convertește linie cu linie microprogramul simbolic, în *pattern*-urile de biți din **Fig. 8.5**.

Adresa în CM	Selectare condiție	Adresă de salt	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	END	Comentarii
0000 (0)	00	0000	0	0	0	0	0	0	0	0	0	1	1	0	0	$A \leftarrow 0$, $F \leftarrow 0$, *COUNT* $\leftarrow 0$, $M \leftarrow$ INBUS
0001 (1)	00	0000	0	0	0	0	0	0	0	0	1	0	0	0	0	$Q \leftarrow$ INBUS
0010 (2)	01	0100	0	0	0	0	0	0	0	0	0	0	0	0	0	if $Q(0) = 0$ then go to 4
0011 (3)	00	0000	0	0	1	1	1	0	0	0	0	0	0	0	0	$A \leftarrow A + M$, $F \leftarrow F \vee [Q(0) \wedge M(7)]$
0100 (4)	10	0110	1	1	0	0	0	0	0	0	0	0	0	1	0	$A(6:0), Q \leftarrow A, Q(7:1)$, $A(7) \leftarrow F$, *COUNT* $\leftarrow$ *COUNT* + 1 if *COUNT7* = 1 then go to 6
0101 (5)	11	0010	0	0	0	0	0	0	0	0	0	0	0	0	0	go to 2
0110 (6)	01	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	if $Q(0) = 0$ then go to 8
0111 (7)	00	0000	0	0	1	1	1	1	0	0	0	0	0	0	0	$A \leftarrow A - M$
1000 (8)	00	0000	0	0	0	0	0	0	1	0	0	0	0	0	0	OUTBUS $\leftarrow A$
1001 (9)	00	0000	0	0	0	0	0	0	0	1	0	0	0	0	0	OUTBUS $\leftarrow Q(7:1)$
1010 (10)	00	0000	0	0	0	0	0	0	0	0	0	0	0	0	1	Semnal sfârșit operație
1011 (11)	11	1011	0	0	0	0	0	0	0	0	0	0	0	0	0	STOP

Fig. 8.5. Microprogramul de înmulțire care utilizează formatul de microinstrucțiune din **Fig. 8.4**.

Microinstrucțiuni consecutive sunt alocate la adrese consecutive și sunt inserați biții potriviți de selectare ai condiției. 00 înseamnă niciun salt, iar restul codurilor de condiție se deduc ușor din **Fig. 8.5**. Când operația de înmulțire s-a terminat, microprogramul introduce o stare de așteptare, prin repetarea executării microinstrucțiunii neoperante din CM cu adresa 1011, în care rămâne până la sosirea unui nou semnal extern *START*. O altă variantă ar fi să se introducă un bistabil care să valideze operarea cu CM, care este setat (resetat) de semnalul *START* (*END*).

Structura unității de control este arătată în **Fig. 8.6**. Se observă că sunt necesare foarte puține din cele 2^{13} *pattern*-uri permise de către câmpul de control al formatului cuvântului microinstrucțiune din **Fig. 8.4**, precum și faptul că anumite seturi de semnale de control sunt întotdeauna activate simultan. Deci, este suficient să rezervăm pentru aceste semnale un singur bit în câmpul de control.

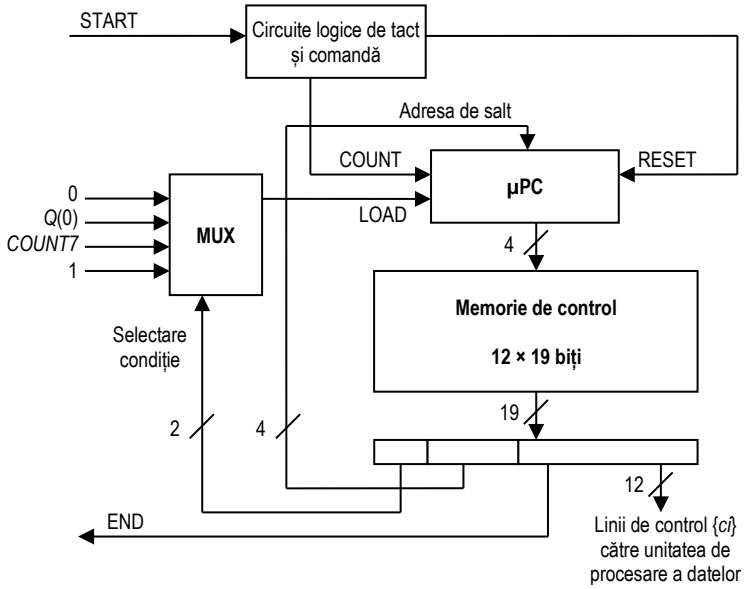


Fig. 8.6. Unitate de control microprogramată pentru înmulțirea a două numere în complement față de 2.

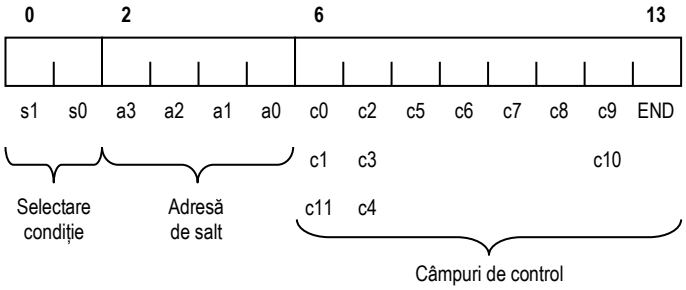


Fig. 8.7. Formatul microinstrucțiune necodificat pentru multiplicatorul în complement față de 2, după extragerea câmpurilor de control redundante.

Asta înseamnă că cei 8 biți rezervați pentru cele 3 seturi: $\{c_0, c_1, c_{11}\}$, $\{c_2, c_3, c_4\}$ și $\{c_9, c_{10}\}$, pot fi înlocuiți de 3 biți. Formatul cuvântului microinstrucțiune se scurtează ca în **Fig. 8.7**.

Codificarea câmpului de control

Prin codificare se obține o mai mare reducere a mărimii câmpului de control. Întrucât sunt 12 microoperații distincte în microprogramul de înmulțire, se poate codifica informația de control la un format pur vertical într-un singur câmp de control de 4 biți, ceea ce limitează posibilitatea modificării setului de microinstrucțiuni.

Presupunem că formatul microinstrucțiune care se proiectează va fi folosit pentru mai multe aplicații, respectiv dispozitivul de înmulțire controlat are toate componentele esențiale ale unei unități aritmetico-logice generale. Deci, prezintă interes codificarea microinstrucțiunilor de așa manieră încât microinstrucțiunile încă nespecificate să poată fi ușor implementate.

O metodă posibilă este să se dividă informația de control în câmpuri de control compatibile. Această metodă minimizează numărul biților de control, menținând gradul maxim de paralelism inerent setului original de microinstrucțiuni.

Notăm $\{I_j\}$ microinstrucțiunile microprogramului de multiplicare, unde j este adresa în CM a microinstrucțiunii I_j. După eliminarea biților redundanți din câmpul de control, au rămas de specificat 8 microoperații, acestea putând fi reprezentate de semnalele de control c_0, c_2, c_5, c_6, c_7, c_8, c_9 și *END*. În **Fig. 8.8** sunt listate cele 12 microinstrucțiuni și semnalele de control pe care le specifică.

Microinstrucțiune	Semnale de control
I_0	c_9, c_{10}
I_1	c_8
I_2	
I_3	c_2
I_4	c_0
I_5	
I_6	
I_7	c_2, c_5
I_8	c_6
I_9	c_7
I_{10}	*END*
I_{11}	

Fig. 8.8. Microinstrucțiuni pentru multiplicatorul în complement față de 2 și semnalele de control specificate.

Primul pas este determinarea *setului maxim de clase compatibile* (*Maximal Compatibility Classes* – MCCs), care este definit ca fiind clasele de compatibilitate la care nu se poate adăuga niciun semnal de control fără a introduce o pereche de semnale de control incompatibile. În situația dată, sunt doar două microoperații incompatibile, c_2 și c_5. Singurele *MCC*-uri sunt:

$C_0 = c_0\ c_2\ c_6\ c_7\ c_8\ c_9\ END$ și

$C_1 = c_0\ c_5\ c_6\ c_7\ c_8\ c_9\ END$, fiecare conținând 7 membri.

Întrucât C_0 și C_1 sunt și acoperirile minime *MCC*, sunt suficiente 2 câmpuri de control codificate.

Sunt 8 semnale de control de specificat, fiind inclusă și situația când nu se generează niciun semnal de control, deci sunt suficienți $[\log_2 8] = 3$ biți de control. Există mai multe modalități de alegere a subseturilor C_0 și C_1, care acoperă toate semnalele de control și dau o valoare minimă pentru cost. De exemplu, se alege $C_0' = c_0\ c_2\ c_6$ și $C_1' = c_5\ c_7\ c_8\ c_9\ END$. Microinstrucțiunea care rezultă are formatul din **Fig. 8.9** și necesită o pereche de decodificatoare pentru generarea semnalelor de control.

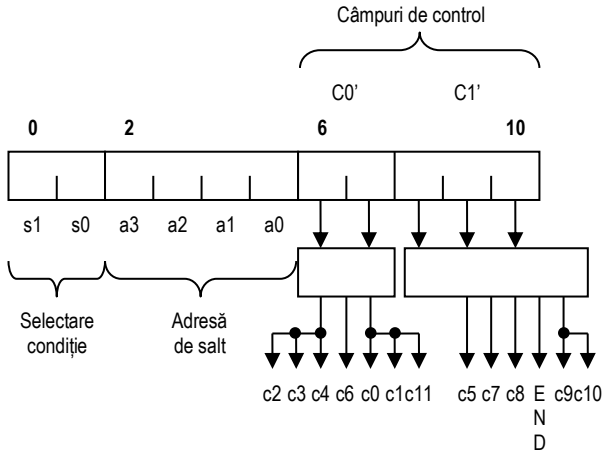


Fig. 8.9. Formatul microinstrucțiune codificat cu paralelism maxim și număr minim de biți de control.

Faptul că există numai două câmpuri de control, indică faptul că există un foarte mic paralelism în algoritmul de înmulțire.

Codificarea câmpurilor de control după funcție

Un dezavantaj al codificării din **Fig. 8.9** este faptul că se codifică în același câmp semnale de control fără legătură, în timp ce semnale codificate în câmpuri diferite sunt destinate unor funcții corelate. De exemplu, atât C_0' cât și C_1' controlează transferul spre magistrala OUTBUS. Toate acestea fac dificilă scrierea microprogramelor, întrucât microprogramatorul trebuie să asocieze coduri fără nicio legătură pentru fiecare câmp de control.

În general este preferat un format de codificare în care fiecare câmp de control specifică semnalele de control pentru o componentă sau pentru un set de operații asociate prin funcții pentru sistem. Examinând multiplicatorul, se observă că există 5 componente majore de controlat: sumatorul, combinația de registre *A* și *Q*, numărătorul extern pentru iterații *COUNT* și cele 2 magistrale de date INBUS și OUTBUS. Aceasta sugerează formatul de microinstrucțiune din **Fig. 8.10**.

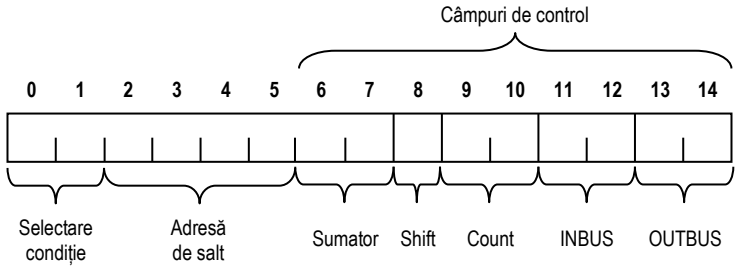


Fig. 8.10. Format microinstrucțiune cu câmpuri de control codificate prin funcție.

În **Fig. 8.11** se arată o alocare posibilă a biților câmpurilor de control și interpretarea lor. Această codificare *ad-hoc* a combinat semnalele c_2 și c_5, ceea ce ar putea fi util într-o dezvoltare ulterioară a setului de microinstrucțiuni.

Asigurarea unui câmp de control separat pentru INBUS este discutabilă. Aceasta împiedică transferul simultan al datelor de la INBUS la două sau mai multe destinații (*Q* și *M*, de exemplu). Acest transfer simultan ar putea fi util pentru resetarea/setarea lor simultană. Astfel, este mai util să se asocieze câte un câmp de control pentru fiecare registru care este un destinatar potențial al datelor de pe INBUS.

Câmp de control	Biți utilizați	Cod	Microoperații specifice	Semnale de control activate
Sumator	6, 7	00	Nicio operație	–
		01	$A \leftarrow A + M$	$c_2c_3c_4$
		10	$A \leftarrow A - M$	$c_2c_3c_4c_5$
		11	Nefolosit	–
Deplasare (Shift)	8	0	Nicio operație	–
		1	Deplasare dreapta *A*,*Q* și încarcă *A*(7)	c_0c_1
Numărător de iterații (Count)	9, 10	00	Nicio operație	–
		01	șterge *COUNT*, *A*, *F*	c_{10}
		10	*COUNT* ← *COUNT* + 1	c_{11}
		11	Nefolosit	–
INBUS	11, 12	00	Nicio operație	–
		01	*Q* ← INBUS	c_8
		10	*M* ← INBUS	c_9
		11	Nefolosit	–
OUTBUS	13, 14	00	Nicio operație	–
		01	OUTBUS ← *A*	c_6
		10	OUTBUS ← *Q*(7:1)	c_7
		11	Nefolosit	–

Fig. 8.11. Interpretarea câmpurilor de control ale microinstrucțiunii codificate prin funcție.

Formate de microinstrucțiuni multiple

În microprogramul din **Fig. 8.5** sunt incluse anumite microinstrucțiuni folosite numai pentru generarea adresei următoare, fără a genera niciun semnal de control. Aceasta sugerează reducerea mărimii unei microinstrucțiuni prin utilizarea unui singur câmp care să conțină fie informații de control, fie adresă de salt; de aici rezultă două tipuri distincte de microinstrucțiuni:

1. *microinstrucțiuni de salt,* care nu specifică nicio informație de control
2. *microinstrucțiuni de operare,* care activează liniile de control, dar nu au posibilități de a specifica saltul în microprogram.

Așa se întâmplă în general și la nivel de instrucțiune, unde instrucțiunile de salt controlează direct operațiile interne ale unității de control, în timp ce instrucțiunile de operare controlează direct unitatea externă de procesare a datelor.

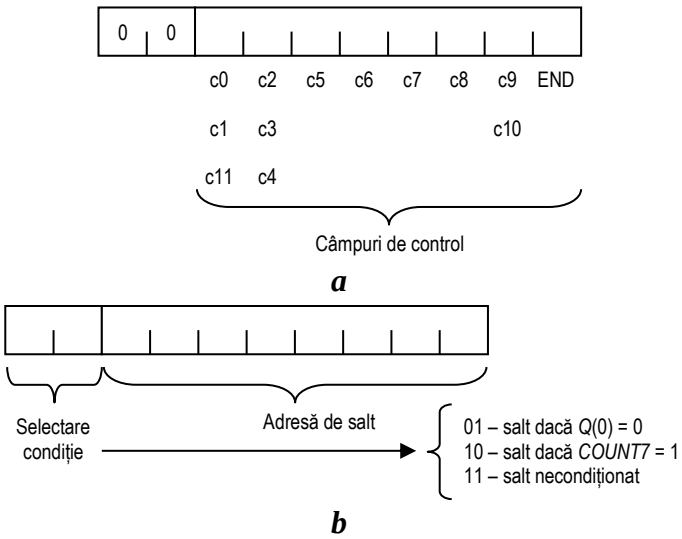


Fig. 8.12. Exemple de formate de microinstrucțiune multiple;
a – microinstrucțiune operațională, ***b*** – microinstrucțiune de salt.

Presupunem că dorim să codificăm câmpurile de control ale dispozitivului de înmulțire, astfel încât să asigurăm o flexibilitate maximă. Pentru aceasta sunt necesari 8 biți de control. Vom defini microinstrucțiunea formată din 2 câmpuri:

1. un câmp de selecție de 2 biți pentru condiția de salt;
2. un câmp de 8 biți care poate conține o *adresă de salt* sau *informație de control*.

Codificarea 00 pentru câmpul de selecție (care semnifică „fără salt") servește pentru identificarea microinstrucțiunilor de operare. Celelalte 3 codificări posibile identifică salturi condiționate sau necondiționate. Se obține astfel un format microinstrucțiune de 10 biți, ca în **Fig. 8.12**. Se observă că în aceste condiții se pot scrie programe cu $2^8 = 256$ instrucțiuni și sunt necesare mai multe microinstrucțiuni pentru a implementa anumite funcții.

Scrierea microprogramului este ușor de derivat din organigrama de funcționare: blocurile de decizie se transformă în microinstrucțiuni de salt, iar cele de operare în microinstrucțiuni de operare. Pentru utilizarea acestor noi formate de microinstrucțiuni, unitatea de control din **Fig. 8.6** se modifică în sensul că cei doi biți ai câmpului de selectare pot fi folosiți pentru a controla un demultiplexor care dirijează biții celuilalt câmp fie spre liniile de control externe (la microinstrucțiuni de operare), fie spre intrarea de încărcare a adresei de salt (la microinstrucțiunile de salt).

Mersul lucrării

1. Să se scrie microprogramul de înmulțire utilizând codificarea câmpurilor din **Fig. 8.9**.
2. Să se scrie microprogramul de înmulțire utilizând codificarea câmpurilor din **Fig. 8.10**.
3. Să se scrie microprogramul de înmulțire utilizând codificarea câmpurilor din **Fig. 8.12**.
4. Discuții și concluzii legate de variantele de microprograme obținute.

Lucrarea 9. Proiectarea unității de control microprogramate pentru unitatea centrală de prelucrare a unui calculator cu structură bazată pe acumulator / Lucrarea 10. Proiectarea unității de procesare a datelor pentru unitatea centrală de prelucrare a unui calculator microprogramat cu structură bazată pe acumulator

Scopul lucrărilor

Scopul acestor lucrări este proiectarea unei unități de comandă microprogramată pentru un CPU în variantă clasică, precum și pentru un CPU extins, care să poată realiza, pe lângă operațiile aritmetice de adunare și scădere, și operațiile de înmulțire și împărțire. Se va urmări proiectarea în detaliu, atât a părții de procesare a datelor, cât și a părții de comandă.

Considerații teoretice

Se ia în discuție organizarea ipotetică a CPU-ului analizat în Lucrarea nr. 6, a cărui structură este prezentată în **Fig. 9.1**.

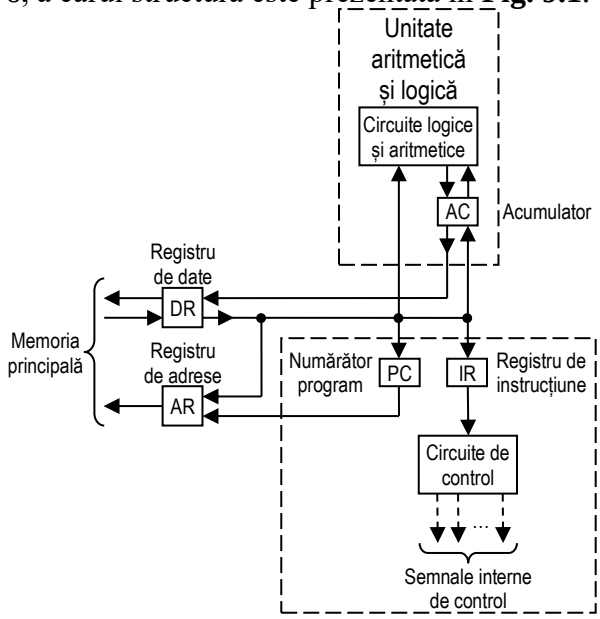


Fig. 9.1. Structura unui CPU simplu bazat pe acumulator.

Presupunem că cele 13 semnale de control din **Fig. 9.2** definesc microoperațiile disponibile microprogramatorului.

Semnal de control	Operație controlată
c_0	AC ← AC + DR
c_1	AC ← AC ∧ DR
c_2	AC ← $\overline{AC}$
c_3	DR ← M(AR) (*READ* M)
c_4	M(AR) ← DR (*WRITE* M)
c_5	DR ← AC
c_6	AC ← DR
c_7	AR ← DR(ADR)
c_8	PC ← DR(ADR)
c_9	PC ← PC + 1
c_{10}	AR ← PC
c_{11}	IR ← DR(OP)
c_{12}	RIGHT-SHIFT AC

Fig. 9.2. Semnalele de control pentru CPU.

Pentru simplitatea prezentării, se vor reprezenta microinstrucțiunile în formă simbolică, utilizând limbajul de descriere prezentat în Lucrarea nr. 2.

Se va scrie un emulator pentru cele 8 instrucțiuni prezentate în Lucrarea nr. 6: LOAD, STORE, ADD, AND, JUMP, JUMPZ, NOT și RSHIFT. Microoperațiile necesare pentru realizarea acestora sunt prezentate în **Fig. 9.3**. Pe baza acestei organigrame se pot scrie fără dificultate microprogramele necesare pentru execuția instrucțiunilor.

Codul instrucțiunii determină microprogramul selectat de fiecare instrucțiune, deci conținutul registrului *IR* poate fi utilizat pentru determinarea adresei de start a microprogramului. Fiecare microinstrucțiune poate specifica o condiție de salt, o adresă de salt, utilizată numai dacă este îndeplinită condiția de salt, și un set de câmpuri de control, care definesc microoperațiile care se execută.

În **Fig. 9.4** este prezentat, în formă simbolică, un emulator complet pentru setul dat de instrucțiuni; conversia fiecărei microoperații în formă binară este simplă. Emulatorul conține un microprogram distinct pentru fiecare din cele 8 cicluri de execuție și un microprogram, numit FETCH, care implementează ciclul aducere al instrucțiunii.

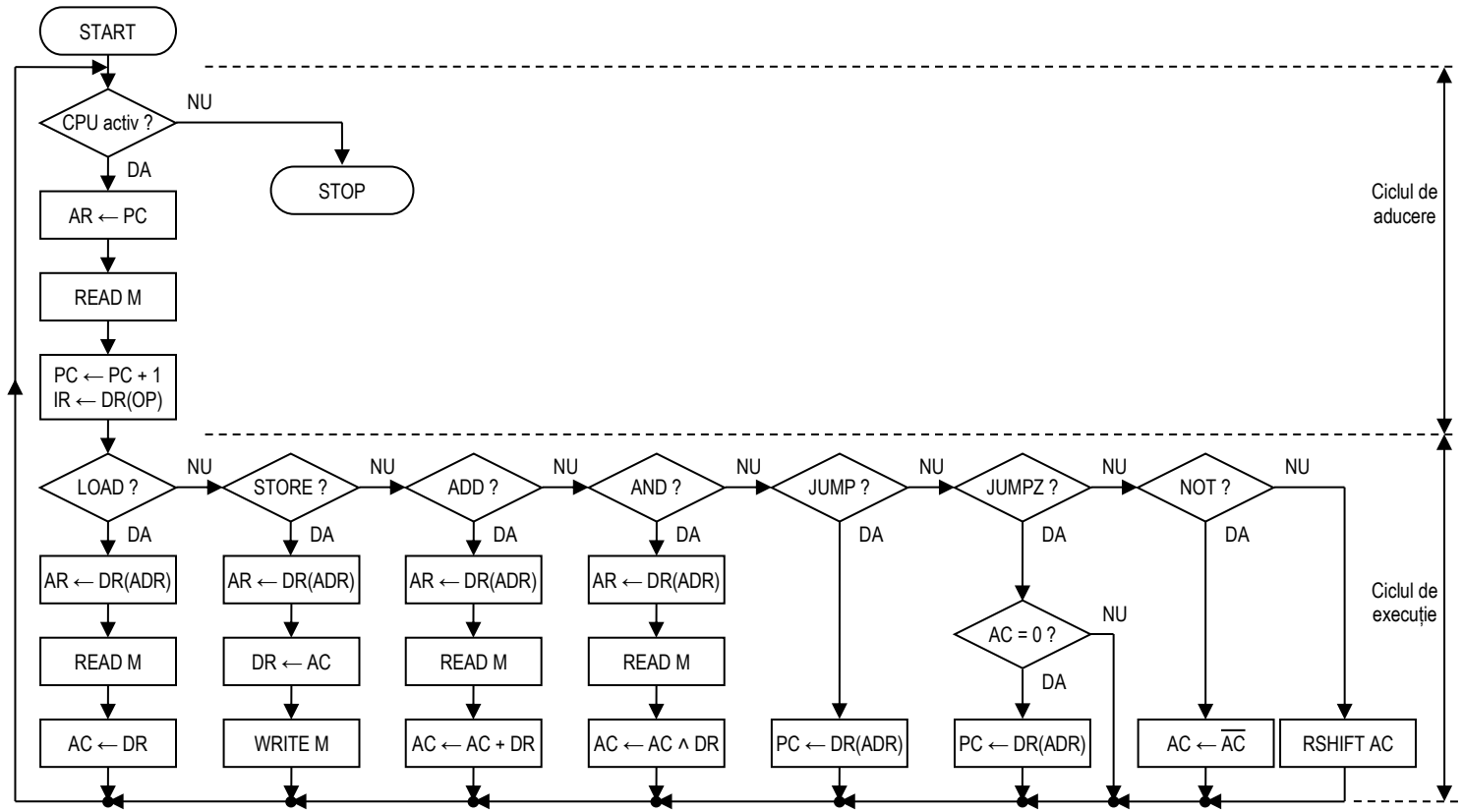


Fig. 9.3. Operarea CPU-ului cu 8 instrucțiuni.

Declarația *go to IR* poate fi implementată sub forma μPC ← IR, care transferă controlul primei microinstrucțiuni a microprogramului care interpretează instrucțiunea curentă. Totuși, se recomandă scrierea unui microprogram numit „SEL”, „INSTR” etc. care să specifice toate condițiile, codurile și adresele de salt, deoarece codurile de instrucțiune din IR s-ar putea să nu corespundă cu cele din microprogram.

```
FETCH:    AR ← PC;
          READ M;
          PC ← PC+1, IR ← DR(OP);
          go to IR;

LOAD:     AR ← DR(ADR);
          READ M;
          AC ← DR, go to FETCH;

STORE:    AR ← DR(ADR);
          DR ← AC;
          WRITE M, go to FETCH;

ADD:      AR ← DR(ADR);
          READ M;
          AC ← AC+DR, go to FETCH;

AND:      AR ← DR(ADR);
          READ M;
          AC ← AC ∧ DR, go to FETCH;

JUMP:     PC ← DR(ADR), go to FETCH;

JUMPZ:    if AC ≠ 0 then go to FETCH;
          PC ← DR(ADR) , go to FETCH;

NOT:      AC ← AC̄, go to FETCH;

RSHIFT:   RIGHT-SHIFT(AC), go to FETCH;
```

Fig. 9.4. Emulator microprogramat.

Presupunem că datorită unei erori de proiectare a fost uitată implementarea unei instrucțiuni, numită CLEAR, a cărei funcție este să reseteze la 0L toți biții acumulatorului. Cu toate că nu a fost prevăzută nicio linie de control pentru ștergerea lui AC, poate fi scris un microprogram care să implementeze instrucțiunea CLEAR:

CLEAR: DR ← AC;
AC ← $\overline{AC}$;
AC ← AC ∧ DR, go to FETCH;

Astfel, prin adăugarea la CM a acestui microprogram, se poate adăuga CLEAR la setul de instrucțiuni fără a fi necesare modificări hardware. Această flexibilitate este principalul avantaj al microprogramării.

Un CPU extins

La structura de bază a CPU-ului din **Fig. 9.1** au fost adăugate circuitele necesare pentru implementarea înmulțirii în virgulă fixă și a împărțirii cu refacerea restului.

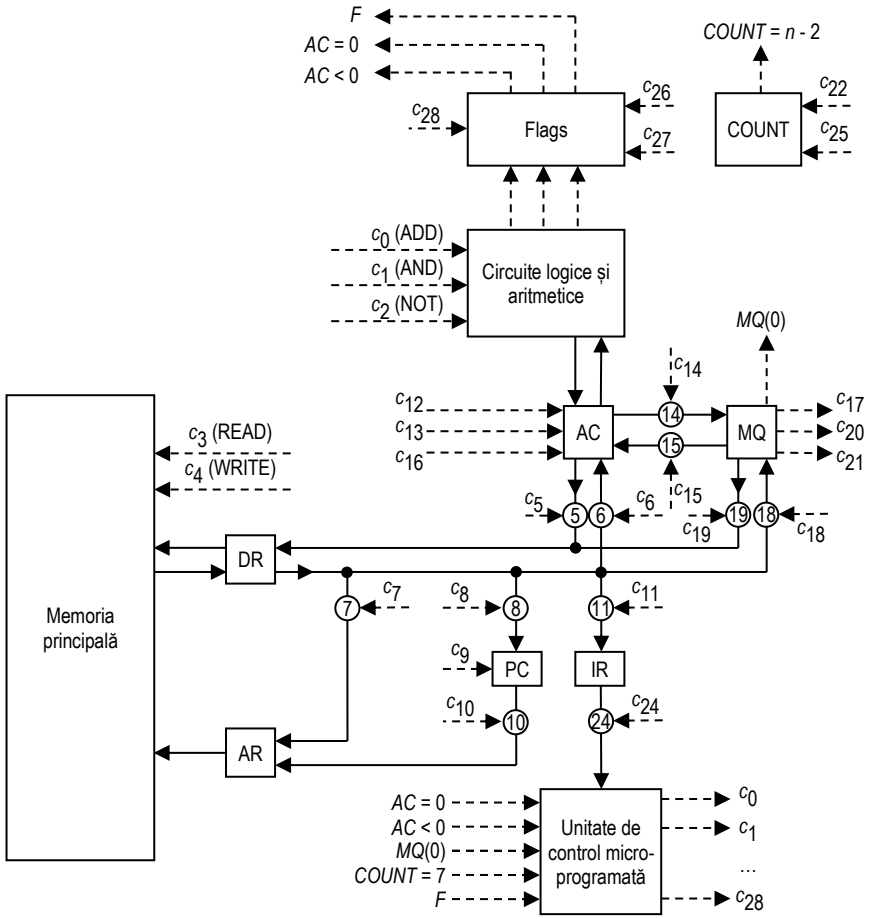


Fig. 9.5. Structura unui CPU extins.

Pentru aceasta sunt necesare două registre noi: un registru *MQ* pentru memorarea înmulțitorului și a câtului și un registru numărător *COUNT* pentru numărarea deplasărilor efectuate la înmulțire sau împărțire. Funcția de memorare a deînmulțitului și a divizorului se va aloca registrului de date. În **Fig. 9.5** este prezentată structura CPU-ului extins, unde numărul liniilor de control este aproximativ dublat. Setul complet de linii de control, definit pentru această unitate, este dat în **Fig. 9.6**, unde liniile $c_0 - c_{12}$ sunt liniile de control definite în circuitul original.

Semnal de control	Operație controlată
c_0	AC ← AC + DR
c_1	AC ← AC ∧ DR
c_2	AC ← $\overline{AC}$
c_3	DR ← M(AR)
c_4	M(AR) ← DR
c_5	DR ← AC
c_6	AC ← DR
c_7	AR ← DR(ADR)
c_8	PC ← DR(ADR)
c_9	PC ← PC+1
c_{10}	AR ← PC
c_{11}	IR ← DR(OP)
c_{12}	RIGHT-SHIFT AC
c_{13}	LEFT-SHIFT AC
c_{14}	RIGHT-SHIFT (AC, MQ)
c_{15}	LEFT-SHIFT (AC, MQ)
c_{16}	AC ← 0
c_{17}	AC(7) ← F
c_{18}	MQ ← DR
c_{19}	DR ← MQ
c_{20}	MQ(0) ← 1
c_{21}	MQ(0) ← 0
c_{22}	COUNT ← COUNT + 1
c_{23}	AC ← AC – DR
c_{24}	μPC ← IR
c_{25}	COUNT ← 0
c_{26}	F ← 0
c_{27}	F ← 1
c_{28}	FLAGS ← 0

Fig. 9.6. Semnale de control pentru CPU-ul extins.

Mai jos este dat microprogramul care implementează înmulțirea în complement față de 2:

```
MULT:   AC ← 0, COUNT ← 0, F ← 0;
TEST1:  if  MQ(0) = 0  then go to SHIFT;
        AC ← AC + DR, F ← F ∨ [Q(0) ∧ M(7)];
SHIFT:  AC,MQ←RIGHT-SHIFT(AC,MQ), AC(7)←F, COUNT
        ← COUNT + 1;
        if  COUNT = 7 then go to TEST2;
        go to TEST1;
TEST2:  if  MQ(0) = 0 then go to FETCH;
        AC ← AC – DR, go to FETCH;
```

Se presupune că înainte de a fi executat microprogramul MULT, înmulțitorul este în MQ și deînmulțitul în DR. Fiecare declarație corespunde unei singure microinstrucțiuni.

Unitatea de control

Se utilizează formatul de microinstrucțiune general, care cuprinde un câmp de selectare a condiției, un câmp al adresei de salt și un set de câmpuri de control. Pe baza **Fig. 9.5** se deduce că trebuie testate 5 condiții: *AC* = 0, *AC* < 0, *MQ*(0), *COUNT* = 7 și *F* (indicator de sub- și supra-depășire).

Pentru a asigura testarea acestor 5 condiții, precum și pentru indicarea saltului necondiționat sau a situației când nu se face salt, se obțin 7 coduri posibile care pot fi reprezentate printr-un câmp de selectare a condiției de 3 biți.

Pentru reducerea dimensiunii unei microinstrucțiuni, pe baza **Fig. 9.6** diferite semnale de control pot fi grupate în câmpuri de codificare comune. De exemplu, semnalele c_3, c_5 și c_{19}, care transferă date spre DR, întrucât sunt exclusive, ele pot fi grupate într-un câmp de 2 biți, la fel și semnalele care acționează asupra lui AC.

Menționăm faptul că trebuie rezervat un *pattern* de un bit pentru situația de ne-operare.

În situația în care semnalele de control nu se codifică (aceasta implică ocuparea de către condiția de selectare și de către câmpurile de control a câte 28 biți pentru fiecare microinstrucțiune) și că se asigură adrese de salt de 8 biți, care pot adresa întreg CM, rezultă necesitatea utilizării unui CM de capacitate 256 cuvinte de 36 de biți.

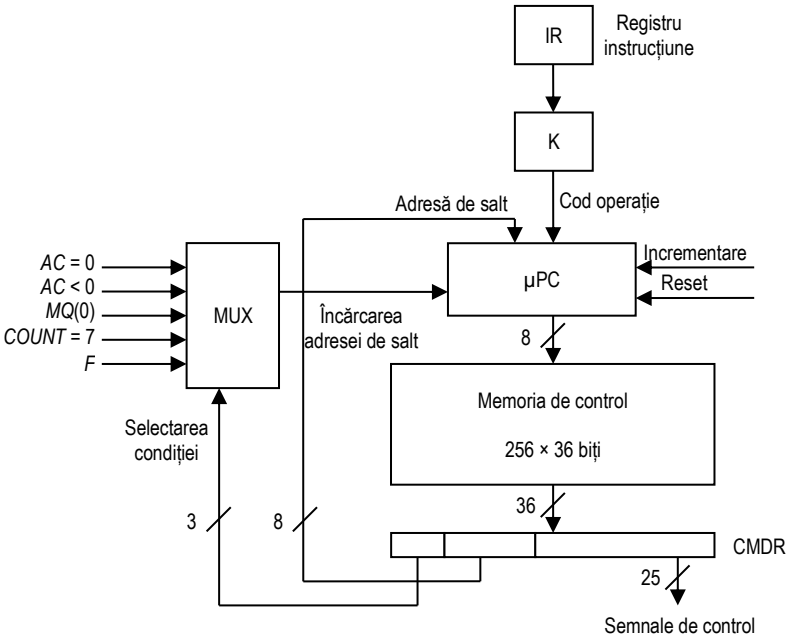


Fig. 9.7. Unitatea de control microprogramată a CPU-ului.

În aceste condiții, în **Fig. 9.7** este arătată o organizare posibilă pentru unitatea de control a CPU-ului extins. Se observă că sunt utilizate condițiile externe pentru încărcarea adreselor de salt în μPC, și că IR asigură încărcarea lui μPC printr-un circuit logic K (care poate fi un ROM), care mapează codul instrucțiunilor în adresă de microinstrucțiune.

Mersul lucrărilor:

Pentru Lucrarea nr. 9:

1. Să se scrie un microprogram pentru unitatea de control a unui CPU în variantă extinsă cu format de microinstrucțiune în care semnalele de control nu sunt codificate.
2. Să se scrie un microprogram pentru unitatea de control a unui CPU în variantă extinsă folosind microinstrucțiuni în care semnalele de control sunt codificate.

Pentru Lucrarea nr. 10:

Să se facă proiectarea unității de procesare a datelor pentru CPU-ul extins.

Bibliografie

1. https://americanhistory.si.edu/collections/search/object/nmah_334741 (accesat 04.05.2021)
2. http://bitsavers.informatik.uni-stuttgart.de/pdf/rand/P-377_The_History_And_Development_Of_The_IAS_Computer_Mar53.pdf (accesat 04.05.2021)
3. http://www.csit-sun.pub.ro/courses/cn1CA/CN1_7-Operatii%20Aritmetice.pdf (accesat 06.05.2021)
4. http://ac.upg-ploiesti.ro/cursuri/calc_num/curs.pdf (accesat 06.05.2021)
5. https://circuitverse.org/ (accesat 30.03.2020)
6. David A. Patterson, John L. Hennessy, *Organizarea și proiectarea calculatoarelor*, Editura All, București, 2002.
7. David A. Patterson, John L. Hennessy, *Computer Architecture and Organization*, McGraw-Hill, John P. Hayes, University of Southern California,1990.
8. Vasile Pop, *Structura sistemelor de prelucrare a datelor numerice*, curs, vol. I și II, Institutul Politehnic „Traian Vuia", Timișoara, 1981.
9. Vasile Pop, *Structura sistemelor de prelucrare a datelor numerice*, îndrumător laborator, Institutul Politehnic „Traian Vuia", Timișoara, 1981.
10. Hill W., Peterson G.R., *Digital Systems: Hardware Organization*, John Wilez & Sons, New-York, 1970.
11. Petrescu A., *Microprogramare. Principii și aplicații*, Editura Tehnică, București, 1975.
12. Petrescu A., *Calculatoare automate și programare*, Editura Tehnică și Pedagogică, București, 1973.
13. Rogojan A., *Calculatoare numerice*, curs, vol. I și II, Institutul Politehnic „Traian Vuia" Timișoara, 1973.
14. Strugaru C., *Echipamente periferice și transmiterea datelor*, curs, Institutul Politehnic „Traian Vuia", Timișoara, 1979.
15. Husson S.S., *Microprogramming. Principles and Practice*, Prentice-Hall Inc., New-Jersey,1970.
16. Dodescu G. ș.a., *Calculatoare electronice și sisteme de operare*, Editura Didactică și Pedagogică, București, 1975.
17. Jurcă I., *Simularea sistemelor continue și discrete*, curs, Institutul Politehnic „Traian Vuia", Timișoara, 1980.
18. Cristea V. ș.a., *Calculatoare personale. Rețele de calculatoare*, Editura Teora, București, 1992.
19. www.altera.com (accesat 25.08.2015)

Anexa 1. Unitate de control cablată pentru multiplicatorul a două numere binare cu virgulă fixă în cod complement față de 2

O posibilitate de executare a Lucrării nr. 3 constă în divizarea în două secțiuni a unității de control cablate care se proiectează prin metoda tabelului de stare:

- ✓ o secțiune care gestionează numai variabilele de stare (circuit secvențial);
- ✓ o secțiune care generează semnalele de control conform valorilor variabilelor de stare (circuit combinațional).

Conform **Fig. a1.1** (**Fig. 3.8**) acest circuit secvențial este compus dintr-un circuit combinațional CLC_1 care primește semnalele externe *START*, $Q(0)$ și *COUNT7* alături de variabilele de stare $x1$, $x2$, $x3$, oferind semnalele de comandă pentru cele trei bistabile JK_1, JK_2 și JK_3 care gestionează cele trei variabile de stare. Astfel, proiectarea acestui dispozitiv se referă mai ales la sinteza circuitului combinațional CLC_1.

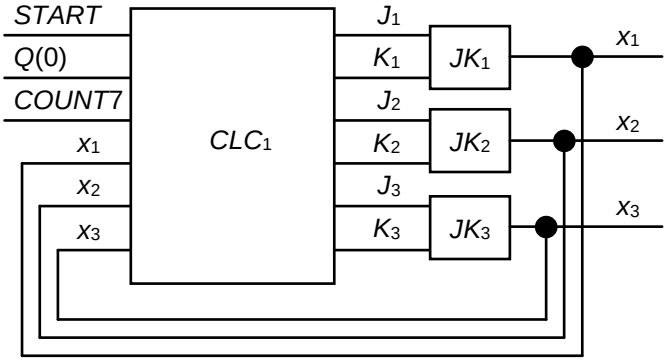


Fig. a1.1. Dispozitivul electronic secvențial care gestionează variabilele de stare.

În acest scop se construiește tabelul de adevăr al acestui circuit combinațional, ținând cont de tabelul de stare din **Fig. 3.4**. Ținând cont de combinațiile posibile dintre stările sistemului și informațiile de intrare, precum și de tranzițiile posibile între aceste stări, tabelul de adevăr pentru acest circuit combinațional are 22 de rânduri, completarea sa fiind destul de ușoară (**Fig. a1.2**)

START	Q(0)	CNT7	x1	x2	x3	Jx1	Kx1	Jx2	Kx2	Jx3	Kx3
0	0	0	0	0	0	0	X	0	X	0	X
0	0	0	0	0	1	0	X	1	X	X	1
0	0	0	0	1	0	1	X	X	1	0	X
0	0	0	1	0	0	X	0	0	X	0	X
0	0	1	0	0	0	0	X	0	X	0	X
0	0	1	1	0	0	X	0	1	X	0	X
0	0	1	1	1	0	X	0	X	0	1	X
0	0	1	1	1	1	X	1	X	1	X	1
0	1	0	0	0	0	0	X	0	X	0	X
0	1	0	0	0	1	0	X	1	X	X	1
0	1	0	0	1	0	0	X	X	0	1	X
0	1	0	0	1	1	1	X	X	1	X	1
0	1	0	1	0	0	X	1	1	X	1	X
0	1	1	0	0	0	0	X	0	X	0	X
0	1	1	1	0	0	X	0	0	X	1	X
0	1	1	1	0	1	X	0	1	X	X	1
0	1	1	1	1	0	X	0	X	0	1	X
0	1	1	1	1	1	X	1	X	1	X	1
1	0	0	0	0	0	0	X	0	X	1	X
1	0	1	0	0	0	0	X	0	X	1	X
1	1	0	0	0	0	0	X	0	X	1	X
1	1	1	0	0	0	0	X	0	X	1	X

Fig. a1.2. Tabelul de adevăr pentru CLC_1.

Realizând o analiză combinațională a acestui tabel de adevăr se poate obține circuitul combinațional CLC_1 care corespunde acestor specificații. Conectând la bornele de ieșire bistabilele JK respective și readucând la intrările circuitului ieșirile corespunzătoare ale acestor bistabile, se obține circuitul secvențial care gestionează variabilele de stare și tranzițiile sistemului de calcul. Acest dispozitiv secvențial, realizat cu aplicația online CircuitVerse [5], este prezentat în **Fig. a1.3**.

Urmează realizarea circuitului combinațional care generează semnalele de control conform stărilor sistemului de calcul, ilustrat în **Fig. a1.4** (**Fig. 3.9**). Ținând cont de valorile variabilelor de stare, acest circuit este foarte ușor de proiectat. În **Fig. a1.5** este prezentat acest circuit, realizat cu aceeași aplicație online CircuitVerse [5].

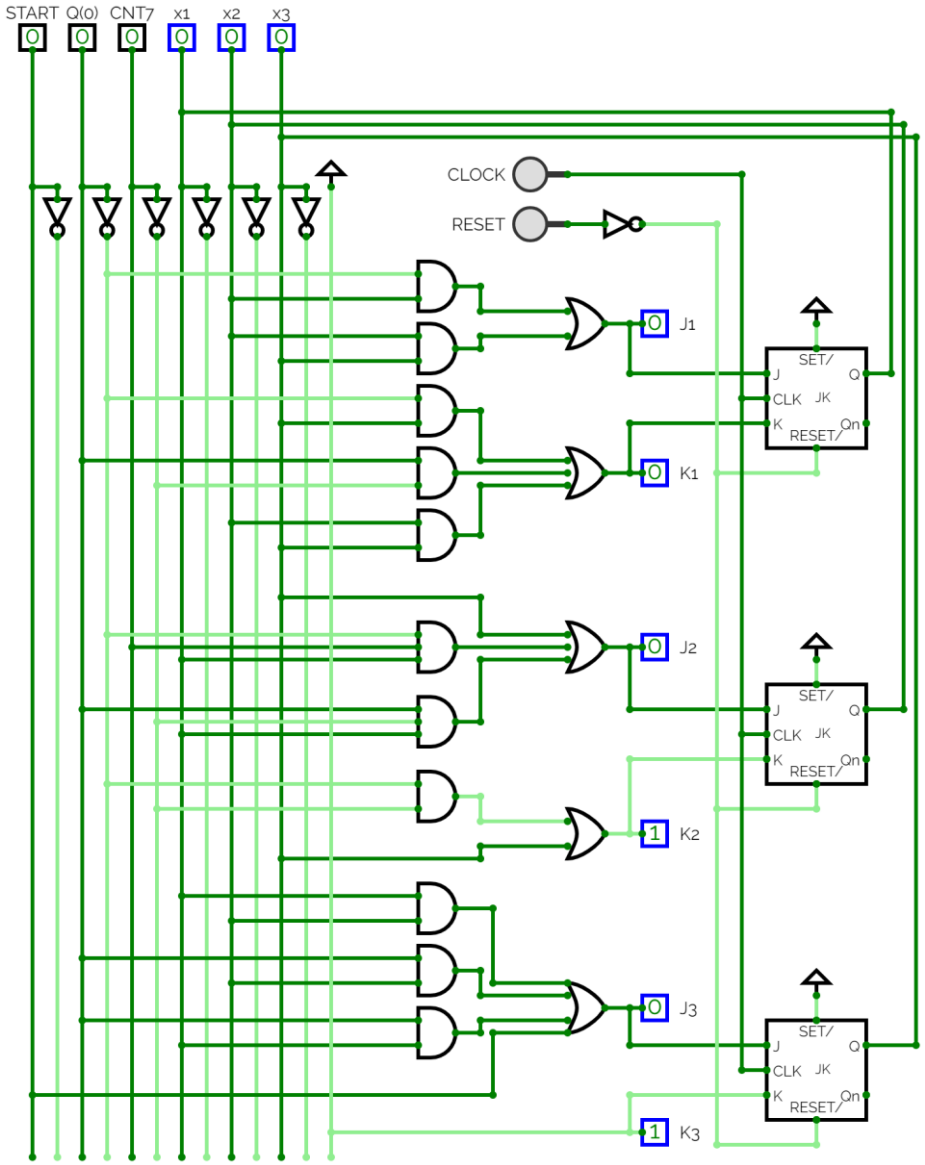


Fig. a1.3. Dispozitivul electronic secvențial care gestionează variabilele de stare, realizat cu CircuitVerse.

Interconectându-se aceste două dispozitive digitale rezultă unitatea de control cablată obținută prin metoda tabelului de stare pentru multiplicatorul a două numere binare în virgulă fixă cu semn.

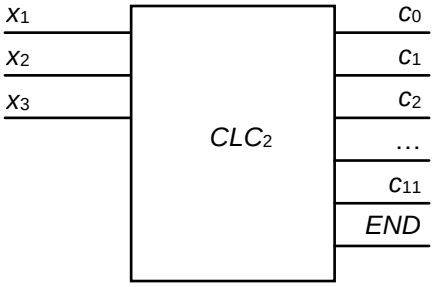


Fig. a1.4. Dispozitivul electronic combinațional care generează semnalele de control în funcție de variabilele de stare.

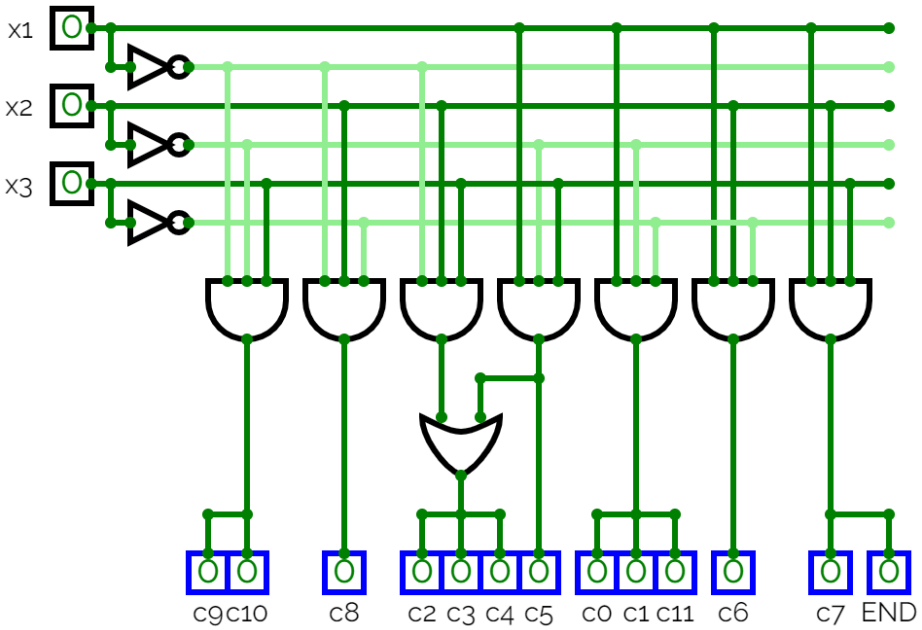


Fig. a1.5. Circuitul combinațional care generează semnalele de control, realizat cu CircuitVerse [5].

Anexa 2. Aplicația Quartus și utilizările acesteia în studiul structurii și organizării calculatoarelor

Cu ajutorul ariilor de porți logice programabile (*FPGA – **F**ield **P**rogrammable **G**ates **A**rray*) pot fi construite, studiate și testate cele mai diverse dispozitive electronice, prin intermediul unor aplicații software dedicate. Datorită flexibilității, FPGA-urile constituie niște ajutoare foarte prețioase în cercetare, permițând îmbunătățirea continuă a soluțiilor alese pe baza testărilor și simulărilor efectuate asupra structurii programate. În urma prelucrării și studiului rezultatelor obținute, structura poate fi reprogramată foarte ușor prin intermediul acelorași aplicații software cu care s-a efectuat programarea. Astfel, pornind de la o idee, se poate ajunge la o structură complexă care, după terminarea cercetărilor și îmbunătățirilor succesive, poate fi construită sub forma unui dispozitiv de-sine-stătător care execută cu maximă operativitate funcția pentru care a fost elaborat.

Compania Altera https://www.altera.com/ [14] construiește atât FPGA-uri de înaltă calitate, cât și aplicațiile software adecvate pentru programarea și utilizarea acestora. Câteva dintre clasele de FPGA-uri Altera, dotate cu memorie inclusă, blocuri de prelucrare a semnalelor digitale (*DSP – **D**igital **S**ignal **P**rocessing*), pini I/O de mare viteză etc., sunt Stratix®, Arria®, Cyclone®, MAX®, fiecare având caracteristicile sale specifice. De exemplu, seria Cyclone® este adecvată pentru aplicații de putere redusă și pentru proiectare cu costuri mici, permițând obținerea de soluții rapide și flexibile pentru crearea prototipurilor și a produselor finite.

FPGA-urile Cyclone® sunt incluse pe plăcile Altera DE0, Altera DE1, Altera DE2, pe kit-urile de dezvoltare Cyclone II Starter Development Kit, Cyclone II „Niomite” EP2C8 Module, Cyclone II „Niomite” EP2C8 USB Blaster Kit, Cyclone II „Quicgate” EP2C8 USB Blaster Kit etc. În cadrul lucrărilor de laborator la disciplina *Arhitectura sistemelor de calcul* vom utiliza plăcile Altera DE1, familia EP2C20F484C7 (**Fig. a2.1**)

Pentru programarea plăcilor Altera DE1 vom utiliza aplicația Quartus II Web Edition, elaborată de aceeași companie Altera. Prin intermediul acestei aplicații, FPGA-ul de pe placa Altera DE1 poate fi configurat pentru a obține atât structuri simple (porți logice NOT, AND, NAND, OR, NOR etc., multiplexoare, decodificatoare zecimale sau hexazecimale pentru afișoare cu 7 segmente), cât și structuri complexe

(microprocesoare NIOS II, porturi I/O, module JTAG UART). Aplicația oferă posibilitatea de simulare a funcționării structurii, prin intermediul aplicării la intrări a unor semnale digitale, configurabile de către utilizator, iar toate semnalele de intrare și de ieșire sunt afișate pe ecranul computerului-gazdă, unde pot fi studiate și interpretate. Prin stabilirea corespondențelor dintre pinii virtuali din structură cu pinii reali ai FPGA-ului, intrările și ieșirile structurii programate pot fi atribuite butoanelor, comutatoarelor, LED-urilor, afișajelor, porturilor audio/video etc., astfel încât utilizatorul să poată testa în realitate funcționarea acesteia.

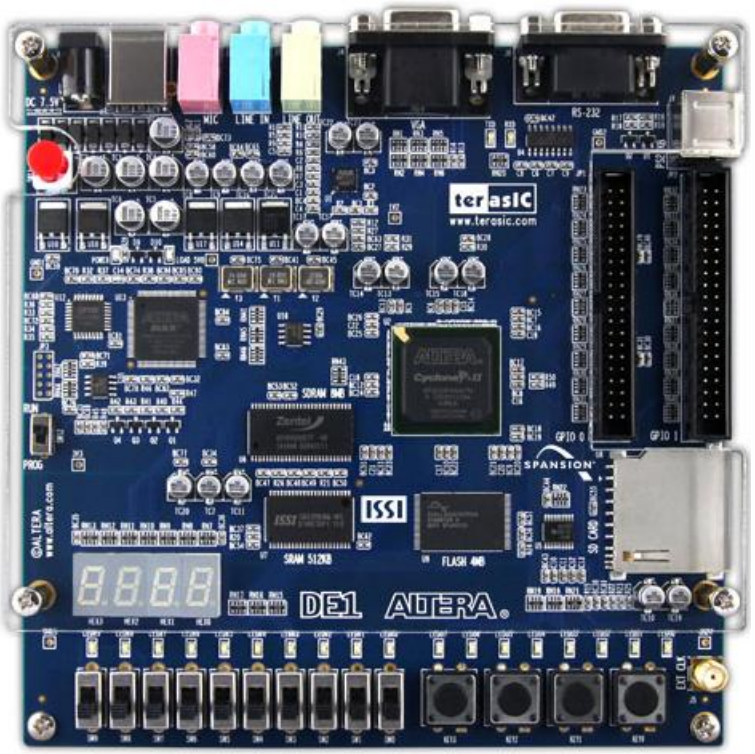


Fig. a2.1. Placa Altera DE1.

Aplicația Quartus II Web Edition permite crearea de proiecte pentru fiecare lucrare de laborator, care trebuie salvate în foldere (directoare) diferite. După pornirea aplicației (**Fig. a2.2**) se trece la crearea noului proiect (**Fig. a2.3**). În cazul în care aplicația Quartus propune un folder de lucru pe discul C, fie și pe *desktop*, este recomandabil să se creeze folderele de lucru pe discul D, deoarece acest disc este destinat documentelor și fișierelor create de către utilizator.

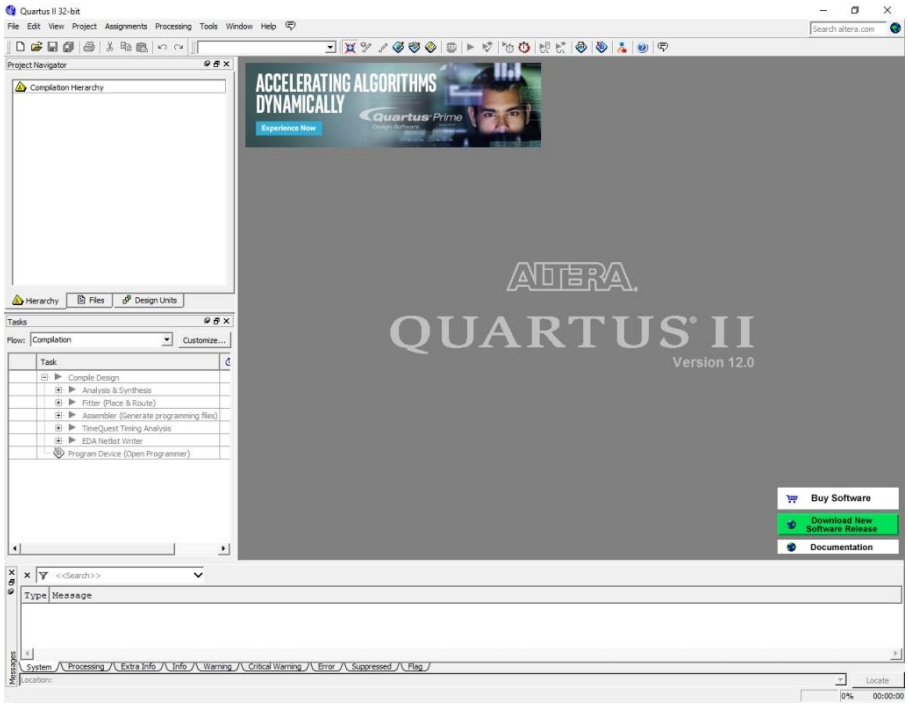


Fig. a2.2. Aplicația Quartus II Web Edition.

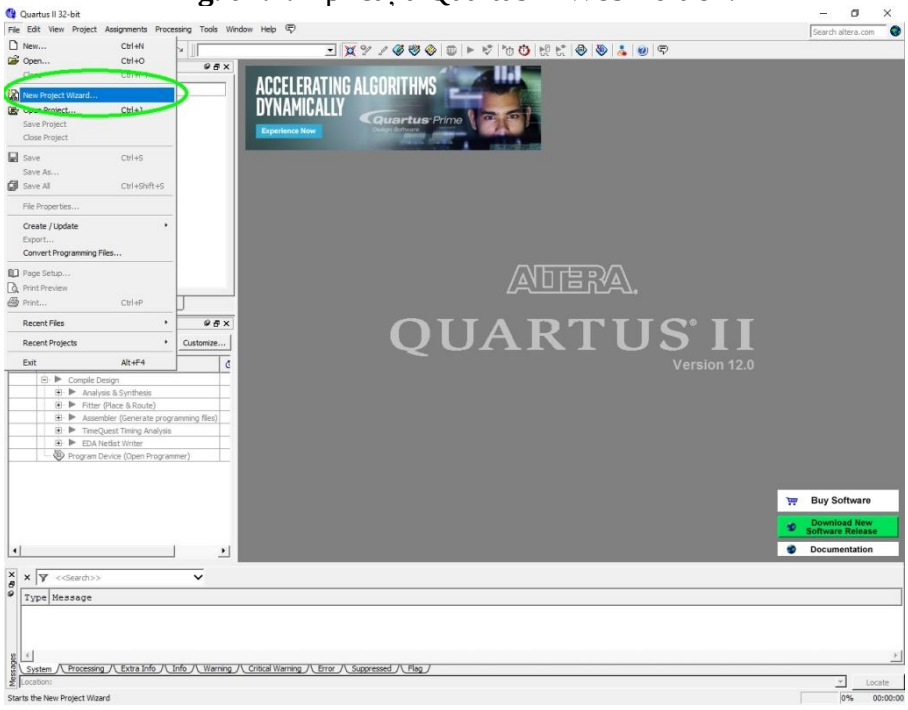


Fig. a2.3. Crearea noului proiect.

Pentru a evita erorile, este bine ca numele folderului să coincidă cu numele proiectului, folosindu-se numai caractere alfanumerice, fără diacritice, spații, semne aritmetice etc. (**Fig. a2.4**).

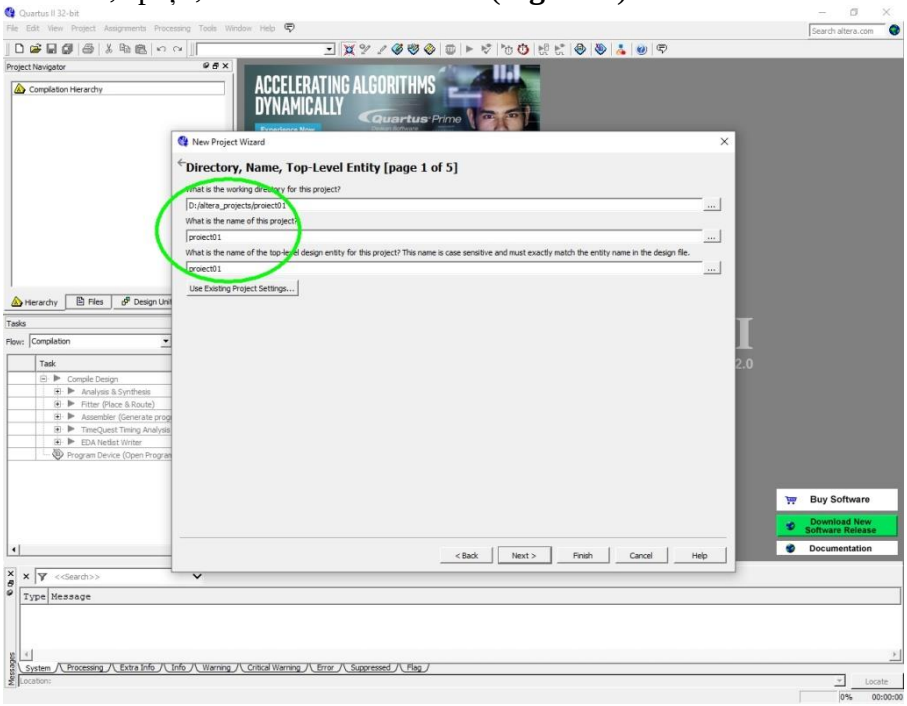


Fig. a2.4. Alegerea folderului de lucru și a numelui proiectului.

Urmează încă patru pagini (dintr-un total de cinci), reprezentând etapele elaborării noului proiect. În cazul în care nu avem de modificat setările din aceste pagini, vom executa *click* pe butonul „Next”.

Pagina a doua oferă posibilitatea de a se adăuga fișiere suplimentare în proiectul nostru, care pot fi selectate și adăugate individual, în ordinea dorită.

Pagina a treia este foarte importantă, deoarece aici selectăm dispozitivul cu care vom lucra. Deoarece avem la dispoziție plăcile Altera DE1, dotate cu FPGA-uri din familia Cyclone II, cu numele circuitului EP2C20F484C7, vom selecta aceste caracteristici din meniurile *drop-down* și listele de dispozitive (**Fig. a2.5** și **Fig. a2.6**).

În pagina a patra se pot selecta unelte EDA suplimentare față de cele incluse deja în mediul de programare Quartus II. Astfel, se pot selecta unelte pentru introducerea schemelor și sinteză, pentru simulare sau pentru analiza temporară. Deoarece noi vom utiliza uneltele incluse în mediul de programare Quartus II, nu vom face nicio setare.

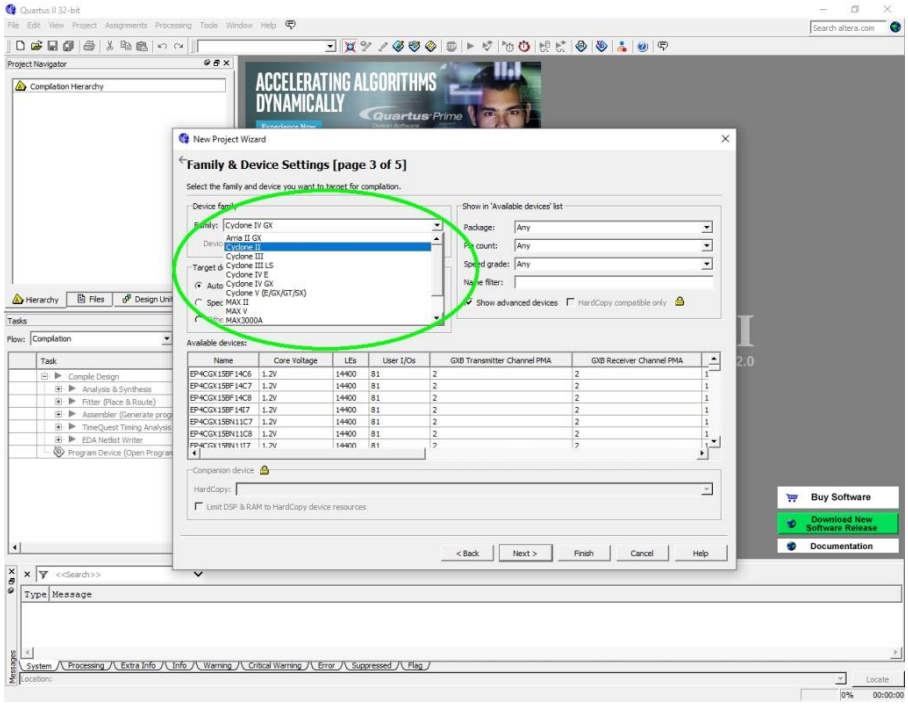


Fig. a2.5. Alegerea familiei de dispozitive.

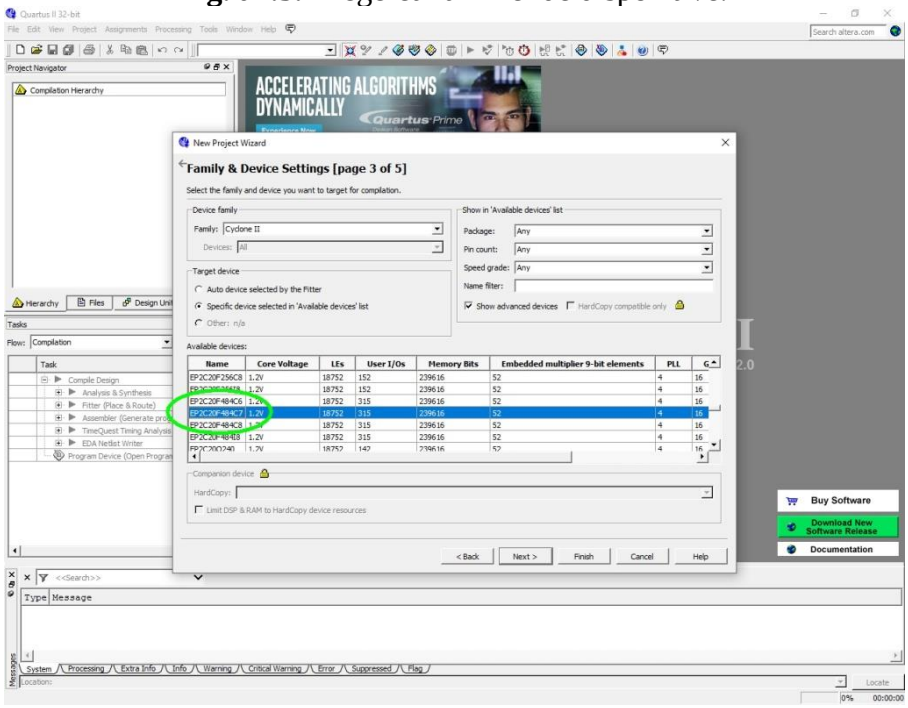


Fig. a2.6. Alegerea dispozitivului.

Pagina a cincea prezintă un rezumat al noului proiect, în care putem verifica numele proiectului, folderul de lucru și dispozitivul ales (**Fig. a2.7**). În cazul în care observăm greșeli, le putem corecta revenind la paginile anterioare executând *click* pe butonul „Back”. Dacă totul este în ordine, atunci încheiem etapa de creare a proiectului executând *click* pe butonul „Finish”.

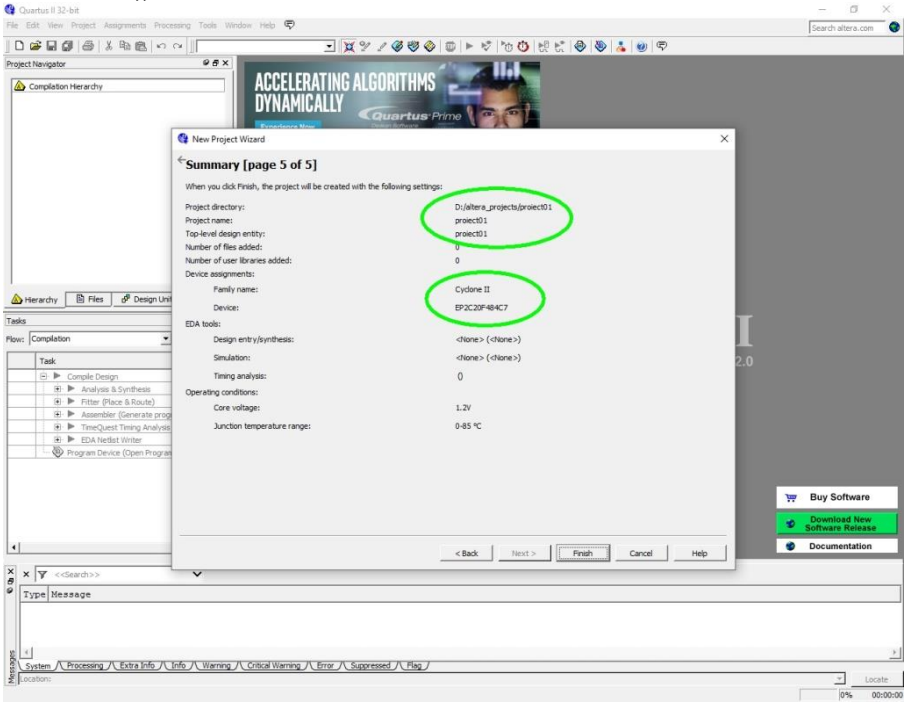


Fig. a2.7. Rezumatul proiectului.

Din acest moment, bara de titlu a paginii Quartus II afișează numele proiectului și calea către acesta, iar fereastra „Project Navigator” conține numele dispozitivului ales și denumirea entității pe care o vom construi (**Fig. a2.8**). Dacă totuși s-a greșit la alegerea dispozitivului de lucru, atunci se mai poate reveni prin executarea unui *click* pe meniul „Assignments” și selectarea opțiunii „Device”, iar după alegerea corectă se execută *click* pe butonul „OK”.

Noul proiect poate fi creat atât prin elaborarea acestuia în limbaj VHDL, cât și prin construcția sa din elemente gata conținute în aplicația Quartus II. În al doilea caz pot fi utilizate uneltele „SOPC Builder” sau „Qsys” din meniul „Tools” (**Fig. a2.9**). Unealta „SOPC Builder” este mai ușor de utilizat decât „Qsys”, însă dispare din versiunile mai recente ale aplicației Quartus II.

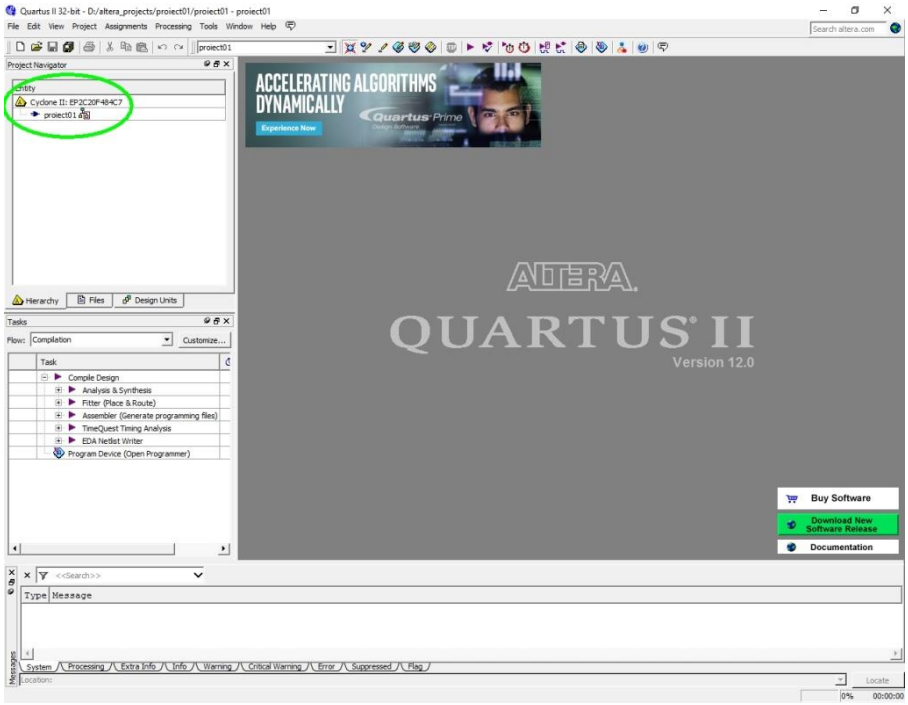


Fig. a2.8. Proiectul nou.

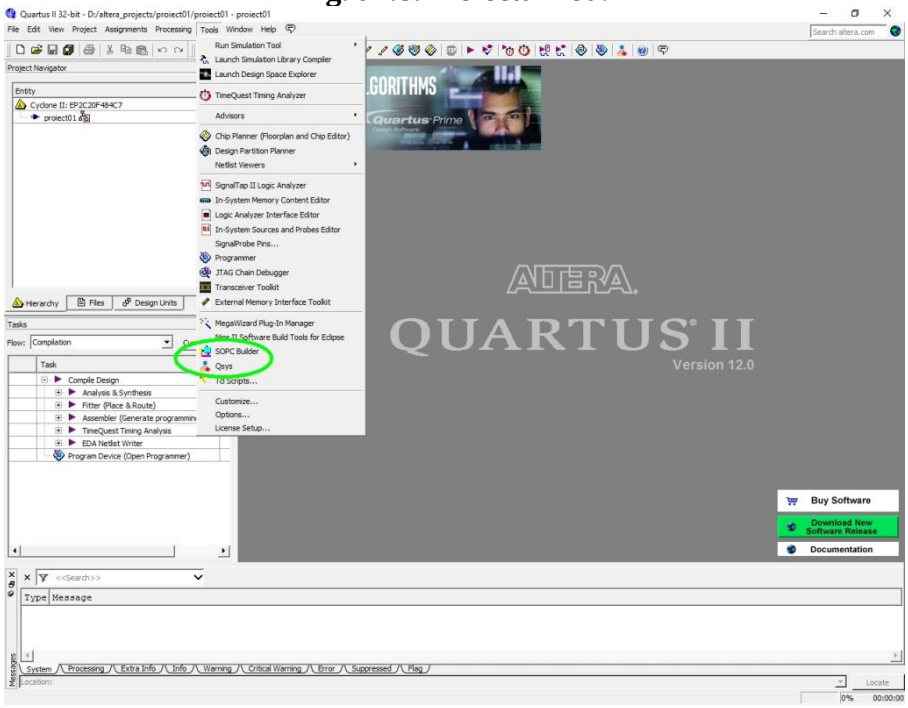


Fig. a2.9. Alegerea uneltelor „SOPC Builder” sau „Qsys”.

Pentru exemplificare, vom folosi unealta „SOPC Builder”. După lansarea acesteia, aplicația solicită denumirea noului sistem. Se recomandă utilizarea unei alte denumiri decât numele proiectului și al dispozitivului de nivel ierarhic maxim, pentru evitarea erorilor de compilare. De asemenea, se recomandă utilizarea în exclusivitate a caracterelor alfanumerice. De exemplu, alegem numele „system01”. Putem lăsa bifată opțiunea „Verilog” sau putem bifa opțiunea „VHDL”.

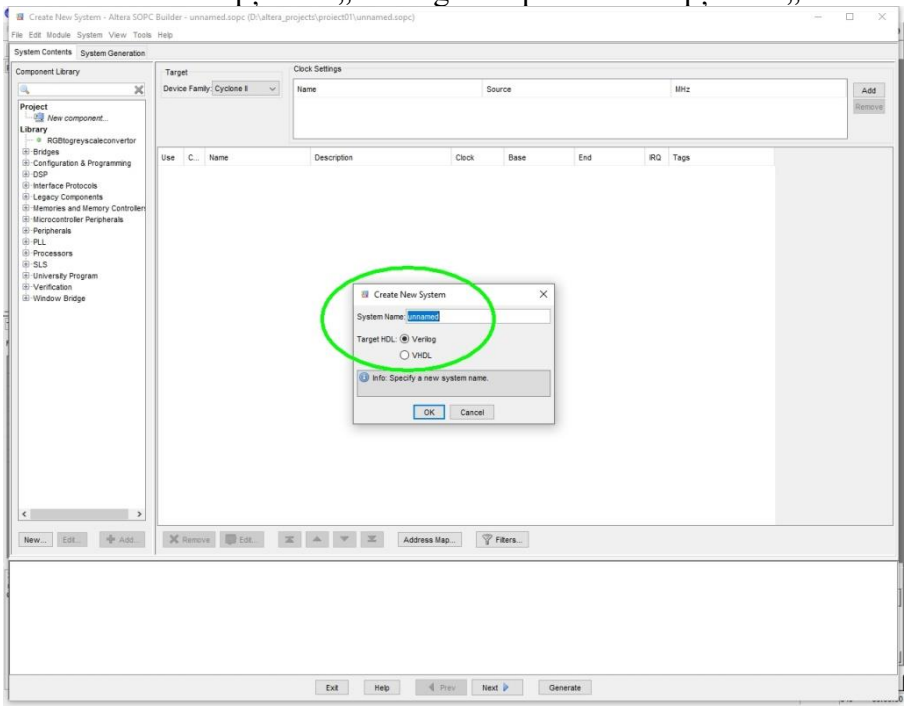


Fig. a2.10. Alegerea denumirii noului sistem.

Din acest moment putem începe construcția sistemului, alegând din lista din stânga componentele dorite – microprocesoare Nios II, memorii, porturi de intrare/ieșire/bidirecționale etc. De exemplu, alegem să construim o configurație formată dintr-un microprocesor Nios II economic și un *chip* de memorie RAM de 16kB.

Din lista din stânga executăm *click* pe semnul „+” din dreptul categoriei „Processors”, apoi executăm *click* pe componenta „Nios II Processor” și pe butonul „Add…” (**Fig. a2.11**). În fereastra care se deschide (**Fig. a2.12**), vom alege versiunea opțiunea „Nios II/e” („economic”). Căsuțele „Reset Vector” și „Exception Vector” nu pot fi completate deocamdată, așa că executăm *click* pe butonul „Finish”.

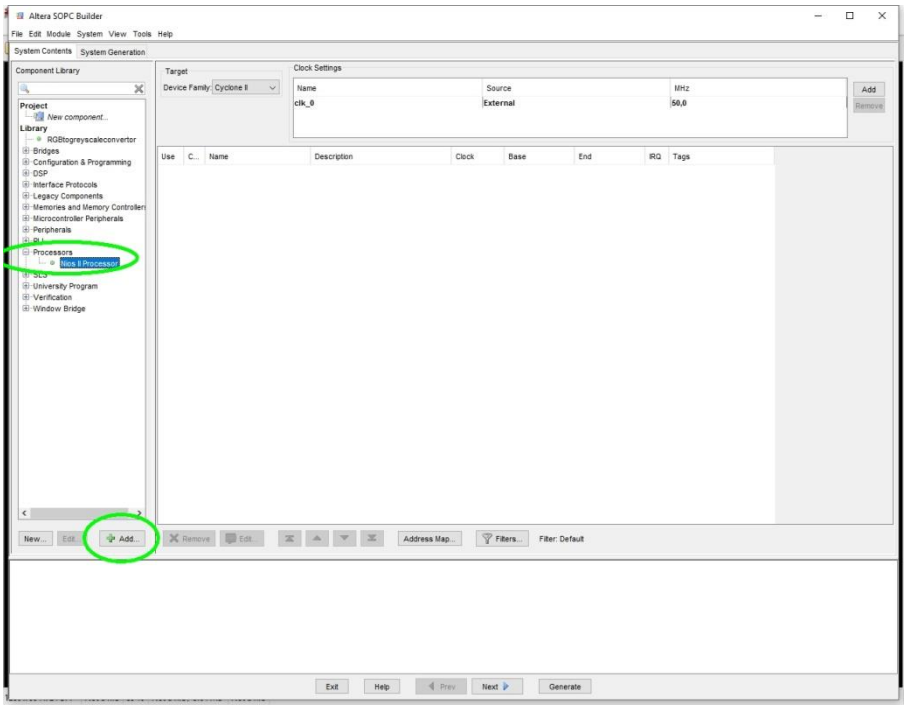


Fig. a2.11. Alegerea microprocesorului Nios II.

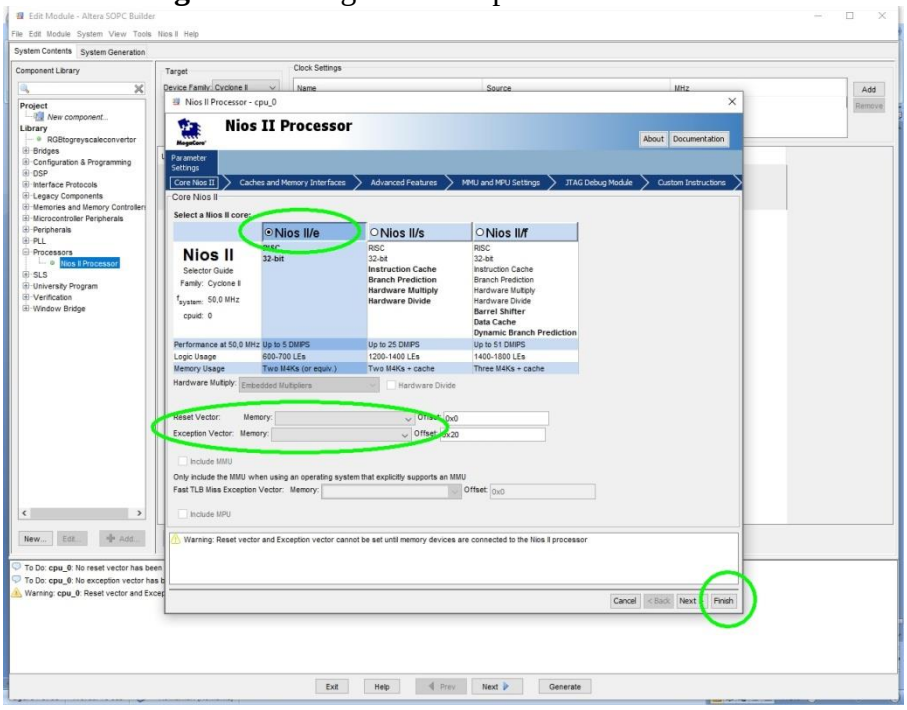


Fig. a2.12. Alegerea versiunii „economic” a microprocesorului Nios II.

După aceste operațiuni, microprocesorul Nios II/e este afișat în lista componentelor sistemului, așa că putem trece la adăugarea memoriei. Într-un mod asemănător alegem un bloc de memorie *on-chip* (**Fig. a2.13**).

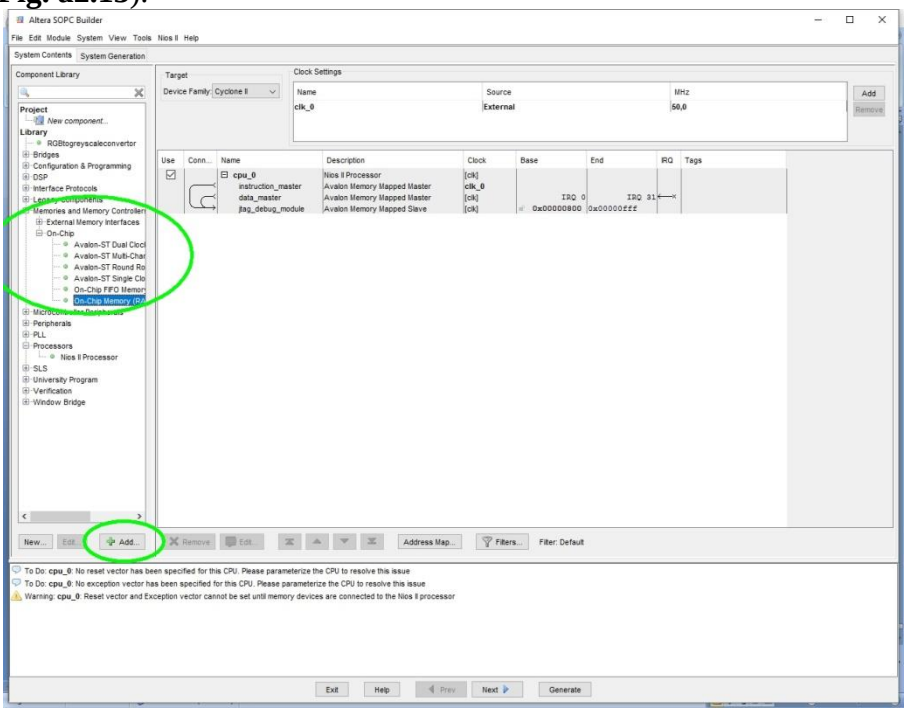


Fig. a2.13. Alegerea unui bloc de memorie *on-chip*.

Se deschide o fereastră cu setări ale memoriei (**Fig. a2.14**). Deocamdată, singura setare pe care o vom modifica este dimensiunea blocului de memorie. Vom modifica dimensiunea implicită de 4096 bytes, înlocuind această valoare cu 16834 bytes, respectiv 16 KB, după care executăm *click* pe butonul „Finish”.

Acum cele două dispozitive sunt afișate în lista componentelor sistemului. Observăm că în fereastra de mesaje din josul paginii avem două avertizări cu privire la setările „reset vector” și „exception vector” ale microprocesorului, așa că le vom efectua. Executăm dublu-*click* pe componenta „cpu_0”, deschizând astfel fereastra de setări ale microprocesorului (**Fig. a2.15**). Activând meniurile *drop-down* ale parametrilor „Reset Vector” și „Exception Vector”, alegem valorile „onchip_memory2_0” și executăm *click* pe butonul „Finish”. Din acest moment, cele două avertismente au dispărut și lista de componente conține microprocesorul „cpu_0” și memoria „onchip_memory2_0”.

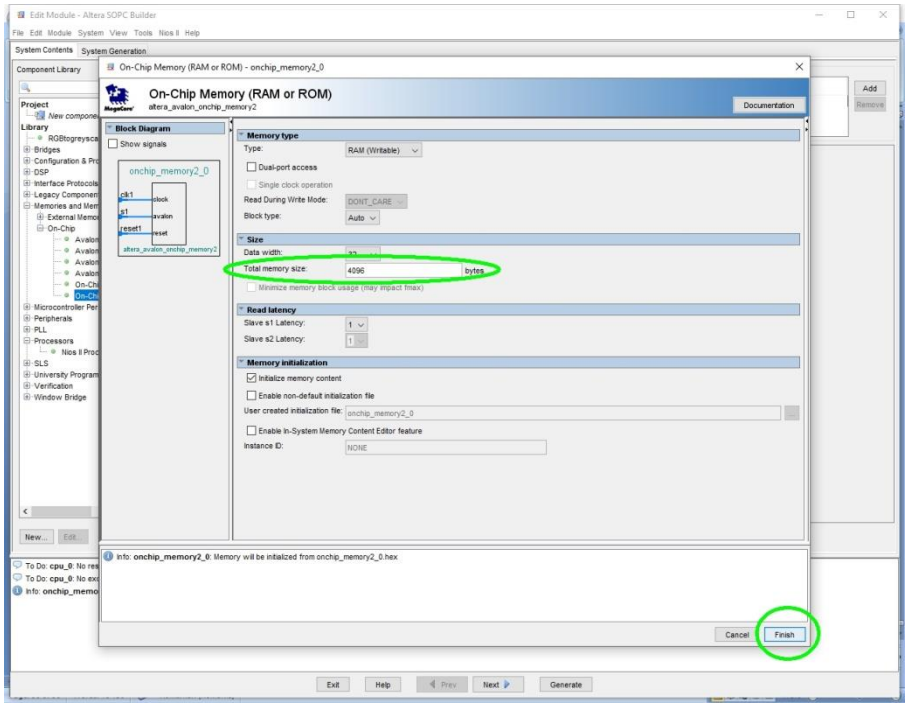


Fig. a2.14. Alegerea mărimii blocului de memorie.

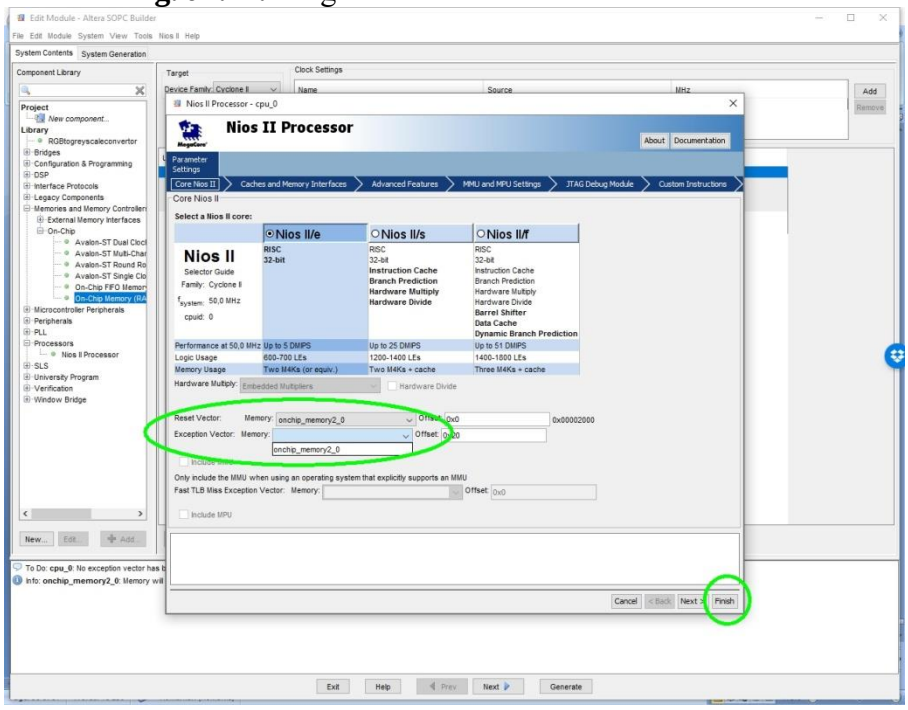


Fig. a2.15. Setarea parametrilor „Reset Vector” și „Exception Vector”.

Aplicația „SOPC Builder” alocă automat blocuri de memorie fiecărui component al sistemului. Este recomandată re-alocarea adreselor de bază după completarea sistemului, alegând opțiunea „Assign Base Addresses” din meniul „System” (**Fig. a2.16**).

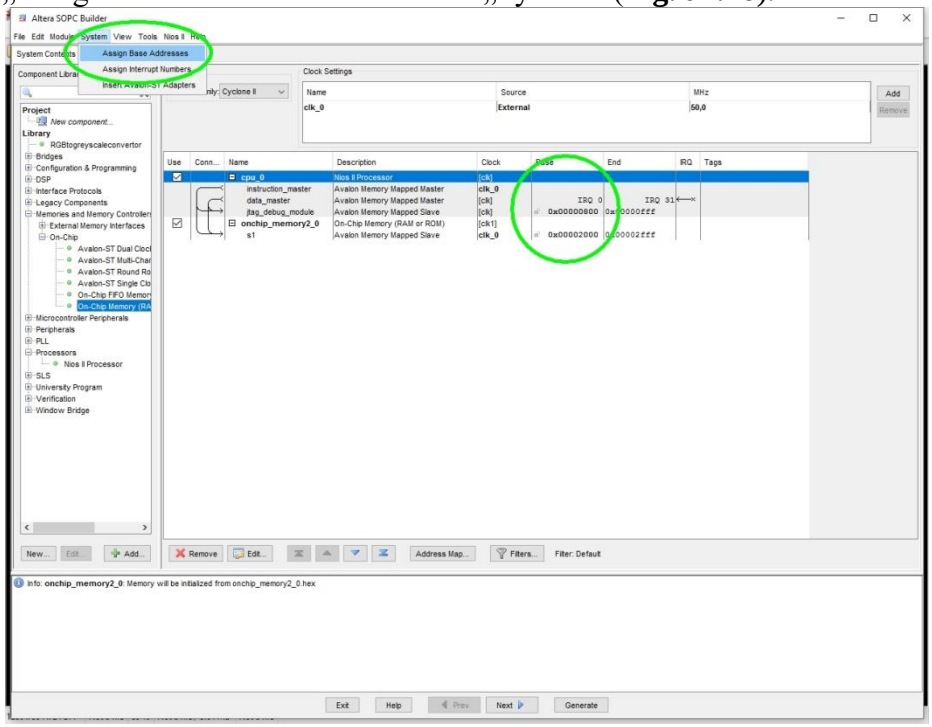


Fig. a2.16. Alocarea adreselor de bază.

Se remarcă faptul că adresele de bază ale celor două componente, alese automat în momentul adăugării lor în sistem, s-au modificat conform configurației finale a sistemului (**Fig. a2.17**). Acum vom face generarea sistemului finalizat, executând *click* pe butonul „Generate”.

La întrebarea „Save changes to unnamed?” („salvăm modificările în fișierul fără nume?”) vom da *click* pe butonul „Save”, apoi vom da numele „system01” în fereastra de dialog care apare, păstrând folderul existent, apoi efectuăm *click* pe butonul „Save”. Din acest moment începe generarea sistemului, operație care poate să dureze câteva minute. În acest timp, noua fereastră care apare se umple cu mesaje prin care se semnalează etapele generării. În final, dacă totul este în ordine, va apărea mesajul „SUCCESS: SYSTEM GENERATION COMPLETED” (**Fig. a2.18**). Acum aplicația „SOPC Builder” poate fi închisă dând *click* pe butonul „Exit” sau poate fi minimizată.

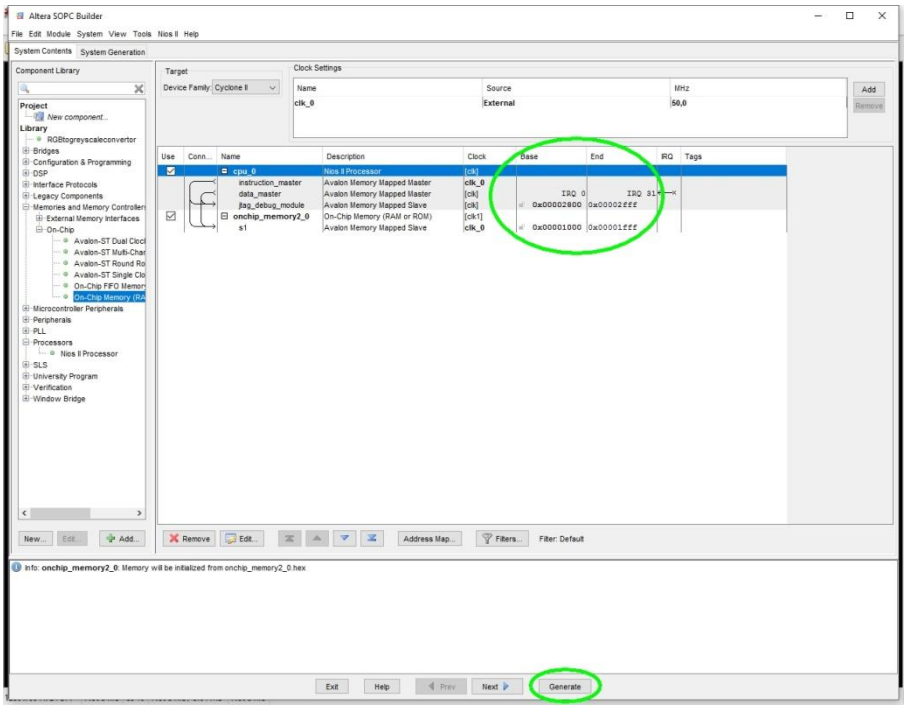


Fig. a2.17. Lansarea generării sistemului.

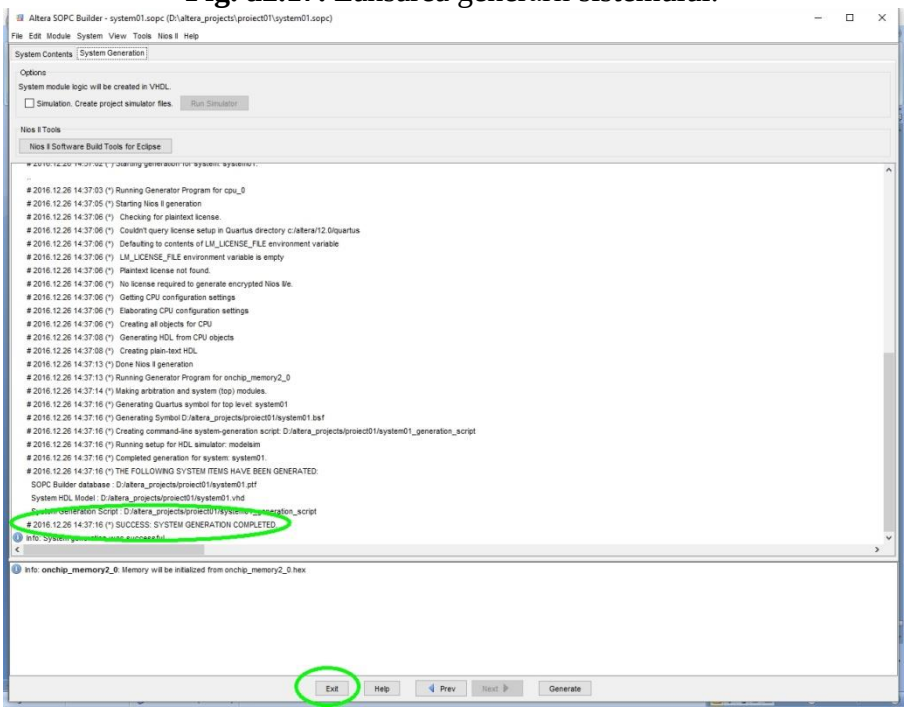


Fig. a2.18. Generarea sistemului, terminată cu succes.

În funcție de complexitatea proiectului, se poate construi schema bloc a acestuia cu opțiunea „Block Diagram / Schematic File” din meniul „File / New…”, i se pot aloca pinii reali ai FPGA-ului cu opțiunea „Import Assignments…” din meniul „Assignments” etc. După finalizarea acestor operații, sistemul poate fi încărcat pe placa Altera DE1 cu opțiunea „Programmer” din meniul „Tools”.

Fișierele create în urma generării sistemului vor fi utilizate de aplicația „Altera Monitor”, împreună cu programul în limbaj de asamblare sau în limbaj C, pentru a programa placa Altera DE1. Astfel, FPGA-ul de pe placă va fi configurat conform proiectului respectiv, iar programul va fi rulat pe această configurație. În cazul semnalării unor erori, se va reveni fie asupra configurației, fie asupra setărilor sau a programului de rulat.

La fiecare lucrare de laborator din prezentul îndrumător se pot realiza aplicații pe plăcile Altera DE1, prin care să se evidențieze pașii algoritmilor și valorile tuturor variabilelor implicate. De asemenea, prin intermediul modulului JTAG-UART se pot introduce informații și/sau valori direct de la tastatura computerului-gazdă, iar rezultatele pot fi vizualizate pe monitorul aceluiași computer-gazdă. Practic, cu ajutorul plăcilor Altera DE1, în combinație cu aplicațiile software Quartus, Altera UP Simulator și Altera Monitor, posibilitățile de efectuare a lucrărilor de laborator la disciplina „Structura și Organizarea Calculatoarelor” se diversifică foarte mult, crescând astfel nivelul de interes al studenților pentru acestea.