

### 3.12 Optimizarea formatului microinstrucțiunii

Optimizarea formatului microinstrucțiunii are drept scop minimizarea timpului de execuție a microrutinelor (obiectiv realizabil prin minimizarea lungimii microrutinelor).

Cea mai performantă microinstrucțiune este aceea care permite proiectantului să activeze întreg setul de comenzi elementare dorit, în fiecare punct al microprogramului de emulare. Așa cum afirmam la finele paragrafului 3.10, trebuie adoptat acel format de microinstrucțiune care nu introduce nici o restricție referitoare la setul de operații concurente pe care proiectantul de microcod ar dori să-l activeze la un moment dat; restricțiile acceptabile sunt doar cele impuse de structura *hardware* a procesorului controlat, nu și cele care provin din formatul microinstrucțiunii!

În paragraful 3.4 am proiectat microinstrucțiunea aferentă procesorului didactic. Am decis să implementăm trei tipuri de microinstructiuni (de ramificație, de transfer sincron și de transfer asincron) și am adoptat formatul fiecărui tip în parte. Fiecare tip de microinstrucțiune a fost dedicat unui anumit scop:

- Microinstrucțiunea de ramificație pentru implementarea legăturilor dintre microrutine
- Microinstrucțiunea de transfer sincron pentru controlul transferurilor de date între registrele interne aferente procesorului (cu eventuala procesare în ALU a informațiilor transferate).
- Microinstrucțiunea de transfer asincron pentru controlul transferurilor procesor-memorie (controlul operațiilor de citire din și respectiv de scriere în memorie).

Trebuie să menționăm faptul că această decizie a avut la bază rațiuni didactice și nu de performanță. Luând decizia de a proiecta trei tipuri diferite de microinstrucțiuni (fiecare tip fiind dedicat unui anumit scop), este evident că ne-am asumat conștient riscul de a „porni la drum” cu o microinstrucțiune neoptimizată. Altfel spus, cu trei microinstrucțiuni dedicate, care controlează tipuri diferite de operații, am acceptat premsa unei „organizări pe verticală” a microprogramului de emulare; trei operații diferite (un transfer sincron, un transfer asincron și o ramificație în microcod) nu pot fi codificate decât în microinstrucțiuni diferite, care nu pot fi procesate decât succesiv (microprogramare pe verticală) și nu concomitent (microprogramare pe orizontală). Prin urmare, decizia de a proiecta trei tipuri diferite de microinstrucțiuni, conduce inevitabil la microrutine mai lungi iar microrutinile lungi vor genera un procesor mai lent.

Pentru a putea procesa concurrent cele trei tipuri de operații (un transfer sincron, un transfer asincron și o ramificație) trebuie să plecăm de la premsa unei microinstrucțiuni generale, capabilă să controleze toate aceste operații. În această manieră, într-o singură microinstrucțiune se va putea codifica un set amplu de microcomenzi, care va iniția și executa (concurrent) un transfer sincron, un transfer asincron și o ramificație în microcod. Evident că prin această paralelizare a operațiilor controlate se va ajunge de fapt la o reducere a lungimii microrutinelor din cadrul microprogramului de emulare.

Microinstrucțiunea generală va fi mai performantă dar va avea o lungime mai mare. Vom încerca să introducem acest concept de microinstrucțiune generală pornind de la cele trei tipuri de microinstrucțiuni proiectate la 3.4. Pașii care ar trebui urmați pentru proiectarea microinstrucțiunii generale sunt:

1. Concatenarea microinstrucțiunilor de transfer sincron (pe 20 biți), de transfer asincron (pe 20 biți) și de ramificație (pe 20 biți) într-o singură microinstrucțiune generală (pe 60 biți).
2. Analiza microinstrucțiunii generale (obținută în pasul 1) în scopul de a identifica biți și respectiv câmpurile care își pierd semnificația (în cadrul microinstrucțiunii generale).
3. Eliminarea bițiilor (câmpurilor) care își pierd semnificația
4. Rafinarea formatului de microinstrucțiune generală (obținută în pasul 3) în vederea optimizării acestuia.

Pași 1, 2 și 3 sunt sintetizați în figura 3.47, în două variante de lucru.

#### **Pasul 1:**

Pasul 1 este identic în cele două variante: se concatenează microinstrucțiunile de transfer sincron, de transfer asincron și de ramificație definite la 3.4. Prin concatenare se obține un cuvânt binar, pe 60 biți, pe care îl vom denumi microinstrucțiune generală (acceptând în mod deliberat un mic abuz de limbaj). În această microinstrucțiune generală se regăsesc (în ordine) toate câmpurile definite în cadrul microinstrucțiunilor de transfer sincron, de transfer asincron și de ramificație.

În pasul 2 urmează să fie identificăți biți și respectiv câmpurile care își pierd semnificația iar în pasul 3 vor fi eliberați acești biți și respectiv aceste câmpuri.

#### **Pasul 2:**

Analiza pe care o facem în acest pas cu scopul de a identifica biții și respectiv câmpurile care își pierd semnificația conduce la următoarele concluzii:

- Biți:
  - MIR<sub>19</sub> și MIR<sub>18</sub> din cadrul microinstrucțiunii de transfer sincron
  - MIR<sub>19</sub> și MIR<sub>18</sub> din cadrul microinstrucțiunii de transfer asincron
  - MIR<sub>19</sub> din cadrul microinstrucțiunii de ramificație

își pierd semnificația în cadrul microinstrucțiunii generale și prin urmare, în pasul 3 vor fi eliberați. **Acești 5 biți sunt eliberați în ambele variante de lucru (vezi figura 3.47).**

#### **Argumentatia care stă la baza acestei decizii:**

*În cadrul celor trei tipuri de microinstrucțiuni (de transfer sincron, de transfer asincron și de ramificație), bitul MIR<sub>19</sub> specifică tipul microinstrucțiuni:*

*MIR<sub>19=0</sub> -defină microinstrucțiunea de ramificație*

*MIR<sub>19=1</sub> -defină microinstrucțiunea de transfer (sincron sau asincron)*

*În cazul microinstructiunilor de transfer, următorul bit (MIR<sub>18</sub>) definește tipul microinstructiunii de transfer:*

*MIR<sub>18</sub>=0 -definește microinstructiunea de transfer sincron*

*MIR<sub>18</sub>=1 - definește microinstructiunea de transfer asincron*

*Microinstructiune generală înseamnă în fapt un singur tip de microinstructiune și prin urmare nu mai sunt necesari acești biți de clasificare și respectiv identificare a celor trei tipuri diferite de microinstructiuni.*

- Biții MIR<sub>11+8</sub> din cadrul microinstructiunii de ramificație (cei mai semnificativi 4 biți din câmpul MICROADRESA DE SALT) nu sunt necesari și prin urmare, în pasul 3 vor fi eliminați.

*Argumentația care stă la baza acestei decizii a fost prezentată la 3.9 (vezi figura 3.41 și explicațiile aferente). Acești 4 biți sunt eliberați în ambele variante de lucru (vezi figura 3.47).*

- Câmpurile SURSA SBUS, SURSA DBUS, OPERAȚIE ALU și SURSA RBUS apar atât în microinstructiunea de transfer sincron cât și în cea de transfer asincron.

În varianta 1 de lucru, în pasul 3, vor fi integrate în microinstructiunea generală doar cele patru câmpuri care provin din microinstructiunea de transfer sincron; cele care provin din microinstructiunea de transfer asincron vor fi eliminate.

În varianta 2 de lucru, în pasul 3, vor fi integrate în microinstructiunea generală ambele seturi de patru câmpuri. Pentru diferențiere, cele patru câmpuri care provin din microinstructiunea de transfer sincron le vom eticheta cu indicele „S” (SURSA SBUS<sub>S</sub>, SURSA DBUS<sub>S</sub>, OPERAȚIE ALU<sub>S</sub> și SURSA RBUS<sub>S</sub>) iar cele care provin din microinstructiunea de transfer asincron le vom eticheta cu indicele „A” (SURSA SBUS<sub>A</sub>, SURSA DBUS<sub>A</sub>, OPERAȚIE ALU<sub>A</sub> și SURSA RBUS<sub>A</sub>).

Să reamintim faptul că în cele patru câmpuri pot fi codificate patru microcomenzi prin care:

- în cadrul microinstructiunii de transfer sincron se selectează registrele sursă, se controlează operația ALU și se selectează rezultatul ALU pentru a fi emis pe RBUS. Microcomenzi codificate în aceste patru câmpuri, împreună cu o microcomandă de tip „Primește Registrul” codificată în câmpul DESTINAȚIE RBUS, vor controla de fapt transferul sincron din interiorul procesorului (transfer registru-registrul cu eventuala procesare în ALU a informațiilor transferate). Microcomenzi codificate în aceste câmpuri sunt activate în starea ST<sub>1</sub> a automatului SEQ (vezi figura 3.32 cu explicațiile aferente).

- în cadrul microinstructiunii de transfer asincron, cele patru câmpuri în discuție au semnificație numai dacă în câmpul **MEMORY OPERATION** este codificată microcomanda **WRITE**; microcomenzile codificate în cele patru câmpuri vor fi activate în sarea **ST<sub>3</sub>** a automatului **SEQ** și au rolul de a emite pe **RBUS** (**MBUS**) datele de scris în memorie (vezi figura 3.32 cu explicațiile aferente).

**Argumentația care stă la baza deciziei din cadrul variantei 1 de lucru:**

*Microcomenzile codificate în cele patru câmpuri, în starea **ST<sub>1</sub>** a automatului **SEQ**, vor emite pe **RBUS** datele ce vor fi implicate în transferul sincron. Aceleași microcomenzi, reactivate pe durata stării **ST<sub>3</sub>** a automatului **SEQ**, vor emite pe **RBUS** (**MBUS**) datele ce vor fi scrise în memorie prin microcomanda **WRITE**. Prin urmare, în varianta 1 de lucru, datele ce vor fi încărcate în registrul destinație (transferul sincron) și datele ce vor fi memorate în memorie prin microcomanda **WRITE** (transferul asincron) nu pot fi decât identice (constrângere acceptată în mod deliberat).*

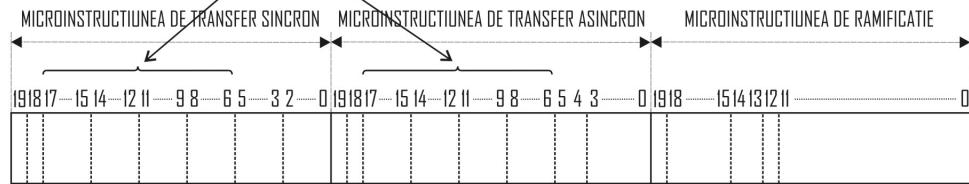
**Argumentația care stă la baza deciziei din cadrul variantei 2 de lucru:**

*Microcomenzile cu indicele „S” (SURSA **SBUS<sub>S</sub>**, SURSA **DBUS<sub>S</sub>**, OPERAȚIE **ALU<sub>S</sub>** și SURSA **RBUS<sub>S</sub>**) vor fi decodificate de un set de patru decodificatoare care vor fi activate pe durata stării **ST<sub>1</sub>** a automatului **SEQ**. Microcomenzile cu indicele „A” (SURSA **SBUS<sub>A</sub>**, SURSA **DBUS<sub>A</sub>**, OPERAȚIE **ALU<sub>A</sub>** și SURSA **RBUS<sub>A</sub>**) vor fi decodificate cu ajutorul unui alt set de patru decodificatoare care vor fi activate pe durata stării **ST<sub>3</sub>** a automatului **SEQ**. Prin urmare, în această variantă de lucru, datele ce vor fi încărcate în registrul destinație (transferul sincron) pot fi altele decât datele ce vor fi memorate în memorie prin microcomanda **WRITE** (transferul asincron).*

În concluzie, microinstructiunea generală obținută în varianta 2 de lucru este mai performantă decât cea obținută în varianta 1. Acest plus de performanță are un cost: lungimea microinstructiunii mai mare cu  $3 \times 4 = 12$  biți, ceea ce conduce la o creștere cu 12 biți a dimensiunii pe orizontală a planului de memorie MPM. Plusul de performanță poate fi însă fructificat numai pe microinstructiuni care fac scriere în memorie. Astfel de microinstructiuni apar în micropogramul de emulare doar în anumite microrutine de execuție (la instrucțiunile care specifică un operand **dest** și nu unul **src** și doar în cazul modurilor de adresare AI și AX).

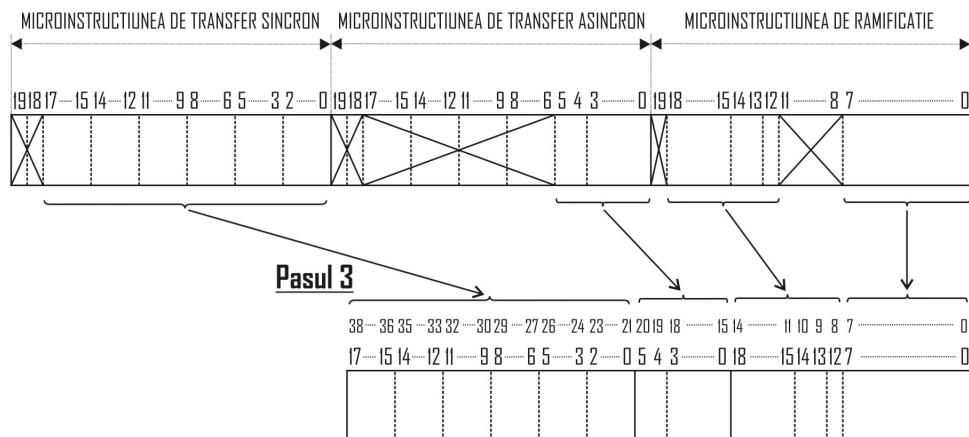


**Pasul 1**



**VARIANTA 1:**

**Pasul 2**



**VARIANTA 2:**

**Pasul 2**

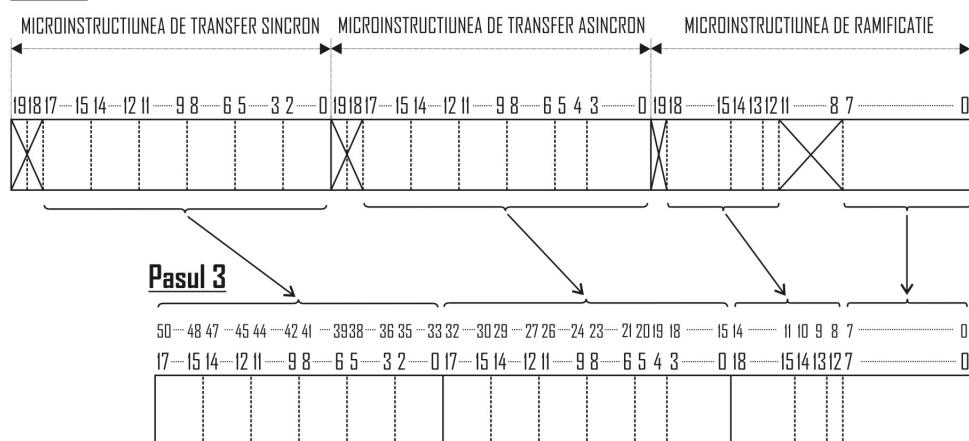


Fig 3.47 Proiectarea microinstructiunii generale

Pentru a alege varianta optimă de lucru proiectantul trebuie să facă o evaluare destul de complicată. Trebuie evaluat plusul de performanță pe care microinstrucțiunea generală obținută în varianta 2 de lucru îl aduce în raport cu microinstrucțiunea rezultată în varianta 1. Pentru aceasta trebuie să evalueze (statistic) frecvența instrucțiunilor care fac scriere în memorie în cadrul microrutinei de **Execuție** și să calculeze plusul de performanță obțenabil în varianta 2 de lucru. Acestui plus de performanță trebuie să i se contrapună costul suplimentar reprezentat de *hardware-ul adițional* necesar în varianta 2 de lucru.

Pasul 4 presupune rafinarea microinstrucțiunii obținute în pasul anterior. Rafinarea presupune un proces de analiză care utilizează, ca date de intrare, microprogramul de emulare. Să dăm un exemplu de problemă care trebuie soluționată în această etapă:

În microinstrucțiunea generală obținută la finele pasului 3 (în ambele variante de lucru) vom avea un câmp **OTHER OPERATION** (provenit din microinstrucțiunea de transfer sincron) și un câmp **SHIFT&OTHER OPERATION** (provenit din microinstrucțiunea de transfer asincron). Ambele sunt câmpuri de tip „*Other Operations*” pentru că conțin microcomenzi din această categorie. Întrebarea care se pune este:

***În microinstrucțiunea generală vom implementa într-adevăr două câmpuri de tip „Other Operations” sau vom reuni cele două câmpuri într-unul singur?***

Prin reunire am face economie la nivelul *hardware-ului* pentru că prin reunire lungimea microinstrucțiunii scade („microprogramare pe verticală”). Pe de altă parte, într-o microinstrucțiune cu un singur câmp „*Other Operations*” vom putea codifica o singură microcomandă din categoria „*other*”. Într-o microinstrucțiune cu două câmpuri de tip „*Other Operations*” („*Other Operations1*” și respectiv „*Other Operations2*”) vom putea codifica două microcomenzi de tip „*other*”, simultane (concurrente).

**Concluzia:** Din nou trebuie făcută o evaluare a microinstrucțiunilor din cadrul microprogramului de emulare și inventariate situațiile în care ar trebui activate concurrent (din aceeași microinstrucțiune) mai multe microcomenzi de tip „*other*”. Pe baza acestei evaluări vom obține răspunsul la întrebarea: „câte câmpuri de tip „*Other Operations*” vom implementa în microinstrucțiunea generală?“.

Răspunsul ar putea fi: „unul, două sau chiar mai multe câmpuri de acest tip”.

În final, să subliniem faptul că, la proiectarea unui procesor microprogramat, rafinarea formatului microinstrucțiunii se realizează printr-un proces iterativ. La prima iterație se schițează un format de microinstrucțiune. Nu acesta va fi formatul final deoarece nu se vor putea stabili cu exactitate, în această etapă, toate câmpurile și respectiv microcomenzile care trebuie codificate în aceste câmpuri. De exemplu, nu se cunosc cu exactitate indecsii care vor fi utilizati în microprogram. Nu se știe câte zerouri vor fi concatenate la dreapta bitilor unui anumit INDEX pentru că nu a fost încă evaluată lungimea microrutinelor vizate de acel INDEX. Nu se cunosc exact condițiile care trebuie testate în microprogram (în microinstrucțiunile de salt condiționat). Doar după elaborarea microprogramului (într-o primă variantă) se poate obține o listă a condițiilor testate și pe baza acestei liste se va putea definitiva câmpul **SUCCESOR** din formatul microinstrucțiunii.

Pe baza acestui format de microinstrucțiune (în schiță) se trece la elaborarea microprogramului de emulare. După elaborarea microprogramului se va reveni la formatul microinstrucțiunii pentru rafinarea lui. Procesul de rafinare presupune multiple evaluări și poate chiar multiple iterații.

Microinstrucțiunea generală poate controla multiple operații paralele:

- din zona de transfer sincron poate controla un transfer registru-registru (cu eventuala procesare în ALU a informațiilor transferate)
- din zona de transfer asincron poate controla un transfer procesor-memorie (citire din memorie sau scriere în memorie)
- din zona de ramificație poate specifica orice succesor.

Datorită potențialului ridicat de paralelism pe care îl prezintă, microinstrucțiunea generală va conduce la reducerea lungimii microrutinelor din cadrul microprogramului de emulare. Cu cât lungimea microrutinelor scade, cu atât performanțele procesorului cresc.

Vom ilustra plusul de performanță pe care microinstrucțiune generală îl poate aduce printr-un exemplu. Primele trei microinstrucțiuni din cadrul microprogramului de emulare aferent procesorului didactic (tabelul 3.16) realizează *fetch*-ul instrucțiunii din memorie. Acestea sunt:

NONE DBUS PC, PdALU, PmADR	; ADR←PC (încărcarea adresei de <i>fetch</i> în ; registrul ADR)
IFCH, +2PC	; IR←MEM <sub> ADR</sub> ( <i>fetch</i> instrucțiune) și ; PC←PC+2 (incrementare PC)
IF ACLOW JUMP PWFAIL else STEP	; test ACLOW (punct de ramificație în 2 direcții)

Prima este o microinstrucțiune de transfer sincron, a doua de transfer asincron și a treia de ramificație. În cazul aplicării formatului general, aceste trei microinstrucțiuni pot fi colapsate într-o singură de forma:

NONE DBUS PC, PdALU, PmADR, IFCH, +2PC, IF ACLOW JUMP PWFAIL else STEP

Această microinstrucțiune generală activează următoarele microcomenzi:

NONE	-în câmpul SURSA SBUS
PdPC	-în câmpul SURSA DBUS
DBUS	-în câmpul OPERAȚIE ALU
PdALU	-în câmpul SURSA RBUS
PmADR	-în câmpul DESTINAȚIE RBUS
IFCH	-în câmpul MEMORY OPERATION
+2PC	-în câmpul SHIFT&OTHER OPERATIONS

Primele cinci microcomenzi vor fi activate în starea  $ST_1$  a automatului SEQ (provin din zona de transfer sincron a microinstrucțiunii generale) și prin urmare vor realiza operația  $ADR \leftarrow PC$ . Următoarele două (IFCH și +2PC) provin din zona de transfer asincron a microinstrucțiunii generale și vor realiza operațiile  $PC \leftarrow PC+2$  (în starea  $ST_1$  a automatului SEQ) și respectiv  $IR \leftarrow MEM|ADR$  (*fetch*-ul instrucțiunii), în starea  $ST_3$  a automatului SEQ. Prin urmare, mai întâi se transferă adresa de *fetch* din registrul PC în registrul ADR (în starea  $ST_1$ ) și mai apoi se extrage codul instrucțiunii din memorie (în starea  $ST_3$ ); este exact succesiunea corectă a operațiilor.

Comanda +2PC este activată tot în starea  $ST_1$  a automatului SEQ. Dacă vom proiecta o microcomandă +2PC de tip impuls atunci frontul activ (urcător) al acestei microcomenzi va fi sincron cu frontul activ al microcomenții PmADR (care este tot de tip impuls). În aceste condiții, în registrul ADR se va încărca valoarea existentă în registrul PC înaintea operației de incrementare. Altfel spus, mai întâi se face operația  $ADR \leftarrow PC$  și apoi operația  $PC \leftarrow PC+2$ ; este exact succesiunea corectă a operațiilor.

Suplimentar, în starea  $ST_1$  a automatului SEQ, microinstrucțiunea generală impune și execuția succesorului condiționat:

IF ACLOW JUMP PWFAIL else STEP.

Pentru execuția acestui succesor dublu, în starea  $ST_1$ , automatul SEQ activează comanda:

- LdMAR, dacă  $ACLOW=1$  (se execută succesorul JUMP PWFAIL)
- +1MAR, dacă  $ACLOW=0$  (se execută succesorul STEP)

Cele trei microinstrucțiuni pe care am construit exemplul de colapsare fac parte din microrutina **Fetch Instrucțiune** (tabelul 3.16). Prin colapsare (raport 3:1) lungimea microrutinei **Fetch Instrucțiune** se reduce semnificativ și deci performanța procesorului crește. Evident că, la nivelul întregului microprogram de emulare, colapsarea nu se va putea face în raportul de trei la unu deoarece vor fi și situații în care colapsarea nu este posibilă sau în care este parțial posibilă.

Pentru a o face însă posibilă, trebuie introdusă o microinstrucțiune generală, capabilă să controleze, multiple operații concurente:

- transferuri registru-registru (interne procesorului), cu eventuala procesare în ALU a informațiilor transferate
- transferuri procesor-memorie (citire din și respectiv scriere în memorie)
- operații din categoria „other” (interne procesorului)
- ramificații în microcod (succesori STEP, JUMP și JUMPI, condiționați și necondiționați)

Un lucru rămâne cert: prin introducerea microinstrucțiunii generale performantele procesorului micropogrammat cresc semnificativ.