IMPLEMENTAREA UNUI PROCESOR RISC

În această lucrare de laborator se descrie arhitectura unui procesor RISC și se prezintă detalii necesare implementării acestui procesor pe o placă de dezvoltare Nexys4 DDR. Arhitectura de bază a acestui procesor este cea descrisă în lucrarea "Logic and Computer Design Fundamentals" de M. Morris Mano și Charles R. Kime. Pentru simplificarea codificării instrucțiunilor și pentru mai multă claritate s-au efectuat unele modificări ale arhitecturii, au fost modificate semnalele de comandă și mnemonicele instrucțiunilor, fiind adăugate și unele instrucțiuni suplimentare.

1. Schema bloc simplificată a procesorului RISC

O caracteristică importantă a arhitecturilor RISC este execuția majorității instrucțiunilor într-un singur ciclu de ceas. În aceste condiții, pentru a fi posibilă reducerea perioadei semnalului de ceas și, implicit, creșterea frecvenței acestui semnal, este necesară utilizarea tehnicii *pipeline*. Pentru aceasta, calea de date trebuie împărțită în mai multe secțiuni sau etaje, între etaje fiind inserate registre care păstrează rezultatele parțiale și permit transferul acestor rezultate la etajele următoare. Perioada semnalului de ceas trebuie aleasă astfel încât să fie egală sau mai mare cu întârzierea maximă a fiecăruia dintre etaje, pentru ca operațiile executate de fiecare etaj să poată fi terminate într-un ciclu de ceas. Această perioadă va putea fi însă mult mai redusă decât în cazul în care ar trebui parcursă întreaga cale de date într-un ciclu de ceas.

Procesorul RISC care va fi implementat se caracterizează prin memorii separate pentru instrucțiuni și pentru date, arhitectură de tip *Load/Store* (toate operațiile sunt executate între registre, memoria de date fiind accesată doar prin instrucțiunile de încărcare și de memorare), un număr de trei formate de instrucțiuni, toate cu aceeași lungime de 32 de biți, un număr redus de moduri de adresare și instrucțiuni simple care execută doar operații elementare. Aceste operații pot fi executate printr-o singură trecere prin calea de date *pipeline*, într-un singur ciclu de ceas.

Figura 1 ilustrează schema bloc simplificată a procesorului RISC. Schema este împărțită în două secțiuni, unitatea de control și calea de date. Unitatea de control conține contorul de program PC, memoria de instrucțiuni, registrul de instrucțiuni RI, decodificatorul de instrucțiuni și registrele dintre etajele *pipeline*. Contorul de program este actualizat în fiecare ciclu de ceas și conține adresa instrucțiunii care va fi extrasă din memoria de instrucțiuni. Adresele din contorul de program PC sunt adrese de cuvinte și nu de octeți. Decodificatorul de instrucțiuni, care generează semnalele de comandă necesare funcționării procesorului, este un circuit combinațional. Acest fapt, combinat cu structura căii de date și utilizarea unor memorii separate de instrucțiuni și de date, permite extragerea unei instrucțiuni și execuția acesteia într-un singur ciclu de ceas.

Decodificatorul de instrucțiuni generează semnalele de comandă descrise în continuare. Primele trei semnale sunt obținute direct din câmpurile instrucțiunii.

- *AdrSA* reprezintă adresa registrului sursă *A*, care conține primul operand al instrucțiunii.
- *AdrSB* reprezintă adresa registrului sursă *B*, care conține al doilea operand al instrucțiunii.
- *AdrD* reprezintă adresa registrului destinație, care va conține rezultatul operației executate de instrucțiune.

- *MxB* este semnalul de selecție pentru multiplexorul MUXB, prin care se aplică la intrarea *B* a unității aritmetice și logice fie conținutul registrului sursă *B*, fie o valoare constantă dintr-un câmp al instrucțiunii.
- *OpUAL* reprezintă codul funcției executate de unitatea aritmetică și logică UAL.
- *MemWr* este semnalul de validare a operației de scriere în memoria de date.
- *MxD* este semnalul de selecție pentru multiplexorul MUXD, prin care se aplică la portul de intrare *DateD* al setului de registre fie ieșirea unității aritmetice și logice, fie cuvântul citit din memoria de date.
- *RegWr* este semnalul de validare a scrierii în setul de registre.

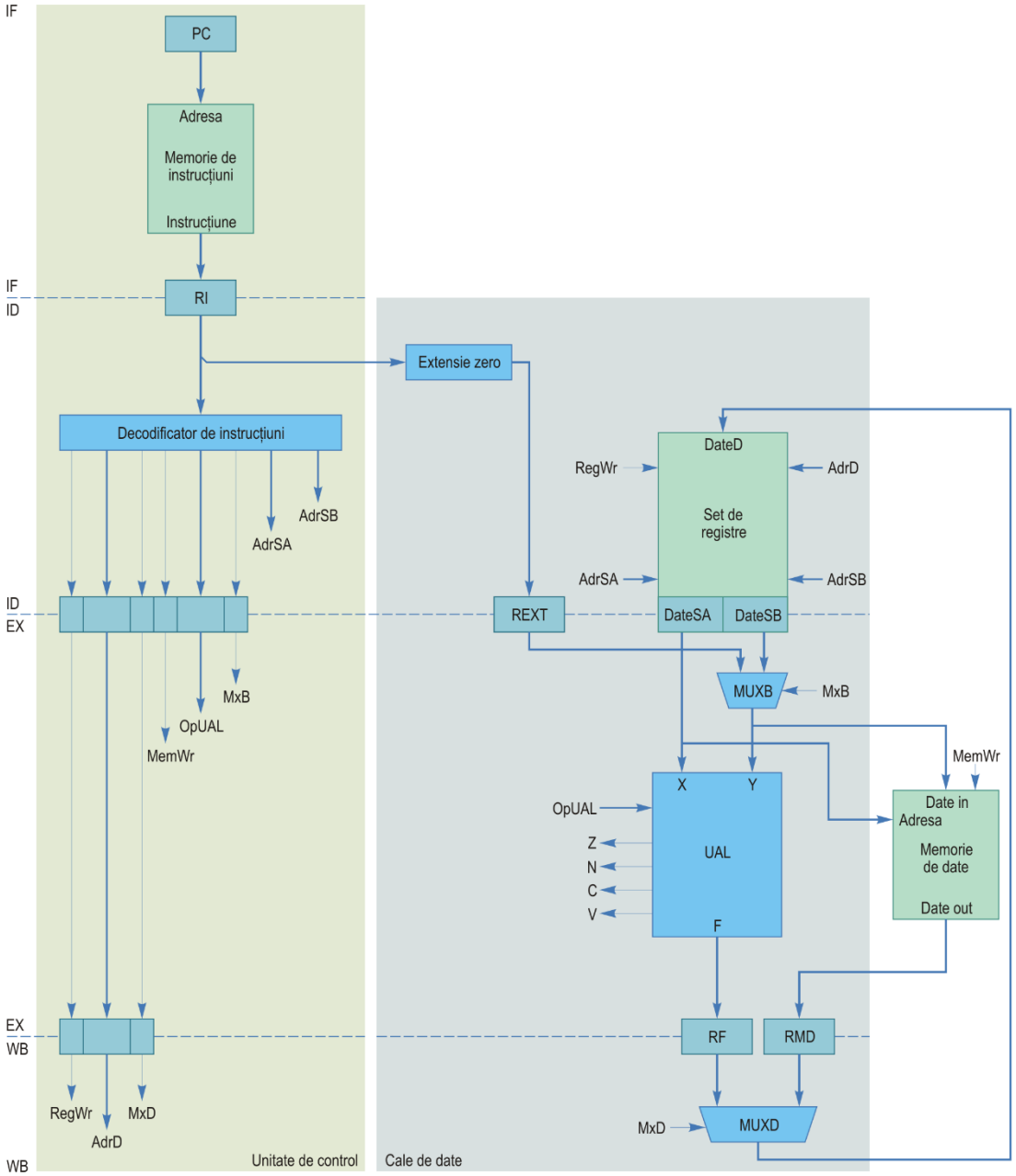

Figura 1. Schema bloc simplificată a unității de control și a căii de date pentru procesorul RISC.

Calea de date conține setul de registre, circuitul pentru extensia cu zero a constantelor de 16 biți la valori de 32 de biți, unitatea aritmetică și logică UAL, multiplexoarele MUXB și MUXD, ca și registrele necesare pentru transferul informațiilor între etajele *pipeline*. Setul de registre conține 16 registre de câte 32 de biți și funcționează ca o memorie rapidă cu 16 cuvin-

te și cu trei porturi de acces. La portul de citire *DateSA* se transferă conținutul primului registru sursă al unei instrucțiuni, registru adresat prin liniile *AdrSA*. La portul de citire *DateSB* se transferă conținutul celui de-al doilea registru sursă, care este adresat prin liniile *AdrSB*. Aceste două porturi de citire sunt prevăzute cu câte un registru de ieșire care păstrează conținutul registrului corespunzător. Al treilea port, *DateD*, permite scrierea registrului destinație al unei instrucțiuni, registru care este adresat prin liniile *AdrD*. Toate cele trei accese la setul de registre au loc în același ciclu de ceas. Scrierea în registre este validată prin activarea semnalului *RegWr*.

Unitatea aritmetică și logică UAL execută operații aritmetice de adunare și scădere, operații logice obișnuite (AND, OR, XOR, NOT) și operații de deplasare logică la dreapta și la stânga cu o poziție. Această unitate setează indicatorii de stare *Z* (*zero*), *N* (*negative*), *C* (*carry*) și *V* (*overflow*). Condițiile în care sunt setați acești indicatori sunt descrise în secțiunea 3.2, care prezintă detalii despre unitatea aritmetică și logică.

În figura 1 se pot distinge patru etaje *pipeline*: IF (*Instruction Fetch*), ID (*Instruction Decode*), EX (*Execution*) și WB (*Write-Back*). Etajul IF conține doar elemente ale unității de control. În acest etaj, se extrage instrucțiunea din memoria de instrucțiuni și se actualizează contorul de program PC. Din cauza complexității mai ridicate a gestionării instrucțiunilor de salt într-o cale de date *pipeline*, actualizarea contorului de program este limitată doar la o incrementare în acest etaj. În etajul de execuție EX va fi adăugată însă o logică suplimentară care va permite încărcarea contorului de program cu adresa de salt dintr-o instrucțiune de salt; detalii sunt indicate în secțiunile următoare. Între primul și al doilea etaj este amplasat registrul de instrucțiuni, cu rol de registru *pipeline* între aceste etaje, permițând transmiterea codului instrucțiunii către etajul următor ID.

Etajul ID realizează decodificarea instrucțiunii din registrul de instrucțiuni și generarea semnalelor de comandă necesare execuției instrucțiunii. O parte a semnalelor de comandă (*AdrSA*, *AdrSB*) sunt utilizate în acest etaj, în timp ce restul semnalelor de comandă sunt înscrise într-un registru *pipeline* de la ieșirea acestui etaj pentru a fi utilizate în etajele următoare. De asemenea, etajul ID realizează și încărcarea operanzilor sursă din setul de registre. Deoarece setul de registre conține registre de ieșire atașate celor două porturi de citire, acestea se utilizează și ca registre *pipeline* pentru transmiterea operanzilor sursă la etajul următor EX în locul unor registre suplimentare. Ieșirea circuitului pentru extensia cu zero a constantelor de 16 biți la valori de 32 de biți este înscrisă în registrul REXT pentru a fi transmisă etajului următor.

Al treilea etaj *pipeline* este etajul de execuție, EX. Pentru majoritatea instrucțiunilor, în acest etaj se execută o operație aritmetică, o operație logică sau o operație cu memoria. De aceea, semnalele de comandă utilizate în acest etaj sunt *MxB*, *OpUAL* și *MemWr*. Celelalte semnale de comandă sunt înscrise în registrul *pipeline* de la ieșirea etajului EX. Rezultatul obținut la ieșirea unității aritmetice și logice este înscris în registrul RF. În cazul unei instrucțiuni de citire a memoriei de date, cuvântul citit din această memorie este înscris în registrul RMD.

Ultimul etaj *pipeline*, WB, selectează prin multiplexorul MUXD valoarea care va fi înscrisă în registrul destinație din setul de registre. Pentru această operație de scriere se utilizează semnalele de comandă *RegWr*, *AdrD* și *MxD*.

Atunci când se utilizează tehnica *pipeline* pentru execuția instrucțiunilor, pot apare probleme care sunt cunoscute sub numele de hazarduri. Acestea reprezintă probleme de sincronizare și apar deoarece execuția unei operații într-un sistem *pipeline* este întârziată cu unul sau mai multe cicluri de ceas din momentul în care instrucțiunea care specifică operația a fost extrasă din memorie. Principalele tipuri de hazarduri sunt hazardurile de date și hazardurile de control. *Hazardurile de date* apar atunci când o instrucțiune încearcă să utilizeze rezultatul unei instrucțiuni precedente ca operand înainte ca rezultatul să fie disponibil, de exemplu, înaintea înscrierii rezultatului într-un registru. *Hazardurile de control* apar ca urmare a execuției instrucțiunilor de salt, care modifică fluxul execuției programului. De exemplu, la întâlnirea unei instrucțiuni de salt condiționat, condiția de salt va fi evaluată atunci când instrucțiunea se află în etajul de execuție EX, iar în cazul în care saltul va fi executat contorul de instrucțiuni va fi încărcat cu adresa de salt în etajul WB. În acest moment însă, instrucțiunile care urmează

după instrucțiunea de salt se află în diferite faze de execuție și își pot termina execuția, chiar dacă intenția programatorului a fost ca aceste instrucțiuni să nu fie executate.

Hazardurile descrise anterior pot fi eliminate prin metode software sau prin metode hardware. De exemplu, o metodă software pentru rezolvarea problemei hazardului de date constă în rearanjarea instrucțiunilor programului de către compilator sau programator astfel încât o operație de citire a unui anumit registru să se execute într-un ciclu de ceas următor unei operații de scriere în același registru. Dacă rearanjarea instrucțiunilor nu este posibilă, se pot insera instrucțiuni NOP în program pentru a întârzia instrucțiunea care execută citirea registrului până când scrierea anterioară în acest registru va fi terminată. Problema hazardului de control poate fi rezolvată prin metoda software numită întârzierea salturilor, care constă în inserarea, după instrucțiunea de salt, a unor instrucțiuni care pot fi introduse în sistemul *pipeline* indiferent dacă saltul va fi executat sau nu. În particular, după instrucțiunea de salt se pot insera instrucțiuni NOP, a căror execuție nu va avea efecte nedorite asupra corectitudinii programului. În secțiunea 4 se prezintă o metodă hardware de rezolvare a problemei hazardului de control, metodă numită predicția salturilor.

2. Arhitectura setului de instrucțiuni

2.1. Formatul instrucțiunilor

Procesorul RISC care va fi implementat conține 16 registre care sunt accesibile programatorului; ele sunt notate cu R0, R1, … R15. Toate registrele sunt de 32 de biți. Aceste registre sunt grupate în setul de registre. Spre deosebire de alte implementări ale unor procesoare RISC, registrul R0 nu conține valoarea constantă zero, ci poate fi utilizat ca oricare alt registru. Dacă este necesar ca un anumit registru să conțină valoarea zero, acel registru poate fi încărcat cu valoarea zero printr-o singură operație logică SAU EXCLUSIV.

Instrucțiunile procesorului RISC utilizează unul din cele trei formate de instrucțiuni care sunt ilustrate în figura 2. Toate formatele utilizează un singur cuvânt de 32 de biți, din care 8 biți cei mai semnificativi reprezintă codul operației din câmpul CODOP. Primul format specifică trei registre. Cele două registre sursă adresate de câmpurile RSA și RSB de câte 4 biți conțin cei doi operanzi ai unei instrucțiuni aritmetice sau logice. Al treilea registru, care este adresat de câmpul RD de 4 biți, reprezintă registrul destinație și va conține rezultatul unei operații aritmetice sau logice.

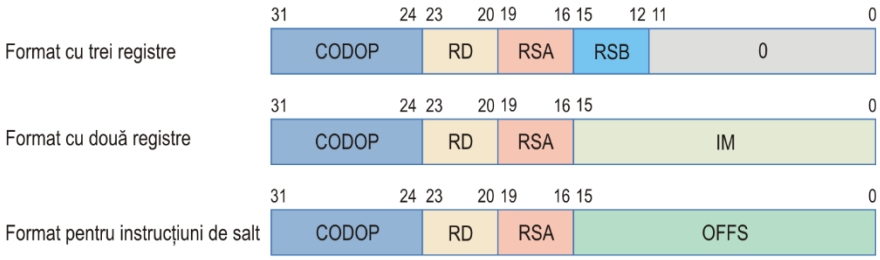

Figura 2. Formatele de instrucțiuni ale procesorului RISC.

Al doilea format al instrucțiunilor specifică doar două registre, registrul sursă A, care este adresat de câmpul RSA, și registrul destinație, care este adresat de câmpul RD. Acest format este utilizat de instrucțiunile aritmetice și logice care necesită o valoare imediată ca al doilea operand. Acest operand este preluat din câmpul IM de 16 biți al instrucțiunii. Câmpul IM este utilizat și pentru a păstra numărul de poziții cu care se deplasează conținutul registrului sursă de către instrucțiunile de deplasare logică la dreapta sau la stânga. Numărul de poziții cu care se realizează deplasarea poate fi o valoare între 0 și 31, valoarea fiind păstrată în cele 5 poziții mai puțin semnificative ale câmpului IM.

Al treilea format al instrucțiunilor este asemănător cu al doilea format, dar câmpul OFFS conține în acest caz deplasamentul (partea de offset) a adresei de destinație pentru instrucțiunile de salt și instrucțiunea de apel al unei proceduri. Adresa de destinație a acestor

instrucțiuni se obține prin adunarea deplasamentului din câmpul OFFS la conținutul contorului de program PC. Deci, instrucțiunile de salt și instrucțiunea de apel utilizează adresarea relativă, registrul PC fiind actualizat prin adunarea la conținutul acestui registru a valorii din câmpul OFFS, valoare care este considerată un număr cu semn în complement față de 2. Pentru instrucțiunea de salt necondiționat se utilizează doar câmpul OFFS al instrucțiunii și nu se utilizează câmpurile RSA și RD. Pentru instrucțiunile de salt condiționat se utilizează în plus câmpul RSA, care specifică registrul sursă al cărui conținut este testat pentru a determina dacă saltul va fi executat sau nu. Pentru instrucțiunea de apel al unei proceduri, în locul câmpului RSA se utilizează câmpul RD, care specifică registrul în care se va salva adresa de revenire.

În tabelul 1 se prezintă instrucțiunile procesorului RISC. Pentru fiecare instrucțiune se indică doar mnemonica și semnificația acesteia. Detalii suplimentare despre execuția instrucțiunilor vor fi prezentate în continuarea acestei secțiuni.

Tabelul 1. Mnemonica și semnificația instrucțiunilor procesorului RISC.

Mnemonica	Semnificația
NOP	No Operation
MOVA	Move A
ADD	Add
SUB	Subtract
AND	Logical AND
OR	Logical OR
XOR	Logical Exclusive-OR
NOT	Logical NOT
ADDI	Add Immediate
SUBI	Subtract Immediate
ANDI	Logical AND Immediate
ORI	Logical OR Immediate
XORI	Logical Exclusive-OR Immediate
ADDU	Add Immediate Unsigned
SUBU	Subtract Immediate Unsigned
MOVB	Move B
SHR	Shift Right Logical
SHL	Shift Left Logical
LD	Load
ST	Store
JMPR	Jump Register
SGTE	Set if Greater Than or Equal
SLT	Set if Less Than
BZ	Branch on Zero
BNZ	Branch on Nonzero
JMP	Jump
JMPL	Jump and Link
HALT	Halt

În tabelul 2 sunt descrise operațiile executate de instrucțiunile procesorului RISC, pentru fiecare instrucțiune fiind prezentat și un exemplu în limbaj de asamblare. În descrierea operației executate de o anumită instrucțiune, R(RD), R(RSA) și R(RSB) reprezintă registrul destinație, registrul sursă A, respectiv registrul sursă B, care sunt adresate prin câmpurile corespunzătoare din formatul instrucțiunii. M(R(RSA)) reprezintă locația de memorie adresată de către registrul R(RSA). Toate operațiile sunt elementare și, în general, pot fi descrise printr-un singur transfer între registre. Singurele instrucțiuni care pot accesa memoria de date sunt instrucțiunile LD și ST. Instrucțiunile cu un operand imediat permit reducerea numărului de accese la memoria de date atunci când se utilizează constante. Deoarece câmpul unei valori imediate este de 16 biți, valoarea trebuie extinsă pentru a forma un operand de 32 de biți. Pen-

tru operațiile logice, valoarea este extinsă cu zerouri în cele 16 poziții mai semnificative, ceea ce se indică prin notația *extz* (IM). Pentru operațiile aritmetice, valoarea este extinsă cu semn, ceea ce se indică prin notația *exts* (IM). Pentru a forma un operand de 32 de biți în complement față de 2, bitul de semn al valorii imediate (bitul 15) este copiat în cei 16 biți mai semnificativi ai operandului. O notație similară, *exts* (OFFS), se utilizează pentru a indica extensia cu semn a câmpului OFFS care conține deplasamentul adresei de destinație pentru instrucțiunile de salt.

Tabelul 2. Operații executate de instrucțiunile procesorului RISC și exemple de instrucțiuni.

Mnemonica	Operație	Exemplu
NOP	---	NOP
MOVA	R(RD) ← R(RSA)	MOVA R1,R4
ADD	R(RD) ← R(RSA) + R(RSB)	ADD R2,R1,R4
SUB	R(RD) ← R(RSA) – R(RSB)	SUB R3,R7,R8
AND	R(RD) ← R(RSA) **and** R(RSB)	AND R4,R3,R6
OR	R(RD) ← R(RSA) **or** R(RSB)	OR R4,R5,R7
XOR	R(RD) ← R(RSA) **xor** R(RSB)	XOR R8,R2,R4
NOT	R(RD) ← **not** R(RSA)	NOT R9,R1
ADDI	R(RD) ← R(RSA) + *exts* (IM)	ADDI R2,R0,4
SUBI	R(RD) ← R(RSA) – *exts* (IM)	SUBI R5,R7,1
ANDI	R(RD) ← R(RSA) **and** *extz* (IM)	ANDI R8,R2,0x8000
ORI	R(RD) ← R(RSA) **or** *extz* (IM)	ORI R1,R3,0x4000
XORI	R(RD) ← R(RSA) **xor** *extz* (IM)	XORI R6,R6,1
ADDU	R(RD) ← R(RSA) + *extz* (IM)	ADDU R2,R3,128
SUBU	R(RD) ← R(RSA) – *extz* (IM)	SUBU R5,R7,10
MOVB	R(RD) ← R(RSB)	MOVB R4,R5
SHR	R(RD) ← **shr** (R(RSA)) cu IM poziții	SHR R2,R2,4
SHL	R(RD) ← **shl** (R(RSA)) cu IM poziții	SHL R6,R6,8
LD	R(RD) ← M(R(RSA))	LD R8,R3
ST	M(R(RSA)) ← R(RSB)	ST R3,R5
JMPR	PC ← R(RSA)	JMPR R3
SGTE	**if** R(RSA) ≥ R(RSB) **then** R(RD) ← 1	SGTE R4,R5,R7
SLT	**if** R(RSA) < R(RSB) **then** R(RD) ← 1	SLT R4,R5,R7
BZ	**if** R(RSA) = 0 **then** PC ← PC + 1 + *exts* (OFFS)	BZ R3,8
BNZ	**if** R(RSA) ≠ 0 **then** PC ← PC + 1 + *exts* (OFFS)	BNZ R3,-12
JMP	PC ← PC + 1 + *exts* (OFFS)	JMP 10
JMPL	R(RD) ← PC + 1, PC ← PC + 1 + *exts* (OFFS)	JMPL R8,16
HALT	PC ← PC	HALT

Instrucțiunile de salt condiționat sunt BZ, BNZ, SGTE și SLT. Instrucțiunile BZ și BNZ determină dacă registrul specificat în respectiva instrucțiune conține valoarea zero, respectiv o valoare diferită de zero, și efectuează saltul în mod corespunzător. Instrucțiunea SGTE înscrie valoarea 1 în registrul destinație dacă registrul sursă A conține o valoare mai mare sau egală cu cea din registrul sursă B; în caz contrar, se înscrie valoarea 0 în registrul destinație. Registrul destinație poate fi examinat apoi printr-o instrucțiune următoare pentru a determina dacă este zero sau nu. Astfel, prin utilizarea a două instrucțiuni, se pot determina valorile relative a doi operanzi sau semnul unui operand (prin setarea prealabilă a registrului sursă B la 0). În mod similar, instrucțiunea SLT înscrie valoarea 1 în registrul destinație dacă registrul sursă A conține o valoare mai mică decât cea din registrul sursă B și 0 în caz contrar.

Instrucțiunea JMPL permite apeluri de proceduri. Conținutul registrului PC este incrementat și este memorat în registrul destinație specificat în instrucțiune. Apoi, suma dintre conținutul registrului PC și extensia cu semn a câmpului OFFS din instrucțiune este înscrisă

în registrul PC. Pentru revenirea din procedura apelată se poate utiliza instrucțiunea JMPR cu același registru sursă A care a fost utilizat ca registru destinație în instrucțiunea de apel JMPL. Dacă într-o procedură trebuie apelată o altă procedură, fiecare procedură apelată trebuie să utilizeze un alt registru pentru păstrarea adresei de revenire. Se poate utiliza și o stivă software care mută adresele de revenire din registrul destinație în memorie la începutul unei proceduri apelate și le reface în registrul sursă A înaintea revenirii din procedură.

2.2. Moduri de adresare

Procesorul RISC are patru moduri de adresare. Acestea sunt adresarea registrelor, adresarea indirectă prin registru, adresarea imediată și adresarea relativă. Modul de adresare este specificat de câmpul codului operației instrucțiunii și nu printr-un câmp separat al modului de adresare. De aceea, modul de adresare pentru o anumită instrucțiune este fix și nu poate fi modificat.

Instrucțiunile care au formatul cu trei registre utilizează adresarea registrelor. Adresarea indirectă prin registru este utilizată doar de instrucțiunile LD și ST, singurele instrucțiuni care accesează memoria de date. Instrucțiunile care au formatul cu două registre utilizează adresarea imediată, valoarea imediată fiind disponibilă în câmpul IM al instrucțiunii. Adresarea relativă este utilizată de instrucțiunile de salt și de instrucțiunea de apel al unei proceduri, astfel încât adresele generate prin acest mod de adresare se referă doar la memoria de instrucțiuni.

Atunci când într-un program trebuie să se utilizeze un mod de adresare care nu este disponibil, cum este, de exemplu, adresarea indexată, este necesară utilizarea unei secvențe de instrucțiuni pentru implementarea modului de adresare respectiv. De exemplu, presupunem că este necesară încărcarea unui operand în registrul R8 din memoria de date prin adresare indexată. Dacă registrul de bază este R7 și indexul este o valoare notată cu X, descrierea simbolică a acestui transfer este:

$$R8 \leftarrow M\,(R7 + X)$$

Acest transfer poate fi realizat prin execuția următoarelor două instrucțiuni:

```
ADDU   R10,R7,X
LD     R8,R10
```

Prima instrucțiune, ADDU, formează adresa operandului prin extensia indexului X cu zerouri în cele 16 poziții mai semnificative și adunarea rezultatului la registrul de bază R7. Indexul este considerat un deplasament pozitiv față de adresa de bază și de aceea se utilizează instrucțiunea de adunare fără semn. Adresa efectivă care se obține este depusă temporar în registrul R10. A doua instrucțiune, LD, utilizează apoi conținutul registrului R10 ca adresă a operandului și transferă operandul în registrul destinație R8.

2.3. Exemplu de program

În continuare se va prezenta un exemplu de program care utilizează instrucțiunile procesorului RISC. Programul efectuează înmulțirea a două numere fără semn de câte 16 biți și obține un produs de 32 de biți. Pentru înmulțire se utilizează prima versiune a algoritmului, cea în care deînmulțitul se deplasează la stânga cu o poziție în fiecare pas al operației (spre deosebire de versiunea finală a algoritmului, în care produsul se deplasează la dreapta). Pentru versiunea utilizată, registrele trebuie sa aibă o lungime dublă față de cea a operanzilor, astfel încât pentru cei doi operanzi și pentru produs se poate utiliza câte un registru de 32 de biți al procesorului. Se utilizează un registru suplimentar pentru păstrarea contorului de iterații. La începutul programului, registrul R0 se inițializează cu zero pentru a permite încărcarea valorilor imediate care reprezintă operanzii și contorul de iterații prin adunări imediate.

```
; Înmulţire de 16 x 16 biţi, produs de 32 de biţi
; R1 – Deînmulţit (45 = 0x2D)
; R2 – Înmulţitor (36 = 0x24)
; R3 – Produs
; R4 – Contor de iteraţii
MULT:
       XOR    R0,R0,R0            ; R0 = 0
       ADDI   R1,R0,45            ; încarcă deînmulţitul (D)
       ADDI   R2,R0,36            ; încarcă înmulţitorul (I)
       MOVA   R3,R0               ; iniţializează produsul (P) cu 0
       ADDI   R4,R0,16            ; iniţializează contorul (C) cu 16
TESTQ0:
       ANDI   R5,R2,1             ; testează bitul 0 al I
       BZ     R5,BITZERO          ; salt dacă bitul este zero
       ADD    R3,R3,R1            ; adună D la P
BITZERO:
       SHL    R1,R1,1             ; deplasează D la stânga
       SHR    R2,R2,1             ; deplasează I la dreapta
       SUBI   R4,R4,1             ; decrementează C
       BNZ    R4,TESTQ0           ; repetă dacă C /= 0
       MOVA   R0,R3               ; copiază rezultatul în registrul R0
       HALT
```

3. Schema bloc detaliată a procesorului RISC

3.1. Modificări faţă de schema bloc simplificată

Figura 3 prezintă schema bloc detaliată a procesorului RISC care va fi implementat. MI reprezintă memoria de instrucţiuni, RI reprezintă registrul de instrucţiuni, DCDI este decodificatorul de instrucţiuni, R este setul de registre, iar MD reprezintă memoria de date. În această schemă există câteva modificări ale căii de date din schema bloc simplificată, modificări care vor fi descrise în continuare. Prima modificare constă în introducerea registrelor între etajele *pipeline* pentru transferul conţinutului contorului de program PC din etajul IF către etajele următoare. Aceste registre sunt denumite PCIF, PCID şi PCEX, numele registrului conţinând acronimul etajului *pipeline* la ieşirea căruia este amplasat (IF, ID, respectiv EX). În etajul IF conţinutul registrului PC este incrementat cu sumatorul INCPC.

O altă modificare a căii de date este înlocuirea circuitului pentru extensia cu zero a constantelor cu o unitate pentru constante, CONST. Această unitate efectuează extensia cu zero dacă semnalul de comandă *SelC* este 0 logic şi extensia cu semn dacă semnalul *SelC* este 1 logic. Registrul REXT de la ieşirea circuitului pentru extensia cu zero din schema bloc simplificată a fost redenumit RCONST. De asemenea, a fost adăugat multiplexorul MUXA pentru a permite memorarea într-un registru din setul de registre a adresei instrucţiunii următoare, care este disponibilă în registrul PCID. Această memorare este necesară pentru implementarea instrucţiunii de apel al unei proceduri, JMPL. Semnalul de selecţie al multiplexorului MUXA este *MxA*.

Unitatea aritmetică şi logică UAL este modificată pentru a permite ca operaţiile de deplasare logică să se efectueze cu mai multe poziţii în locul deplasării cu o singură poziţie. Aceasta presupune utilizarea unui circuit combinaţional de deplasare cu poziţii multiple; acest circuit este unul mai complex şi va fi descris în secţiunea 3.2 care prezintă detalii despre unitatea aritmetică şi logică. Numărul de poziţii cu care se realizează deplasarea poate fi între 0 şi 31; acest număr este transmis modulului UAL prin semnalul *Sh*. Acest semnal de 5 biţi provine din cele 5 poziţii mai puţin semnificative ale registrului de instrucţiuni RI, semnalul respectiv fiind notat cu *Shn*. Deoarece registrul de instrucţiuni RI se află la ieşirea etajului IF, iar modulul UAL se află în etajul de execuţie EX, semnalul *Shn* este transferat în etajul EX prin intermediul registrului *pipeline* RID de la ieşirea etajului ID.

Pentru implementarea instrucţiunilor SGTE şi SLT s-a adăugat o logică suplimentară care permite încărcarea valorii 1 sau 0 în registrul destinaţie specificat în aceste instrucţiuni, în funcţie de anumite condiţii. De exemplu, în cazul instrucţiunii SLT în registrul destinaţie se încarcă valoarea 1 dacă R(RSA) – R(RSB) < 0 şi valoarea 0 în caz contrar. Această condiţie

poate fi evaluată utilizând indicatorul de stare *N*, care este setat dacă un rezultat este negativ, și indicatorul *V*, care este setat dacă apare o depășire la o operație aritmetică. Condiția R(RSA) – R(RSB) < 0 este adevărată dacă *N* este 1 și *V* este 0 (deci, rezultatul scăderii este un număr negativ și nu apare depășire) sau dacă *N* este 0 și *V* este 1 (deci, rezultatul scăderii este un număr pozitiv și apare depășire). Această condiție poate fi evaluată în mod simplu printr-o poartă SAU EXCLUSIV la intrările căreia se aplică indicatorii *N* și *V*. În cazul instrucțiunii SGTE, condiția este inversată. Semnalul $N \oplus V$ și complementul acestuia se înscriu în registrul CCEX de 2 biți de la ieșirea etajului *pipeline* EX. Cele două ieșiri ale acestui registru sunt extinse la stânga cu 31 de biți de 0 și sunt aplicate la două intrări suplimentare ale multiplexorului MUXD. Semnalul de selecție *MxD* pentru multiplexorul MUXD este extins la 2 biți.

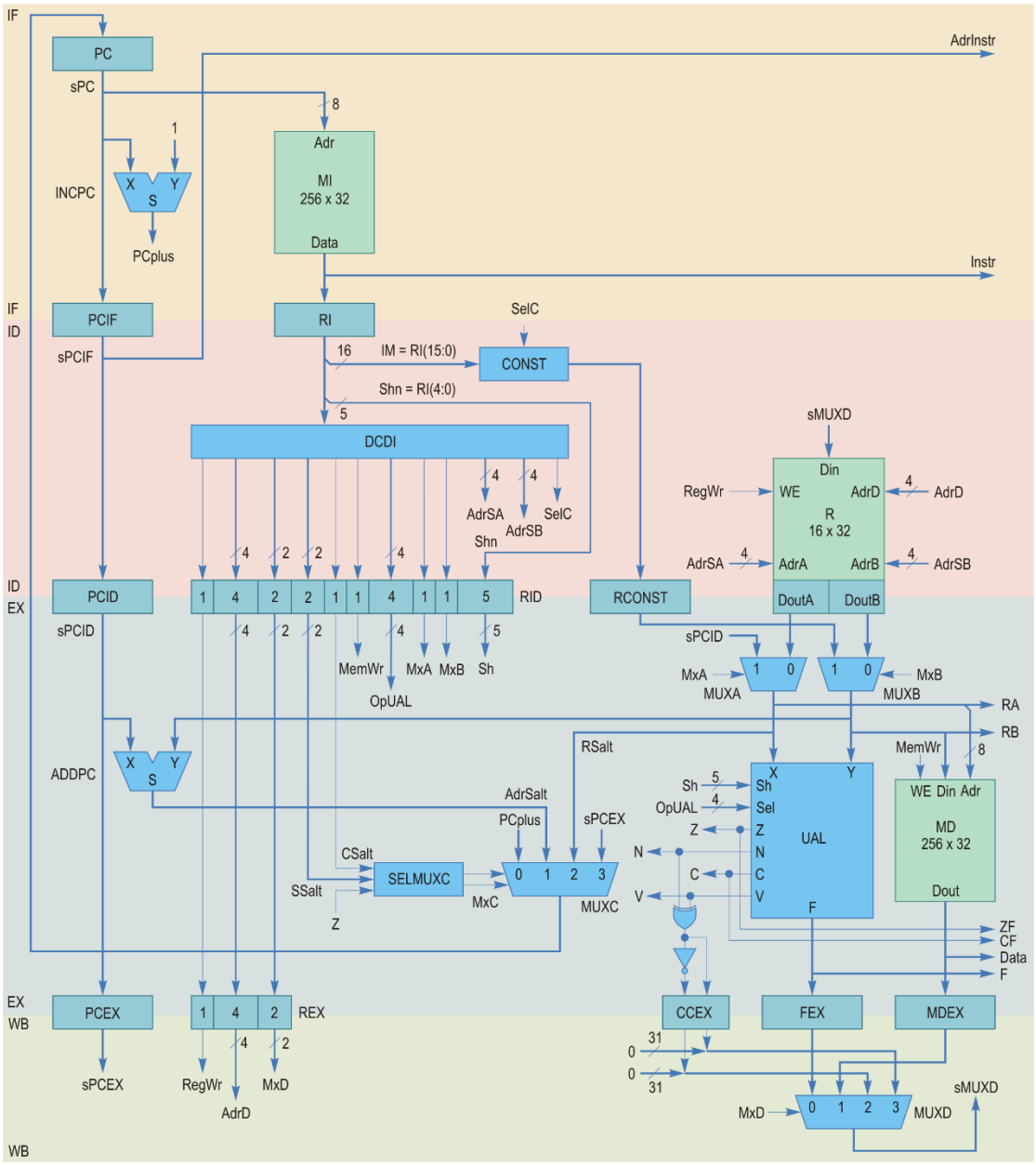

Figura 3. Schema bloc detaliată a procesorului RISC.

Un ultim detaliu se referă la logica de control pentru registrul PC. Această logică permite încărcarea adreselor în registrul PC pentru implementarea instrucțiunilor de salt. Multiplexorul MUXC permite selecția unei adrese dintr-un număr de patru adrese care pot reprezenta adresa următoarei instrucțiuni executate. *PCplus* reprezintă valoarea incrementată a contorului de program și se utilizează pentru execuția secvențială a instrucțiunilor. *AdrSalt* se

utilizează de către instrucțiunile de salt și se obține prin adunarea deplasamentului adresei de destinație din câmpul OFFS la valoarea incrementată a contorului de program. *RSalt* se utilizează de către instrucțiunea JMPR și reprezintă adresa la care se efectuează saltul, adresă conținută în registrul specificat în instrucțiune. Ultima adresă aplicată la intrarea multiplexorului MUXC este *sPCEX*, care reprezintă conținutul curent al contorului de program PC; această adresă se utilizează numai de către instrucțiunea de oprire HALT, care este implementată printr-un salt la aceeași adresă la care se află această instrucțiune. Pentru selecția uneia din cele patru adrese de la intrarea multiplexorului MUXC se utilizează semnalul *MxC* de 2 biți; acest semnal este generat de către modulul SELMUXC. Intrările acestui modul sunt două semnale de comandă suplimentare generate de către decodificatorul de instrucțiuni, *SSalt* și *CSalt*, ca și indicatorul de stare *Z*. Detalii despre modul în care se generează semnalul de selecție *MxC* sunt prezentate în secțiunea 3.3 care descrie unitatea de control.

Observație

- Este important ca elementele căii de date care execută diferite operații să fie amplasate în etajul *pipeline* corect. De exemplu, sumatorul ADDPC pentru calculul adresei de salt este amplasat în etajul EX. Ieșirea acestui sumator este conectată la una din intrările multiplexorului MUXC. Acest multiplexor și logica sa de selecție sunt amplasate în același etaj EX, pentru că selecția adresei cu care se va încărca registrul PC în etajul IF se realizează pe baza valorilor calculate în etajul de execuție EX.

3.2. Unitatea aritmetică și logică

Unitatea aritmetică și logică (UAL) este un circuit combinațional care execută operații aritmetice de adunare și scădere, operații logice de bază (ȘI, SAU, SAU EXCLUSIV, complement) și operații de deplasare logică. Setul de instrucțiuni nu necesită execuția operațiilor de înmulțire și împărțire. Schema bloc a unității aritmetice și logice este ilustrată în figura 4.

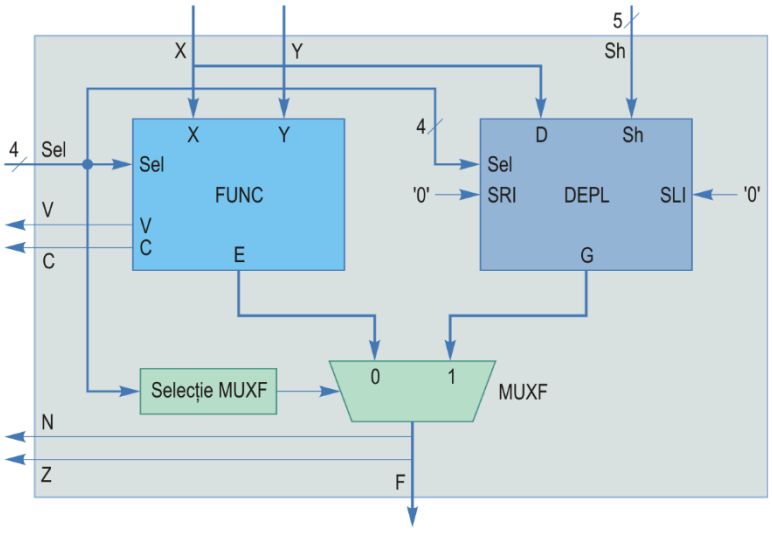

Figura 4. Schema bloc a unității aritmetice și logice.

La intrările *X* și *Y* ale unității aritmetice și logice se aplică doi operanzi asupra cărora trebuie efectuată operația. La intrarea de selecție *Sel* se aplică codul de 4 biți al operației de executat. Pentru o operație de deplasare logică la dreapta sau la stânga, la intrarea *Sh* se aplică o valoare de 5 biți care indică numărul de poziții cu care trebuie efectuată deplasarea. Rezultatul operației executate se obține la ieșirea *F*. În funcție de rezultatul operației sunt setați indicatorii de stare *Z*, *N*, *C* și *V*.

Indicatorul *Z* (*zero*) este setat la 1 logic dacă rezultatul unei operații aritmetice sau logice este zero și la 0 logic în caz contrar. Indicatorul *N* (*negative*) este setat la valoarea bitu-

lui cel mai semnificativ al ieșirii UAL, care este bitul de semn al rezultatului pentru operații cu numere cu semn. Dacă operanzii sunt considerați numere fără semn, indicatorul *C* (*carry*) este setat la 1 logic dacă apare un transport (depășire) la o operație de adunare sau dacă nu este necesară o corecție la o operație de scădere. De asemenea, pentru operanzi considerați numere fără semn, indicatorul *C* este setat la 0 logic dacă nu apare un transport (depășire) la o operație de adunare sau dacă este necesară o corecție la o operație de scădere. Corecția se realizează prin complementarea față de 2 a rezultatului și adăugarea semnului negativ. Dacă operanzii sunt considerați numere cu semn, indicatorul *V* (*overflow*) este setat la 1 logic dacă apare o depășire aritmetică la o operație de adunare sau scădere (bitul cel mai semnificativ nu poate fi reprezentat). Indicatorul *V* este setat la 0 logic dacă după o operație de adunare sau scădere nu apare o depășire aritmetică și rezultatul este corect.

Circuitul UAL este împărțit în două module, FUNC și DEPL. Modulul FUNC implementează toate operațiile aritmetice și logice, cu excepția operațiilor de deplasare. Acest modul setează indicatorii de stare *C* și *V*; indicatorii *C* și *V* nu sunt setați de către operațiile de deplasare. Modulul DEPL implementează operațiile de deplasare logică la dreapta și la stânga ale operandului aplicat la intrarea *X* a circuitului UAL. Multiplexorul MUXF selectează ieșirea modulului FUNC sau ieșirea modulului DEPL pentru a fi transmisă la ieșirea *F* a circuitului UAL. Indicatorii *Z* și *N* sunt setați pentru fiecare operație conform cu rezultatul operației respective.

Deoarece toate instrucțiunile trebuie executate într-un singur ciclu de ceas, pentru deplasare nu se utilizează un registru de deplasare, ci un circuit combinațional de deplasare. În implementarea curentă, deplasarea se realizează cu o singură poziție, indiferent de valoarea specificată la intrarea *Sh*, cu excepția cazului când această valoare este zero și când operandul de la intrare este transmis nemodificat la ieșire. Aplicația 4.5 are ca scop modificarea circuitului de deplasare pentru a permite deplasarea cu un număr de poziții cuprins între 0 și 31. Implementarea este realizată cu multiplexoare 4:1, pentru fiecare poziție binară fiind utilizat un multiplexor. Intrările seriale *SRI* și *SLI* nu sunt utilizate pentru instrucțiunile implementate, care sunt instrucțiuni de deplasare logică. Aceste intrări ar putea fi utilizate însă pentru implementarea unor instrucțiuni de deplasare aritmetică sau de rotire.

Tabelul 3 prezintă operațiile executate de circuitul UAL și codurile de selecție pentru aceste operații. Operațiile *F*= *X* și *F* = *Y* sunt operații de transfer, prin care operandul de la intrarea *X*, respectiv *Y* a circuitului este transmis nemodificat la ieșire.

Tabelul 3. Operații executate de unitatea aritmetică și logică.

Operație	Cod de selecție
F = *X*	0000
F = *X* + *Y*	0010
F = *X* – *Y*	0101
F = *X* **and** *Y*	1000
F = *X* **or** *Y*	1001
F = *X* **xor** *Y*	1010
F = **not** *X*	1011
F = *Y*	1100
F = **shr** *X*	1101
F = **shl** *X*	1110

În continuare se descrie circuitul combinațional care permite deplasarea logică la stânga sau la dreapta cu un număr de poziții cuprins între 0 și 31. O schemă bloc a acestui circuit este prezentată în figura 5. Intrarea de date este un operand de 32 de biți *D*, iar ieșirea este rezultatul de 32 de biți *G*. Semnalul de comandă *Left*, care este decodificat din codul de selecție al operației aritmetice și logice, selectează deplasarea la stânga prin valoarea logică 1 și la dreapta prin valoarea logică 0. Intrarea *Sh* de 5 biți specifică numărul de poziții cu care trebuie efectuată deplasarea, între 0 și 31.

O deplasare logică cu un număr de poziții *p* implică inserarea unui număr de *p* biți de zero în rezultat. Pentru a simplifica structura circuitului de deplasare, atât deplasarea la stânga cât și deplasarea la dreapta vor fi efectuate prin operații de rotire la dreapta. Intrarea operației de rotire va fi valoarea de 64 de biți obținută din data de intrare *D* concatenată la stânga cu 32 de biți de zero. O deplasare la dreapta va fi efectuată prin rotirea intrării cu *p* poziții la dreapta; o deplasare la stânga va fi efectuată prin rotirea intrării cu 64 – *p* poziții la dreapta. Acest număr de poziții se poate obține prin complementarea față de 2 a valorii de 6 biți '0' & *Sh*. Modulul SELC2 efectuează complementarea față de 2 în mod selectiv. Astfel, dacă intrarea *Left* este 0 logic (indicând deplasarea la dreapta), circuitul transmite la ieșire intrarea de 6 biți '0' & *Sh* nemodificată. Dacă intrarea *Left* este 1 logic (indicând deplasarea la stânga), circuitul transmite la ieșire complementul față de 2 al valorii '0' & *Sh* de la intrare.

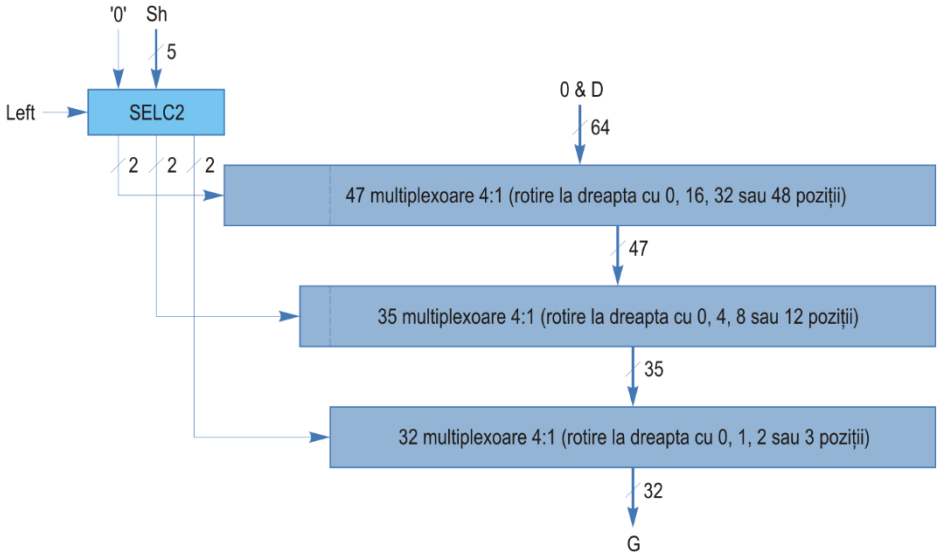

Figura 5. Schema bloc a circuitului combinațional de deplasare de 32 de biți.

Cele 63 de rotiri posibile se pot obține utilizând trei nivele de multiplexoare 4:1, după cum se arată în figura 5. Primul nivel efectuează rotirea cu 0, 16, 32 sau 48 de poziții, al doilea nivel cu 0, 4, 8 sau 12 poziții, iar al treilea nivel cu 0, 1, 2 sau 3 poziții. Numărul de poziții cu care se realizează rotirea de către fiecare nivel de multiplexoare poate fi controlat de grupele de câte doi biți ai valorii de la ieșirea modulului SELC2. Cei doi biți mai semnificativi controlează primul nivel, următorii doi biți controlează al doilea nivel, iar cei doi biți mai puțin semnificativi controlează al treilea nivel. De exemplu, în funcție de valoarea celor doi biți mai semnificativi, în primul nivel se va selecta rotirea cu următoarele numere de poziții: pentru 00, cu 0 poziții; pentru 01, cu 16 poziții; pentru 10, cu 32 de poziții; pentru 11, cu 48 de poziții.

Pentru exemplificare presupunem că este necesară deplasarea la stânga cu 6 poziții, operație echivalentă cu rotirea la dreapta cu 64 – 6 = 58 de poziții. Acest număr de poziții se poate obține prin complementarea față de 2 a valorii 6 reprezentată pe 6 biți (000110); valoarea binară a complementului față de 2 este 111010 (echivalentă cu valoarea 58). Cei doi biți mai semnificativi (11) ai acestei valori determină rotirea cu 48 de poziții în primul nivel, următorii doi biți (10) determină rotirea cu 8 poziții în al doilea nivel, iar ultimii doi biți (10) determină rotirea cu 2 poziții în al treilea nivel, pentru un număr total de 48 + 8 + 2 = 58 de poziții.

Datorită faptului că intrarea de 64 de biți conține 32 de biți de zero, în fiecare nivel se poate utiliza un număr de multiplexoare care este mai mic decât 64. În general, un anumit nivel necesită un număr de multiplexoare care este 32 (numărul de biți ai datei de intrare care nu a fost extinsă cu zerouri) plus numărul total de poziții cu care ieșirea nivelului poate fi deplasată de nivelele următoare. Ieșirea primului nivel poate fi deplasată cu cel mult 63 – 48 = 15 poziții la dreapta. De aceea, acest nivel necesită 32 + 15 = 47 de multiplexoare. Ieșirea ce-

lui de-al doilea nivel poate fi deplasată încă cu cel mult 63 – (48 + 12) = 63 – 60 = 3 poziții, rezultând 32 + 3 = 35 de multiplexoare pentru acest nivel. Ieșirea ultimului nivel nu mai poate fi deplasată suplimentar, astfel încât al treilea nivel necesită doar 32 de multiplexoare.

3.3. Unitatea de control

Circuitul cel mai important al unității de control este decodificatorul de instrucțiuni. Acesta este un circuit combinațional care generează toate semnalele de comandă necesare funcționării procesorului pe baza diferitelor câmpuri ale instrucțiunii aflate în registrul de instrucțiuni RI. O parte a semnalelor de comandă se pot obține direct din câmpurile instrucțiunii. Astfel, semnalele *AdrD*, *AdrSA* și *AdrSB* se pot obține din câmpurile RD, RSA, respectiv RSB ale instrucțiunii, care conțin adresa registrului destinație, adresa registrului sursă A, respectiv adresa registrului sursă B. De asemenea, semnalul *Shn* se poate obține din biții 4..0 ai câmpului valorii imediate IM. Acest semnal este păstrat în registrul RID de la ieșirea etajului *pipeline* ID, fiind utilizat în etajul EX de instrucțiunile de deplasare pentru a indica numărul de poziții cu care se realizează deplasarea; semnalul corespunzător din etajul EX este denumit *Sh*.

Pentru o decodificare mai simplă a instrucțiunilor, codurile de operații ale acestora sunt alese astfel încât cei patru biți mai puțin semnificativi ai codurilor să corespundă cu codul de selecție al operației respective executate de UAL ori de câte ori acest cod este utilizat. Astfel, semnalul *OpUAL* care este aplicat la intrarea de selecție a operației executate de UAL poate fi extras direct din cei patru biți mai puțin semnificativi ai câmpului codului operației CODOP.

Pentru încărcarea contorului de program PC cu adresa următoarei instrucțiuni care trebuie executată se utilizează multiplexorul MUXC. Intrările de date ale acestui multiplexor sunt cele patru adrese din care se va selecta adresa următoarei instrucțiuni executate, după cum s-a prezentat în secțiunea 3.1. Semnalul de selecție *MxC* de doi biți al acestui multiplexor este generat de către un modul suplimentar numit SELMUXC, deoarece pentru instrucțiunile de salt condiționat BZ și BNZ selecția depinde și de indicatorul de stare *Z*. Principalul semnal utilizat pentru selecția adresei următoarei instrucțiuni este semnalul *SSalt* de 2 biți. În cazul instrucțiunilor BZ și BNZ se utilizează un semnal suplimentar de selecție, *CSalt*. Același semnal *CSalt* este utilizat și pentru diferențierea între instrucțiunile JMP și JMPL, pe de o parte, și instrucțiunea HALT, pe de altă parte. Modul de selecție al adresei următoarei instrucțiuni este sintetizat în tabelul 4, care indică, pentru combinațiile semnalelor *SSalt* și *CSalt*, adresa care se va încărca în registrul PC și semnalul de selecție corespunzător *MxC*. Reamintim că *AdrSalt* reprezintă suma dintre deplasamentul adresei de destinație din câmpul OFFS și conținutul incrementat al registrului PC, iar *RSalt* este adresa de salt din registrul specificat în instrucțiunea JMPR.

Tabelul 4. Generarea semnalului de selecție *MxC* de către modulul SELMUXC.

SSalt	CSalt	Z	Operație	MxC	Explicație
0 0	X	X	PC ← PC + 1	0 0	Execuție secvențială
0 1	0	0	PC ← PC + 1	0 0	Instrucțiunea BZ cu *Z* = 0
0 1	0	1	PC ← *AdrSalt*	0 1	Instrucțiunea BZ cu *Z* = 1
0 1	1	0	PC ← *AdrSalt*	0 1	Instrucțiunea BNZ cu *Z* = 0
0 1	1	1	PC ← PC + 1	0 0	Instrucțiunea BNZ cu *Z* = 1
1 0	X	X	PC ← *RSalt*	1 0	Instrucțiunea JMPR
1 1	0	X	PC ← *AdrSalt*	0 1	Instrucțiunile JMP, JMPL
1 1	1	X	PC ← PC	1 1	Instrucțiunea HALT

În tabelul 5 se indică, pentru fiecare instrucțiune, valorile semnalelor de comandă, cu excepția semnalelor care reprezintă adresele registrelor și care sunt preluate direct din câmpurile instrucțiunii. Acest tabel poate fi utilizat pentru implementarea decodificatorului de instrucțiuni.

Tabelul 5. Starea semnalelor de comandă pentru instrucțiunile procesorului RISC.

Mnemonica	CODOP	OpUAL	RegWr	MxD	SSalt	CSalt	MemWr	MxA	MxB	SelC
NOP	0000 0000	XXXX	0	XX	00	X	0	X	X	X
MOVA	0100 0000	0000	1	00	00	X	0	0	X	X
ADD	0000 0010	0010	1	00	00	X	0	0	0	X
SUB	0000 0101	0101	1	00	00	X	0	0	0	X
AND	0000 1000	1000	1	00	00	X	0	0	0	X
OR	0000 1001	1001	1	00	00	X	0	0	0	X
XOR	0000 1010	1010	1	00	00	X	0	0	0	X
NOT	0000 1011	1011	1	00	00	X	0	0	X	X
ADDI	0010 0010	0010	1	00	00	X	0	0	1	1
SUBI	0010 0101	0101	1	00	00	X	0	0	1	1
ANDI	0010 1000	1000	1	00	00	X	0	0	1	0
ORI	0010 1001	1001	1	00	00	X	0	0	1	0
XORI	0010 1010	1010	1	00	00	X	0	0	1	0
ADDU	0100 0010	0010	1	00	00	X	0	0	1	0
SUBU	0100 0101	0101	1	00	00	X	0	0	1	0
MOVB	0000 1100	1100	1	00	00	X	0	X	0	X
SHR	0000 1101	1101	1	00	00	X	0	0	X	X
SHL	0000 1110	1110	1	00	00	X	0	0	X	X
LD	0001 0000									
ST	0010 0000									
JMPR	0111 0000	XXXX	0	XX	10	X	0	0	X	X
SGTE	0111 0101									
SLT	0110 0101									
BZ	0110 0000	0000	0	XX	01	0	0	0	1	1
BNZ	0101 0000	0000	0	XX	01	1	0	0	1	1
JMP	0110 1000	XXXX	0	XX	11	0	0	X	1	1
JMPL	0011 0000									
HALT	0110 1001	XXXX	0	XX	11	1	0	X	X	X

Observație

- O parte a liniilor din tabelul 5 nu sunt completate în mod intenționat, completarea lor fiind subiectul aplicației 5.4.

4. Eliminarea hazardurilor de control

Hazardurile de control apar în situațiile în care anumite instrucțiuni, cum sunt cele de salt, modifică ordinea secvențială de execuție a instrucțiunilor dintr-un program. Ori de câte ori este executată o instrucțiune de salt, trebuie să se încarce o nouă adresă în contorul de program, ceea ce necesită invalidarea instrucțiunilor a căror execuție a început deja. Instrucțiunile de salt condiționat sunt mai dificil de gestionat față de alte instrucțiuni de salt, deoarece pentru aceste instrucțiuni se va cunoaște doar în timpul execuției dacă o condiție de salt este îndeplinită, deci dacă saltul va fi executat sau nu.

Considerăm următoarea secvență conținând instrucțiuni ale procesorului RISC; la începutul fiecărei linii se indică adresa la care se află instrucțiunea respectivă:

```
00      BZ      R5,06
01      MOVA    R1,R2
02      MOVA    R3,R4
03      ADD     R3,R3,R1
        ...
07      SHL     R1,R1,1
```

Diagrama de execuție a acestei secvențe de instrucțiuni este ilustrată în figura 6(a). Dacă registrul R5 conține valoarea zero, se va executa saltul la instrucțiunea de la adresa 07 (adresă obținută prin adunarea adresei relative 06 specificată în instrucțiune la conținutul incrementat al contorului de program). În caz contrar, continuă execuția secvențială a instrucțiunilor. Conținutul registrului R5 este evaluat în etajul de execuție EX, în ciclul 3 al primei instrucțiuni din figură. Presupunând că registrul R5 conține valoarea zero, deci saltul va fi executat, în același ciclu 3 se aplică la intrarea registrului PC adresa la care trebuie efectuat saltul. Registrul PC va fi înscris cu noua adresă în ciclul 4, dar în acest ciclu se va extrage încă instrucțiunea de la adresa 03 aflată în registrul PC la începutul ciclului. Abia în ciclul 5 se va extrage instrucțiunea de la adresa de salt 07. Însă, în acest ciclu instrucțiunile de la adresele 01, 02 și 03 se află deja în diferite faze de execuție. Deci, aceste instrucțiuni vor fi executate chiar dacă în program se specifică faptul că ele nu trebuie executate atunci când condiția de salt este îndeplinită.

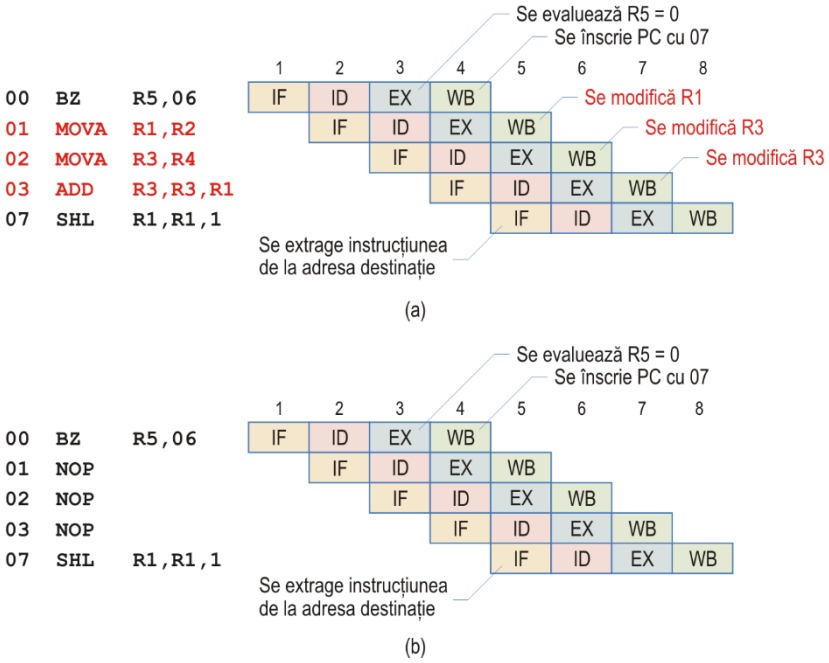

Figura 6. Exemplu de hazard de control: (a) ilustrarea problemei hazardului de control; (b) eliminarea hazardului de control prin inserarea unor instrucțiuni NOP.

După cum s-a descris în secțiunea 1, hazardurile de control pot fi eliminate prin metoda software de întârziere a salturilor, care constă în inserarea după o instrucțiune de salt a unor instrucțiuni NOP sau a unor instrucțiuni care pot fi executate indiferent dacă saltul va fi executat sau nu. Figura 6(b) ilustrează modificarea secvenței de program prezentate anterior pentru a evita hazardul de control. Sunt inserate trei instrucțiuni NOP după instrucțiunea de salt condiționat BZ, instrucțiuni care pot fi lansate în execuție indiferent dacă saltul va fi executat sau nu, deoarece ele nu vor afecta corectitudinea programului. Dacă se utilizează această metodă, timpul necesar execuției unui program va crește cu trei cicluri de ceas pentru fiecare instrucțiune de salt, indiferent dacă saltul va fi executat sau nu. Totuși, în unele cazuri este posibilă rearanjarea instrucțiunilor astfel încât să se execute instrucțiuni utile după instrucțiunile de salt.

O metodă hardware care se poate utiliza pentru eliminarea hazardurilor de control este *predicția salturilor*. O formă simplă de predicție a salturilor este de a presupune că salturile condiționate nu vor fi executate niciodată. În acest caz, după întâlnirea unei instrucțiuni de salt condiționat se continuă extragerea, decodificarea și execuția secvențială a instrucțiunilor următoare, până când se va cunoaște dacă saltul specificat de respectiva instrucțiune va fi executat sau nu. Dacă saltul nu va fi executat, se continuă execuția normală a instrucțiunilor din etajele *pipeline*. Dacă saltul va fi executat, un anumit număr de instrucțiuni de după in-

strucțiunea de salt (în cazul procesorului RISC prezentat, un număr de trei instrucțiuni) trebuie anulate, astfel încât operațiile din etajele EX și WB să nu fie executate. Aceasta va avea același efect ca și inserarea unor instrucțiuni NOP după instrucțiunea de salt. Totuși, timpul necesar execuției nu va crește dacă saltul nu va fi executat, deoarece nu trebuie adăugate instrucțiuni NOP în program.

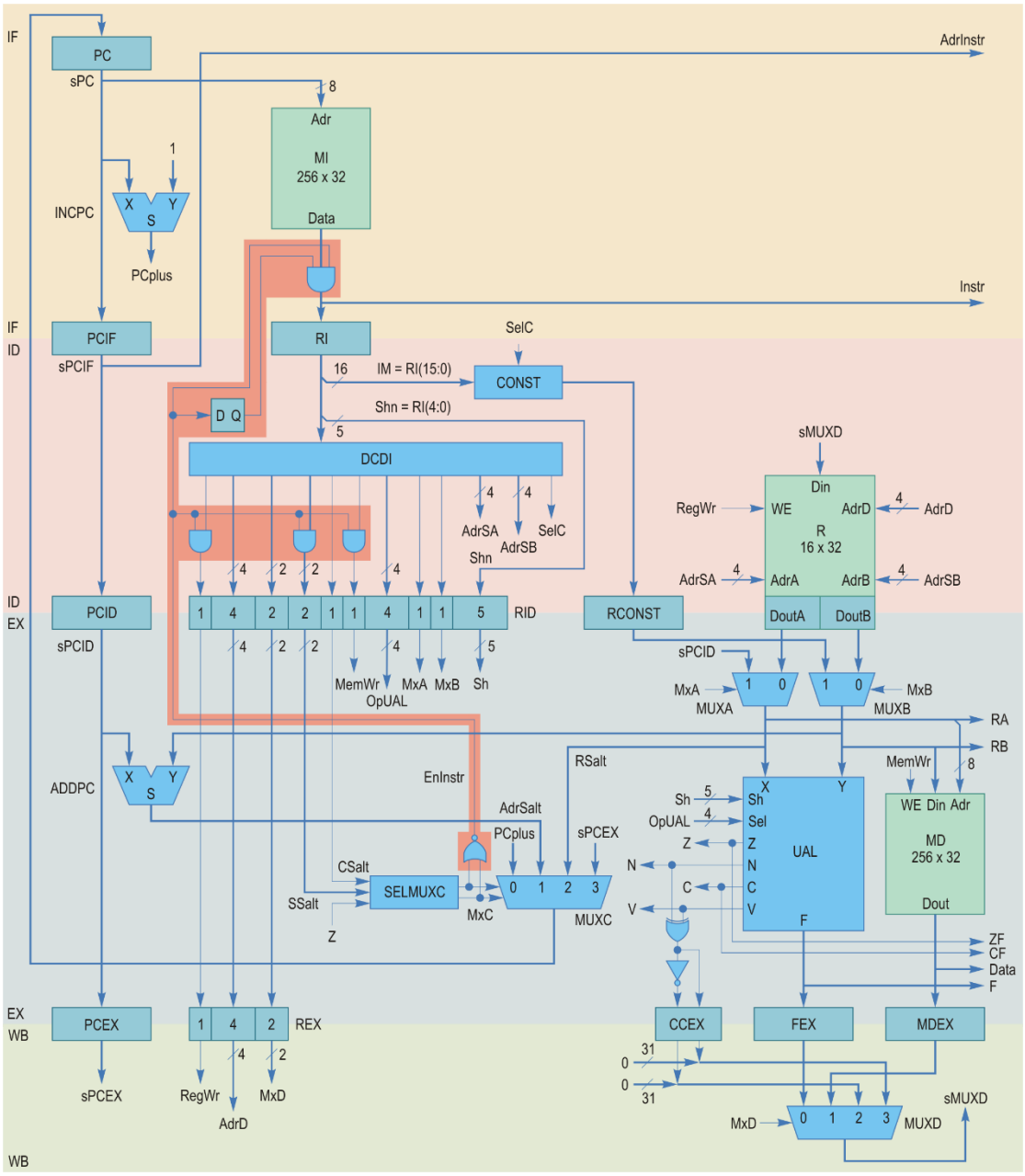

Figura 7. Schema bloc detaliată a procesorului RISC, completată cu logica pentru eliminarea hazardurilor de control.

Pentru utilizarea acestei forme de predicție a salturilor la procesorul RISC, trebuie să se determine cazurile în care un salt condiționat este executat. Aceste cazuri pot fi determinate pe baza semnalului de selecție *MxC* al multiplexorului MUXC, multiplexor care se utilizează pentru încărcarea contorului de program PC cu adresa următoarei instrucțiuni care trebuie executată. Din tabelul 4 se poate observa că, pentru cazurile în care contorul de program PC se încarcă cu o adresă diferită de cea a instrucțiunii următoare, vectorul *MxC* are valorile "01", "10" sau "11". Aceste cazuri corespund execuției unor instrucțiuni de salt, incluzând aici și instrucțiunea HALT, care este implementată printr-o instrucțiune de salt. Deci, o instrucțiune de salt va fi executată pentru toate valorile diferite de "00" ale vectorului *MxC*. Printr-o sim-

plă operație SAU logic între cele două linii ale vectorului *MxC* se poate obține un semnal care va fi activ atunci când un salt condiționat va fi executat. Dacă se inversează acest semnal, se obține un semnal *EnInstr* care poate fi utilizat pentru anularea unor instrucțiuni care nu trebuie executate.

Figura 7 prezintă schema bloc detaliată a procesorului RISC, în care se pune în evidență logica suplimentară necesară pentru eliminarea hazardurilor de control prin metoda predicției salturilor. Pentru anularea instrucțiunii de după o instrucțiune de salt, se utilizează porți ȘI pentru combinarea semnalului *EnInstr* cu semnalele *RegWr* și *MemWr* de la ieșirea decodificatorului de instrucțiuni DCDI, semnalele rezultate fiind înscrise în registrul RID. Astfel, atunci când semnalul *EnInstr* este dezactivat, se vor dezactiva și cele două semnale, ceea ce va preveni înscrierea registrelor și a memoriei de date. Pentru cazul în care după instrucțiunea de salt urmează o altă instrucțiune de salt, semnalul *EnInstr* este combinat și cu semnalul *SSalt* de la ieșirea decodificatorului de instrucțiuni, astfel încât semnalul *SSalt* va fi forțat la "00", indicând o execuție secvențială a instrucțiunilor.

Pentru anularea celei de-a doua instrucțiuni care urmează după o instrucțiune de salt, se utilizează alte porți ȘI pentru condiționarea datelor înscrise în registrul de instrucțiuni RI. Atunci când semnalul *EnInstr* este dezactivat, în registrul de instrucțiuni se va înscrie o valoare constând din toți biții egali cu 0, ceea ce corespunde codului instrucțiunii NOP. Astfel, atunci când se întâlnește o instrucțiune de salt pentru care saltul trebuie executat, instrucțiunea următoare acesteia este anulată prin înlocuirea codului acestei instrucțiuni cu codul instrucțiunii NOP.

Pentru anularea celei de-a treia instrucțiuni care urmează după o instrucțiune de salt, se utilizează un bistabil pentru a obține un semnal *EnInstr* întârziat cu un ciclu de ceas. Datele înscrise în registrul de instrucțiuni sunt condiționate și de acest semnal, astfel încât dacă semnalul *EnInstr* a fost dezactivat, semnalul de la ieșirea bistabilului va fi dezactivat în următorul ciclu de ceas, codul celei de-a treia instrucțiuni fiind înlocuit cu codul instrucțiunii NOP.

Observație

- În această lucrare de laborator nu sunt propuse modificări ale structurii procesorului RISC pentru a rezolva problema hazardurilor de date. Se presupune că hazardurile de date sunt eliminate prin metode software.

5. Aplicații

5.1. Descrieți în limbajul VHDL procesorul RISC prezentat în lucrarea de laborator. Pentru aceasta parcurgeți etapele descrise în continuare.

1. Creați un nou proiect pentru placa de dezvoltare Nexys4 DDR.
2. Extrageți într-un director temporar fișierele din arhiva proc_RISC.zip, disponibilă pe pagina laboratorului, și includeți aceste fișiere în proiect. Arhiva conține descrierea următoarelor module ale procesorului RISC: memoria de instrucțiuni MI; setul de registre R; unitatea aritmetică și logică UAL; un multiplexor 4:1 de un bit, utilizat de circuitul de deplasare al unității aritmetice și logice; memoria de date MD. Fișierul RISC_pkg.vhd conține un pachet cu definițiile codurilor de operații ale instrucțiunilor, definițiile codurilor de funcții pentru unitatea aritmetică și logică, ca și funcții de conversie care vor fi utilizate pentru transmiterea unor mesaje prin interfața serială UART. Fișierul proc_RISC.vhd conține modulul principal al procesorului RISC, în care sunt instanțiate entitățile modulelor MI, R, UAL și MD. De asemenea, sunt utilizate instrucțiuni concurente pentru implementarea sumatoarelor INCPC, ADDPC, a multiplexoarelor MUXA, MUXB, MUXC, MUXD și sunt conectate semnalele la porturile de ieșire ale procesorului.
3. Creați un modul VHDL pentru unitatea CONST, care realizează extensia intrării de 16 biți la o valoare de 32 de biți. Dacă semnalul de comandă *SelC* este 0 logic, se adaugă cifre de 0 în cele 16 poziții mai semnificative ale ieșirii. Dacă semnalul *SelC*

este 1 logic, în cele 16 poziții mai semnificative ale ieșirii se adaugă cifre de 0 dacă intrarea este un număr pozitiv sau cifre de 1 dacă intrarea este un număr negativ.

4. Creați un modul VHDL pentru unitatea SELMUXC, care generează semnalul de selecție de doi biți *MxC* necesar multiplexorului MUXC. Utilizați tabelul 4 pentru descrierea acestui modul.

5. Creați un modul VHDL pentru decodificatorul de instrucțiuni DCDI. Intrarea acestui modul este codul instrucțiunii de 32 de biți (biții 11 .. 0 nu vor fi utilizați). Ieșirile modulului sunt semnalele de comandă *RegWr*, *AdrD*, *MxD*, *SSalt*, *CSalt*, *MemWr*, *OpUAL*, *MxA*, *MxB*, *AdrSA*, *AdrSB* și *SelC*. Semnalele de comandă *AdrD*, *OpUAL*, *AdrSA* și *AdrSB* se pot extrage direct din codul instrucțiunii. Utilizați tabelul 5 pentru definirea celorlalte semnale de comandă, înlocuind valoarea indiferentă × cu valoarea binară 0. În prima etapă, decodificați doar următoarele instrucțiuni: NOP, MOVA, instrucțiuni aritmetice și logice cu registre (ADD, SUB, AND, OR, XOR, NOT, SHR, SHL), instrucțiuni aritmetice și logice cu valori imediate (ADDI, SUBI, ANDI, ORI, XORI), BZ, BNZ și HALT. Ulterior, se va completa modulul cu decodificarea restului instrucțiunilor (aplicația 5.4).

6. Includeți în proiect modulul FDN al registrului de *n* biți care a fost utilizat pentru circuitul de înmulțire secvențială. Inițializați genericul *n* cu valoarea 32, care este dimensiunea majorității registrelor necesare. Utilizați acest modul pentru instanțierea registrelor necesare procesorului RISC. Utilizați clauza `generic map` pentru instanțierea registrelor a căror dimensiune este diferită de 32 (RID, REX, CCEX). Deoarece registrele acestui procesor se vor încărca în fiecare ciclu de ceas, intrarea de validare CE a acestor registre trebuie conectată la 1 logic.

7. Completați modulul principal proc_RISC.vhd cu instanțierea restului entităților, conform schemei bloc detaliate din figura 7; definiți și inițializați semnalele suplimentare necesare. Pentru generarea porților ȘI ale logicii de eliminare a hazardurilor de control utilizați operatorul `and` și construcția `for .. generate`, iar pentru bistabilul necesar acestei logici scrieți un proces secvențial în același modul principal. Dacă semnalul de resetare *Rst* este '1', semnalul de ieșire al bistabilului trebuie setat la '1' pentru a valida încărcarea primei instrucțiuni în registrul de instrucțiuni RI (semnalul de ieșire al bistabilului trebuie inițializat cu '1' la declarare).

5.2. Deschideți fișierul sursă a memoriei de instrucțiuni MI. În acest fișier, memoria de instrucțiuni este inițializată cu codul exemplului de program din secțiunea 2.3. Urmăriți modul în care au fost adăugate instrucțiuni NOP pentru eliminarea hazardurilor de date.

Observație

- În codul din fișierul sursă al memoriei de instrucțiuni MI, au fost adăugate două instrucțiuni NOP după instrucțiunea BZ pentru a se evita anularea de către logica de eliminare a hazardului de control a două instrucțiuni de la adresa de destinație a saltului, adresă care este mai apropiată decât instrucțiunile care ar fi anulate.

Creați un fișier al bancului de test. În acest fișier, declarați și inițializați semnalele de intrare corespunzătoare porturilor de intrare ale procesorului, declarați semnalele de ieșire corespunzătoare porturilor de ieșire, instanțiați entitatea procesorului, scrieți un proces pentru generarea semnalului de ceas și scrieți un proces pentru generarea unui impuls de resetare pentru procesor. Simulați apoi funcționarea procesorului cu simulatorul Vivado, urmărind execuția programului în fiecare ciclu de ceas.

5.3. Implementați procesorul RISC pe placa de dezvoltare Nexys4 DDR și testați funcționarea acestuia. Pentru aceasta, extrageți într-un director temporar fișierele din arhiva implem_RISC.zip, disponibilă pe pagina laboratorului, și includeți aceste fișiere în proiectul creat pentru aplicația 5.1. Modulul filtru_buton din această arhivă conține un filtru pentru eliminarea oscilațiilor butonului utilizat pentru execuția pas cu pas a instrucțiunilor. Modulul

conv_uart realizează conversia ieșirilor procesorului în formatul ASCII necesar pentru afișarea lor în ecranul aplicației *HyperTerminal* prin interfața serială UART. Acest modul interpretează codul instrucțiunii și convertește diferitele câmpuri ale instrucțiunii într-un format apropiat de cel din limbajul de asamblare. Modulul uart_send64 transmite pe interfața serială UART o linie formată din 64 de caractere; acest modul utilizează modulul uart_tx al transmițătorului serial care a fost utilizat în lucrarea de laborator *Testarea și depanarea proiectelor VHDL*. În modulul afis_uart sunt instanțiate entitățile modulelor conv_uart și uart_send64. Arhiva mai conține modulul principal implem_RISC.vhd, în care sunt instanțiate entitățile procesorului, a filtrului pentru buton și a modulului de afișare, și fișierul de constrângeri pentru placa Nexys4DDR.

Generați șirul de biți pentru configurarea circuitului FPGA și configurați circuitul. Lansați în execuție aplicația *HyperTerminal* și creați o conexiune cu placa de dezvoltare; portul serial utilizat trebuie să fie diferit de COM1. Selectați **115200** pentru debitul binar, **8** biți de date, fără paritate și **1** bit de STOP; în câmpul *Flow control* selectați **None**. Urmăriți execuția pas cu pas a programului aflat în memoria de instrucțiuni. Butonul utilizat pentru generarea unui nou impuls de ceas este butonul BTNU. La fiecare apăsare a acestui buton, trebuie să se afișeze o linie conținând următoarele: adresa instrucțiunii curente; mnemonica instrucțiunii și celelalte câmpuri ale sale; caracterele 'F:' urmate de valoarea de la ieșirea *F* a modulului UAL; caracterele 'RA:' urmate de valoarea de la ieșirea *DoutA* a setului de registre; caracterele 'RB:' urmate de valoarea de la ieșirea *DoutB* a setului de registre.

5.4. Completați tabelul 5 cu valorile semnalelor de comandă necesare pentru instrucțiunile LD, ST, SGTE, SLT și JMPL.

5.5. Completați modulul decodificatorului de instrucțiuni cu decodificarea restului instrucțiunilor procesorului, care nu au fost decodificate în aplicația 5.1.

5.6. Modificați circuitul de deplasare DEPL din unitatea aritmetică și logică (modulul unit_aritm.vhd) pentru a realiza deplasarea combinațională cu un număr de poziții cuprins între 0 și 31. Desenați mai întâi schema de conectare a multiplexoarelor pentru a realiza rotirea la dreapta cu numărul corespunzător de poziții, conform figurii 5.