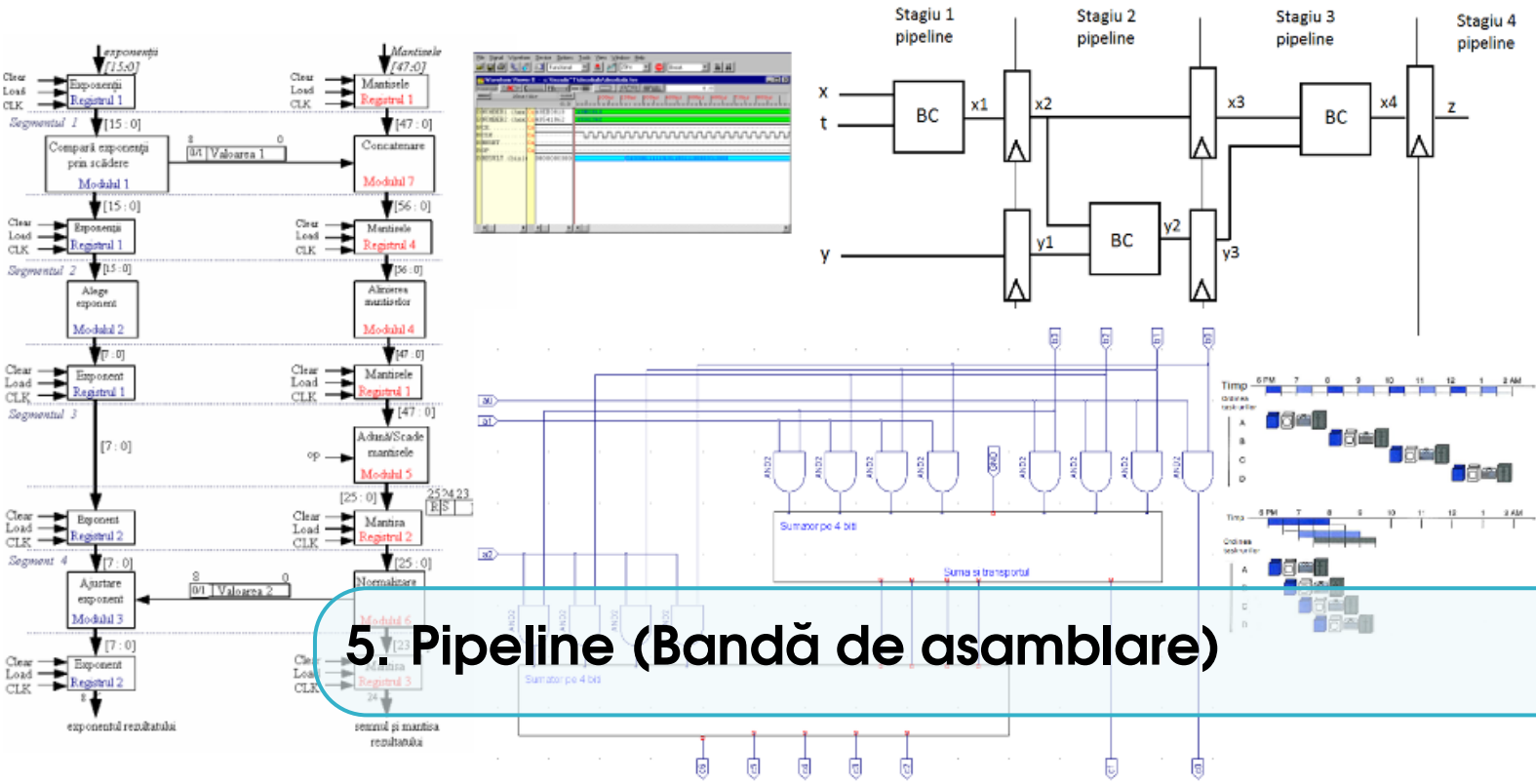

5. Pipeline (Bandă de asamblare)

5.1 Aspecte teoretice

Banda de asamblare este o tehnică în care mai multe instrucțiuni sunt executate simultan. Se utilizează intens în procesoarele moderne:

- AMD Opteron X4 (Barcelona)
- procesoarele Intel

Un ciclu de spălare pentru rufe versiunea non-pipe:

- Introducerea rufelor în mașina de spălat
- După terminarea ciclului de spălare, se introduc rufele în uscător
- Rufele uscate se calcă
- Rufele călcate se pun într-un loc pentru utilizarea lor viitoare
- Se repornește cu ciclul de spălare

Cât timp este necesar pentru un ciclu de spălare în acest caz?

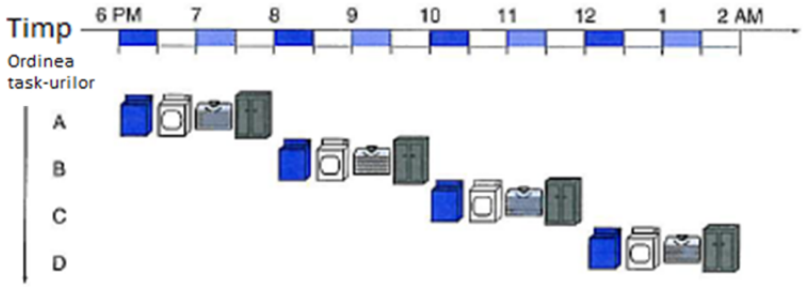

Figura 5.1: Un ciclu de spălare pentru rufe versiunea non-pipeline

Versiunea pipe-line:

- După terminarea primului ciclu de spălare și încărcarea uscătorului cu rufe, se poate iniția un nou ciclu de spălare

- Se scot rufele din uscător și se începe procesul de călcare a lor, se mută rufele spălate în uscător și se pune o nouă încărcătură în mașina de spălat
- Se pun rufele la loc sigur și se continuă procesul

Cât durează versiunea pipe?

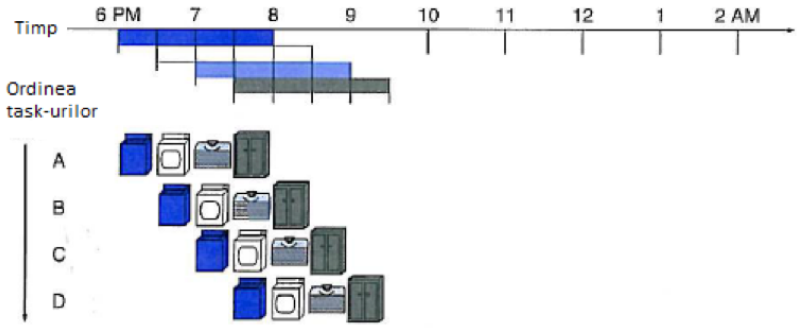

Figura 5.2: Un ciclu de spălare pentru rufe versiunea pipeline

Observații:

- Perioada de ceas alocată fiecărei operații trebuie să fie egală
- Banda de asamblare îmbunătățește performanța/ randamentul (throughput) sistemului
- Timpul total pentru executarea tuturor task-urilor este mai mic
- Varianta pipeline este mult mai rapidă decât varianta non-pipe

5.2 Proiectare pipeline

Background: Cursul 10 legat de proiectarea procesorului MIPS în versiunea Banda de asamblare

5.2.1 Execiţiu

Să se proiecteze în Verilog următorul circuit utilizând banda de asamblare:

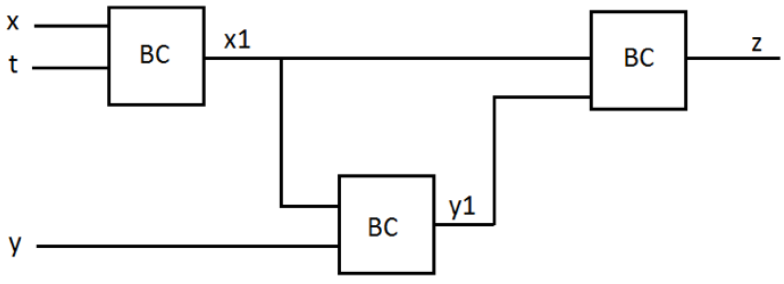

Figura 5.3: Versiune fără pipeline

În figura 5.3, blocul logic combinațional BC poate fi considerat ca implementând orice logică combinaţională ($c = a + b; c = a \wedge b; etc.$) Presupunând ca BC realizează operația simplă $a = b \wedge c$, implementarea Verilog non-pipe este:

```
23  module bloc(a,b,c);
24  input a,b;
25  output c;
26
27  assign c = a ^ b;
28
29  endmodule
```

(a) Modul bloc

```
32  module final(x,t,y,z);
33  input x,y,t;
34  output z;
35
36  wire x1, y1;
37
38  bloc b1(x, t, x1);
39  bloc b2(x1, y, y1);
40  bloc b3(x1, y1, z);
41
42  endmodule
```

(b) Modul final

Figura 5.4: Implementare non pipeline

Observați cu atenție ce se întâmplă în fiecare stagiu al pipeline-ului si felul în care sunt inserate registrele fiecărui stagiu! Ce observați în urma simulării circuitului?

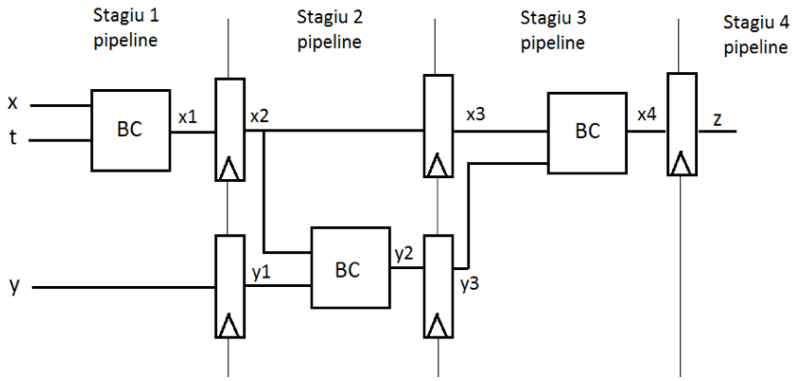

Figura 5.5: Versiune cu pipeline. Ceasul și semnalele registrelor (reset/ load/ clear) sunt comune pentru întregul circuit.

5.3 Reprezentarea numerelor în virgulă mobilă

Reprezentarea numerelor în virgulă mobilă face obiectul standardului IEEE 754. Conform acestui standard reprezentarea lor este alcătuită din două părți. Prima parte reprezintă un număr cu semn denumit *mantisă*. Partea a doua specifică poziția punctului zecimal și se numește *exponent*.

Reprezentarea numerelor în virgulă mobilă poate fi făcută în precizie simplă sau dublă.

Folosirea preciziei simple impune o reprezentare a numărului utilizând 32 de biți. Primul bit (bitul cel mai semnificativ) este bitul de semn. Dacă valoarea acestui bit este 0, numărul este pozitiv, altfel numărul este negativ. Următorii 8 biți sunt alocați pentru valoarea exponentului, iar ultimii 23 de biți reprezintă mantisa. Gama de reprezentare în cazul preciziei simple este: $-1.8 * 10^{-38} \div 3.4 * 10^{38}$.

Reprezentarea generală a unui număr în virgulă mobilă, folosind precizia simplă, este dată în ecuația 5.1.

$$(-1)^s * (1 + MANTISA) * 2^{Exponent-127} \tag{5.1}$$

Valoarea variabilei *Exponent* din cadrul ecuației 5.1. este 127 pentru a se evita valori negative ale exponentului. Pentru precizia dublă, această valoare se modifică devenind 1023.

5.4 Adunarea și scăderea în virgulă mobilă

Resursele hardware necesare implementării acestor operații sunt prezentate în figura 5.6.

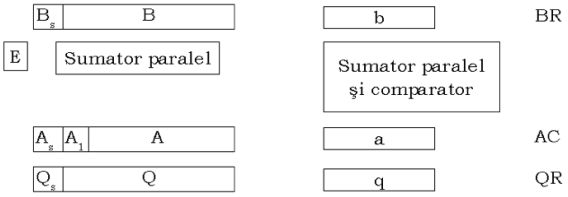

Figura 5.6: Resursele hardware necesare adunării/scăderii a două numere în virgulă mobilă.

Algoritmul necesită parcurgerea următoarelor etape:

1. Se verifică dacă unul dintre operanzi este zero sau nu;
2. Se aliniază mantisele;
3. Se adună sau se scad mantisele;
4. Se normalizează rezultatul obținut.

Algoritmul de adunare/scădere a două numere în virgulă mobilă utilizând tehnica bandă de asamblare poate fi observat în figura 5.7.

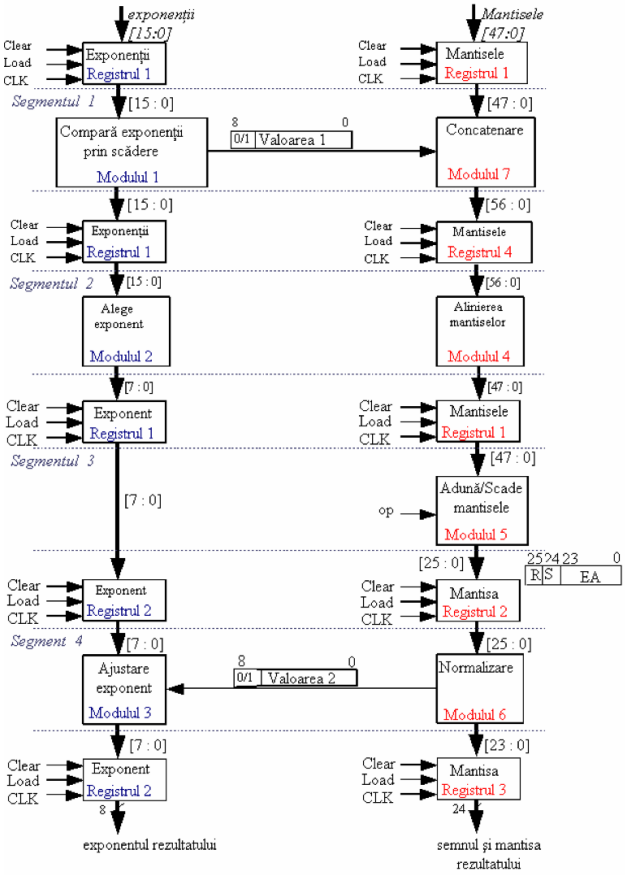

Figura 5.7: Adunarea și scăderea a două numere reprezentate în virgulă mobilă.

În cadrul segmentului 1 al benzii de asamblare, exponenții sunt comparați prin scădere. Exponentul mai mare este ales ca exponent al rezultatului. Rezultatul diferenței indică de câte ori mantisa asociată cu exponentul mai mic trebuie deplasată către dreapta. Astfel se realizează alinierea mantiselor.

Variabila *Valoarea1* memorează rezultatul comparării. Dacă bitul cel mai semnificativ al acestei variabile este 0, mantisa primului număr trebuie deplasată dreapta. Pentru a memora numărul de deplasări care trebuie efectuate, au fost alocați 8 biți (*Valoarea1[7 : 0]*).

Dacă *Valoarea1[8]* este 1, atunci mantisa asociată numărului 2 trebuie deplasată dreapta de un număr de ori egal cu valoarea lui *Valoarea1[7 : 0].* Pentru implementarea sumatorului care apare în cadrul benzii de asamblare se recomandă utilizarea unui sumator cu transport anticipat (CARRY LOOK AHEAD).

În cadrul segmentului 3 al benzii de asamblare, cele două mantise sunt adunate sau scăzute în funcție de valoarea semnalului *OP*. Dacă *OP = 0* atunci trebuie realizată operația de adunare și, în consecință, cele două mantise trebuie adunate. Dacă *OP = 1,* operația de scădere trebuie realizată și, în consecință, cele două mantise trebuie scăzute.

Rezultatul pasului de adunare/scădere a mantiselor se reprezintă pe 26 de biți. Bitul cel mai semnificativ, bitul 25, este utilizat pentru a specifica dacă mantisa este egală sau nu cu zero. Restul de biți este folosit pentru a reprezenta registrele *E* și *A* concatenate.

Rezultatul este normalizat în cadrul segmentului 4 al benzii de asamblare. Când apare o depășire superioară, mantisa-rezultat este deplasată dreapta și exponentul este incrementat cu o unitate. Când apare o depășire inferioară, numărul de zerouri din cadrul mantisei (pozițiile cele mai semnificative) determină numărul de deplasări stânga al mantisei, număr care trebuie scăzut din exponent. Memorarea acestui număr se face prin intermediul variabilei *Valoarea2*.

Dacă bitul cel mai semnificativ al variabilei *Valoarea1* este 0, atunci exponentul trebuie deplasat spre stânga, altfel exponentul trebuie să fie deplasat spre dreapta. Numărul de deplasări este indicat de către *Valoarea2[7 : 0]*.

Rezultatele simulării acestui algoritm folosind mediul de dezvoltare Xilinx pot fi observate în figura 5.8.

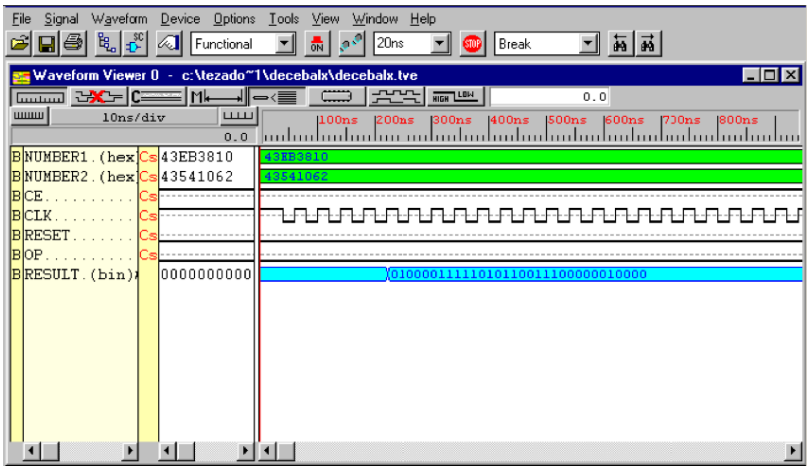

Figura 5.8: Rezultatele simulării algoritmului de adunare/scădere.

5.5 Înmulțirea în virgulă mobilă

Operația de înmulțire în virgulă mobilă presupune realizarea următorilor pași:

1. Se verifică dacă unul dintre numere este zero sau nu;
2. Se adună exponenții;
3. Se înmulțesc mantisele;
4. Se normalizează produsul.

Implementarea în bandă de asamblare al acestui algoritm poate fi observată în figura 5.9.

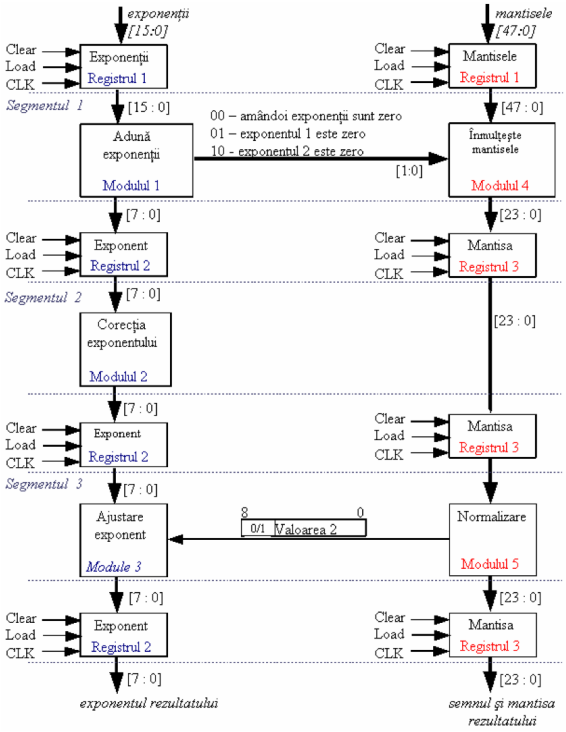

Figura 5.9: Algoritmul de înmulțire a două numere în virgulă mobilă.

Înmulțirea a două numere binare se realizează cu ajutorul unui circuit combinațional (denumit matrice de multiplicare) care formează toți biții produsului odată. Aceasta este cea mai rapidă metodă deoarece timpul necesar efectuării operației este egal cu timpul de propagare al semnalului luând în calcul ruta cea mai dezavantajoasă.

Circuitul matrice de multiplicare necesită un număr mare de porți, implementarea lui este neeconomică, acesta fiind principalul motiv pentru care acest circuit nu a avut o răspândire foarte mare până la apariția circuitelor integrate. Proiectarea unui asemenea circuit ale cărui intrări ar avea dimensiunile de j respectiv k biți va necesita $j*k$ porți I și sumatoare de $(j-1)*k$ biți pentru a obține un produs de $(j+k)$ biți.

Un exemplu de circuit matrice de multiplicare este prezentat în figura 5.10.

Trebuie remarcat că exponentul rezultat corect este obținut prin scăderea valorii 127 din exponentul sumă.

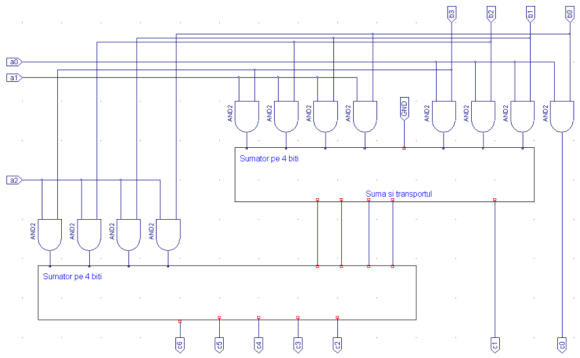

Figura 5.10: Circuitul matrice de multiplicare pentru k = 4 și j = 3..

5.6 Împărțirea în virgulă mobilă

Algoritmul de împărțire a două numere reprezentate în virgulă mobilă necesită parcurgerea următorilor pași:

1. Se verifică dacă unul dintre cei doi operanzi este zero sau nu;
2. Se inițializează registrele și se evaluează semnul;
3. Se aliniază deîmpărțitul;
4. Se scad exponenții;
5. Se împart mantisele.

Algoritmul în bandă de asamblare al acestei operații este prezentat în figura 5.11.

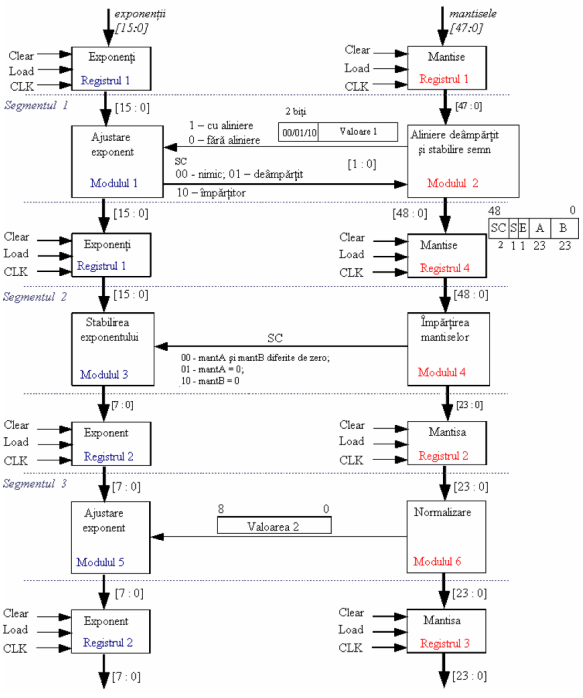

Figura 5.11: Algoritmul de împărțire a două numere în virgulă mobilă