Lucrarea 2

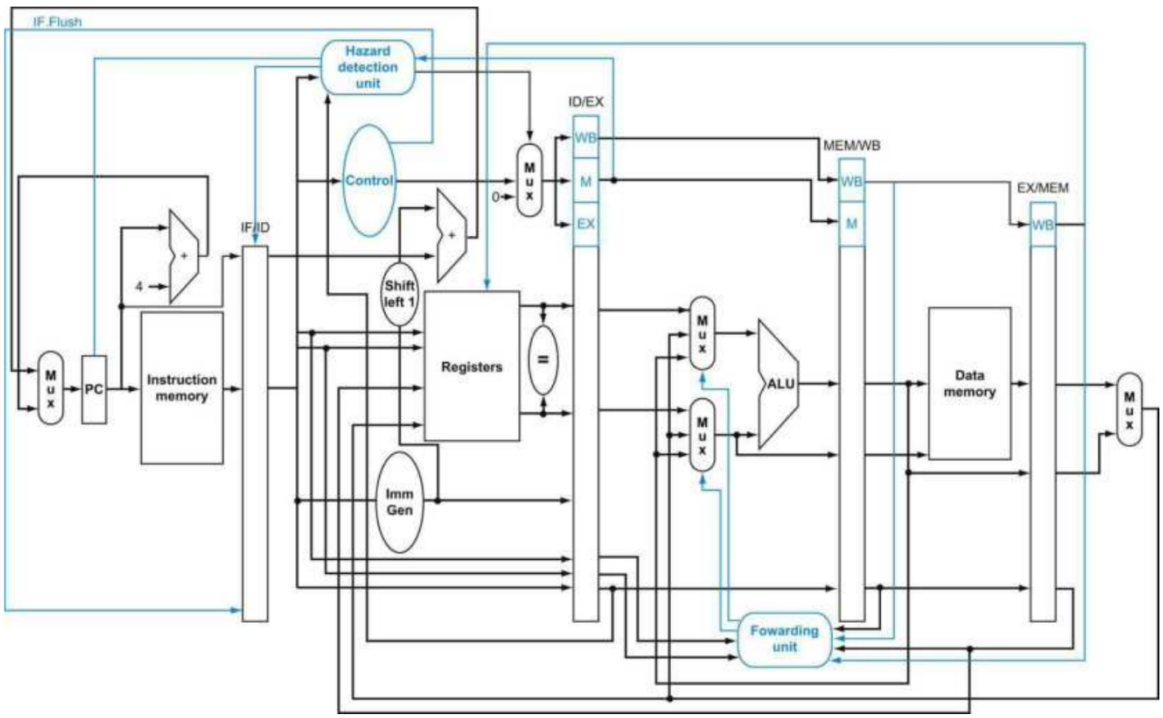


Fig. 1: Pipeline RISC-V

O arhitectură de set de instrucțiuni sau ISA (Instruction Set Arhitecture) este un model abstract al arhitecturii unui computer, definind lucruri precum modelul de registru și instrucțiunile codului mașinii. O realizare a unui ISA, cum ar fi o unitate centrală de procesare (CPU), se numește implementare. ISA utilizate pe scară largă includ x86 și ARM.

Pronunțat „risk-five”, RISC-V este un ISA bazat pe principiile computerului cu set de instrucțiuni reduse (RISC). Spre deosebire de majoritatea celorlalte modele ISA, acesta este furnizat sub o licență open source care nu necesită taxe pentru utilizare.

Pe baza acestui ISA [2] se pot propune diferite design-uri de procesor, dar toate trebuie sa respecte formatul și modul de execuție a instrucțiunilor propuse. Procesorul ce va fi studiat și implementat în cadrul laboratorului va urma principiile expuse în [1] și [2], respectând extensia RV32I (pagina 9 din [2]) ce propune un procesor pe 32 de biți (atât pentru date cât și pentru instrucțiuni) capabil sa execute instrucțiuni aritmetico-logice cu numere întregi fără suport hardware pentru virgulă mobilă sau operații de înmulțire/împărțire.

Fig. 1 prezinta schema bloc a unui procesor RISC-V cu 5 etape de pipeline:

1. IF(Instruction Fetch): citirea instructiunii curente
2. ID(Instruction Decode): decodarea instructiunii si citirea registrilor
3. EX(Execute): executarea instructiunii
4. MEM(Memory): operatii de citire/scriere cu memoria de date
5. WB(Write Back): scrierea rezultatelor finale inapoi in registri

Lucrarea de fata isi propune implementarea a ceea ce se numește “front-end-ul” unui processor. Acesta se ocupă cu citirea și decodificarea instrucțiunilor ce urmează a fi executate de către processor și este reprezentat în acest design de către primele 2 etape de pipeline: IF+ID.

Pentru a executa o instructiune, primii pasi pe care trebuie sa-i facem, sunt sa citim instructiunea din memorie(mai precis codificarea acesteia), si sa o decodificam pentru a intelege anumite aspecte despre ea: ce tip de instructiune este, ce operatie trebuie executata, ce registri trebuie folositi, etc.

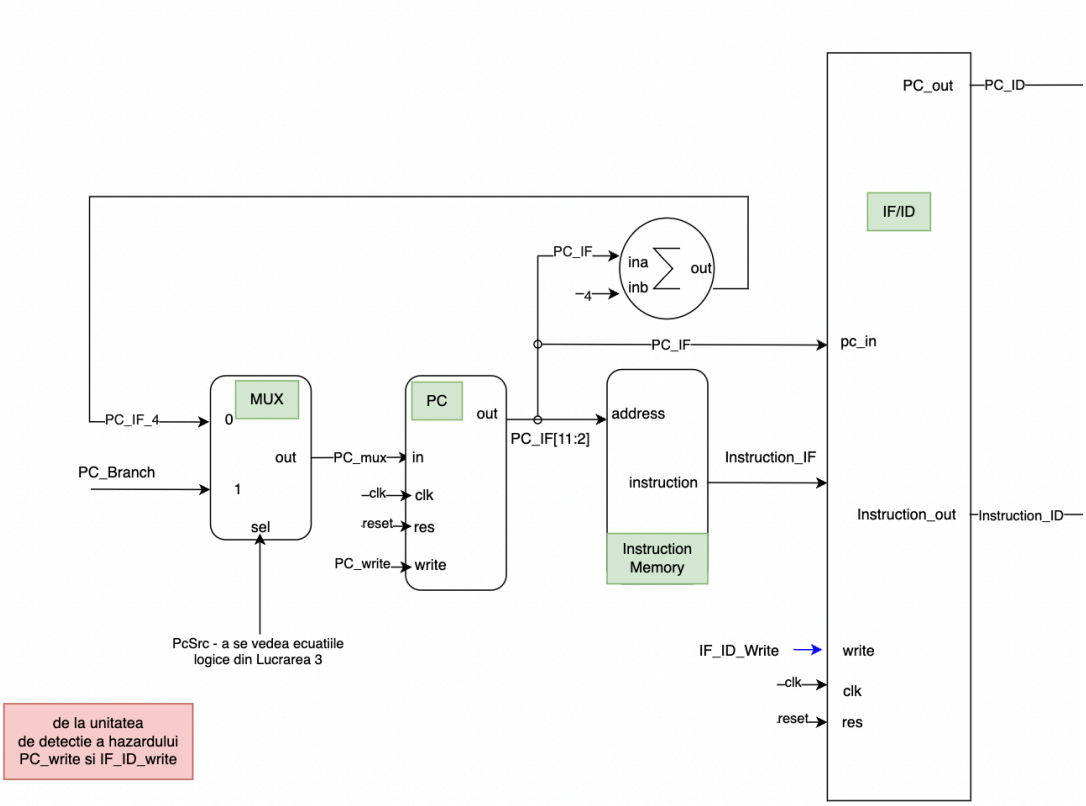


Fig. 2: Etapa IF împreună cu registrul de pipeline IF/ID

Pentru a ne pregati de executia urmatoarei instructiuni, trebuie de asemenea sa incrementam contorul de program(program counter - PC), astfel incat adresa sa de iesire sa reprezinte urmatoarea instructiune din memorie, aflata la o distanta de 4 octeti (32 de biți) de instructiunea curenta(mai multe detalii la Cap 4.3 din [1])

Această adresă este trimisă mai departe la memoria de instrucțiuni de unde este extrasă instrucțiunea corespondentă, instrucțiune ce este trimisă mai departe la etapa ID pentru decodificare printr-un registru de pipeline.

Întru-cât procesorul va fi implementat în două lucrări, etapele IF și ID vor conține în implementarea curentă semnale ce provin din etapele viitoare ce vor fi implementate în lucrarea 3. Din acest punct de vedere, pentru a putea implementa

complet front-end-ul procesorului în această etapa, vom simula in testbench acele semnale ce nu pot fi generate în lucrarea curentă.

Spre exemplu, în etapa IF, semanelele IF_ID_Write și PC_Write din Fig. 2 sunt generate de către unitatea de detecție a hazardurilor, unitate ce va fi implementată în Lucrarea 3. Din acest motiv, în implementarea front-end-ului noi vom implmenta logica de funcționare pentru aceste semnale, iar pentru a testa front-end-ul, vom simula valorile acestor semnale în testbench. Acest lucru este valabil și pentru semnalel PC_Branch și PcSrc.

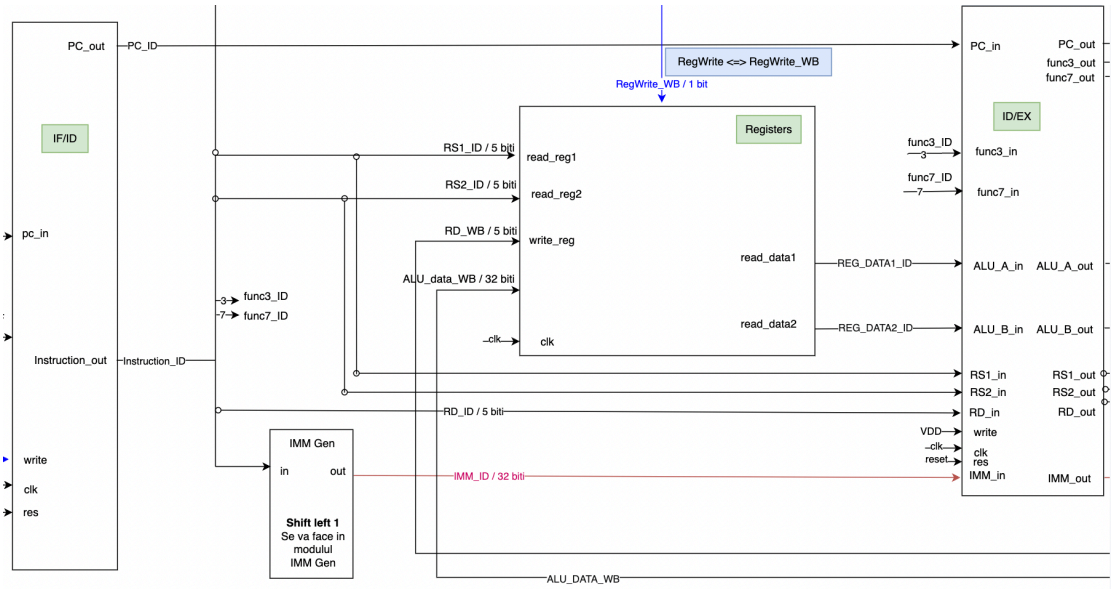


Fig. 3: Etapa ID împreună cu regiștrii de pipeline IF/ID și ID/EX

În etapa ID, instrucțiunea citită în IF este descompusă în mai multe câmpuri precum adresele regiștrilor sursă RS1 și RS2 (dacă este cazul), registrul destinație RD, valoarea imediată IMM (dacă este cazul) și diferite codificări pentru operația ce trebuie executată în câmpurile OPCODE, FUNCT3 și FUNCT7. Toate aceste câmpuri definesc ce face instrucțiunea, și ce regiștri sunt folosiți.

Și în cadrul acestei etape vom acea o serie de semnale ce vor trebui simulate din cazuza faptului că acestea vor fi implementate proriu-zis în Lucrarea 3: RD_WB, RegWrite_WB, și ALU_DATA_WB.

Atât aceste semnale cât și cele de la etapa IF sunt toate simulate în cadrul fișierului de testbench furnizat a cărui simulare poate fi văzută în Fig. 4.

Cerințe:

1. Sa se implementeze urmatoarele module din IF:
   a) Mux:

```
module mux2_1(input [31:0] ina, inb,
              input sel,
              output [31:0] out);
```

b) PC – un registru de tipul D. Semnalul ***clk*** și semnalul ***res*** sunte semnale unice la nivelul pipe-ului.

```
module PC(input clk,res,write,
          input [31:0] in,
          output reg [31:0] out);
```

c) Memorie Instructiuni - cu citire asincronă. În acest caz nu este nevoie de semnal ***clk***. (hint: folositi $readmemh pt a incarca continutul memoriei dintr-un fisier; pentru codificarile instructiunilor si opcode-urile aferente, se va consulta documentul ***riscv-spec-v2.2.pdf, paginile 104-107***):

```
module instruction_memory(input [9:0] address,
                          output reg [31:0] instruction);
```

Exemplu de folosire a directivii $readmemh pentru incarcarea continutului fisierului *code.mem* in memoria ***codeMemory****(codul este folosit in modulul instruction_memory!)*:

```
reg [31:0] codeMemory [0:1023];

initial $readmemh("code.mem", codeMemory);
```

De asemenea, in cazul de fata cuvantul memoriei noastre este pe 32 de biti(o linie de memoria va cuprinde toti cei 32 de biti ai unei instructiuni). Procesorul nostru RISC-V stie ca o instructiune este formata din 4 octeti, si astfel se asteapta ca instructiunea urmatoare se va afla la 4 octeti de cea curenta(de aici si adunarea cu 4 a PC-ului).
O problema ce apare in varianta propusa mai sus a memoriei, este ca Verilog va accesa instructiunile din memoriei la distanta de o linie, una de cealalta(prima instructiune va fi pe prima linie, a doua instructiune pe cea de-a doua linie s.a.m.d).
Pentru a rezolva aceasta problema, va trebui sa shiftam adresa noastra PC la dreapta de doua ori(PC>>2) inainte de a extrage ultimii 10 biti pentru adresa instructiunii noastre. Motivul este urmatorul: adresa noastra PC va fi un multiplu de 4, din cauza adunarilor repetate cu 4, astfel pentru a trece de la o secventa de adrese multiple de 4, la o secventa de adrese consecutive, cu distanta 1, vom face o impartire la 4(sau o shiftare la dreapta cu 2).
Obs: Cunoasteti o metoda in Verilog pentru a extrage direct cei 10 biti de adresa corecti fara a folosi nici macar shiftarea?

d) Sumator:

```
module adder(input [31:0] ina,inb,
             output reg [31:0] out);
```

2. Sa se completeze modulul IF care conecteaza modulele definite la cerința 1 avand urmatorele intrari si iesiri:

```
module IF(input clk, reset,
          input PCSrc, PC_write,
          input [31:0] PC_Branch,
          output [31:0] PC_IF,INSTRUCTION_IF);
```

3. Sa se implementeze si simuleze separate bancul de registri (Registers) din ID, avand urmatoarele intrari si iesiri:

```
module registers(input clk,reg_write,
                 input [4:0] read_reg1,read_reg2,write_reg,
                 input [31:0] write_data,
                 output [31:0] read_data1,read_data2);
```

Hint: Bancul de registri pentru procesorul RISC-V contine 32 de registri pe 32 de biti, notati de la x0 la x31 (x0 va fi mereu 0). Bancul reprezinta o memorie multi-port cu 2 porturi de citire pentru citirea celor 2 operanzi si un port de scriere unde va fi scris registrul rezultat.
Semnalul sincron ***reg_write*** activ pe 1 logic, declanseaza scrierea in bancul de registri la adresa data de ***write_reg*** a valorii ***write_data***.
Citirea este asincrona si se realizeaza imediat cu schimbarea adreselor de citire ***read_reg1*** respectiv ***read_reg2***, valorile citite aflandu-se in ***read_data1***, respectiv ***read_data2***
Pentru o mai buna simulare, folositi un ciclu for(intr-un bloc initial definit in modulul registers) in care initializati fiecare registru cu numere de la 0 la 31.

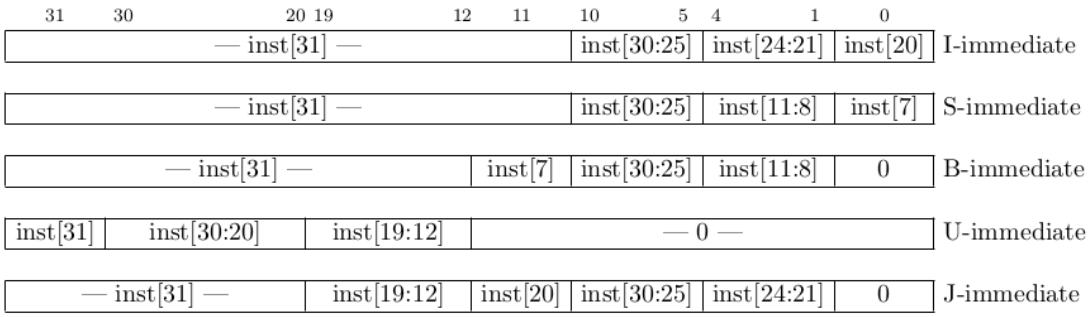
4. Sa se implementeze unitatea de generare a datelor imediate cu urmatoarea definitie a modulului:

```
module imm_gen(input [31:0] in,
               output reg [31:0] out);
```

Generarea valorilor imediate se va face pentru urmatoarele instructiuni:

- lw
- addi
- andi
- ori
- xori
- slti
- sltiu
- srli,srai
- slli
- sw
- beq,bne,blt,bge,bltu,bgeu

Hint: urmariti generarea valorii immediate specificata in tabelul de mai jos, in functie de tipul instructiunii: I,S,B,U,J(se va consulta documentul ***riscv-spec-v2.2.pdf, pagina 12, Fig 2.4***):



5. Sa se completeze modulul ID care conecteaza modulele definite la cerințele 3 și 4 avand urmatorele intrari si iesiri:

```
module ID(input clk,
          input [31:0] PC_ID,INSTRUCTION_ID,
          input RegWrite_WB,
          input [31:0] ALU_DATA_WB,
          input [4:0] RD_WB,
          output [31:0] IMM_ID,
          output [31:0] REG_DATA1_ID,REG_DATA2_ID,
          output [2:0] FUNCT3_ID,
          output [6:0] FUNCT7_ID,
          output [6:0] OPCODE_ID,
          output [4:0] RD_ID,
          output [4:0] RS1_ID,
          output [4:0] RS2_ID);
```

6. Sa se implementeze registrul de pipeline ce leaga IF de ID
7. Sa se implementeze si simuleze un modul top care sa uneasca cele 2 module IF si ID cu urmatoarele intrari si iesiri, folosind fisierul *ID.v* dat:

```
module RISC_V_IF_ID(input clk,   //semnalul de ceas global
                    input reset, //semnalul de reset global

                    //semnale provenite din stagii viitoare
                    //sunt pre-setate pentru aceasta lucrare
                    //vor fi discutate in lucrarile urmatoare
                    input IF_ID_write,       //semnal de scriere pentru registrul de pipeline IF_ID
                    input PCSrc,PC_write,    //semnale de control pentru PC
                    input [31:0] PC_Branch,  //PC-ul calculat in etapa EX pentru instructiunile de salt
                    input RegWrite_WB,       //semnal de activare a scrierii in bancul de registri
                    input [31:0] ALU_DATA_WB,//rezultatul calculat de ALU
                    input [4:0] RD_WB,       //registrul rezultat in care se face scrierea

                    //semnale de iesire din ID
                    //vor fi vizualizate pe simulare
                    output [31:0] PC_ID,          //adresa PC a instructiunii din etapa ID
                    output [31:0] INSTRUCTION_ID, //instructiunea curenta in etapa ID
                    output [31:0] IMM_ID,         //valoarea calculata
                    output [31:0] REG_DATA1_ID,   //valoarea primului registru sursa citit
                    output [31:0] REG_DATA2_ID,   //valoarea celui de-al doilea registru sursa citit

                    output [2:0] FUNCT3_ID,  //funct3 din codificarea instructiunii
                    output [6:0] FUNCT7_ID,  //funct7 din codificarea instructiunii
                    output [6:0] OPCODE_ID,     //opcode-ul instructiunii
                    output [4:0] RD_ID,      //registru destinatie
                    output [4:0] RS1_ID,     //registru sursa1
                    output [4:0] RS2_ID);    //registru sursa2
```

8. Codificati instructiunile RISC-V urmatoare conform informatiilor din Fig. 3 (pentru codificarile instructiunilor si opcode-urile aferente, se va consulta documentul ***riscv-spec-v2.2.pdf, paginile 104-107***):

add x2,x1,x0
addi x1,x1,1
and x3,x1,x2
ori x4,x1,1
sw x4,4(x5)
lw x12,8(x0)
beq x18,x0,5c

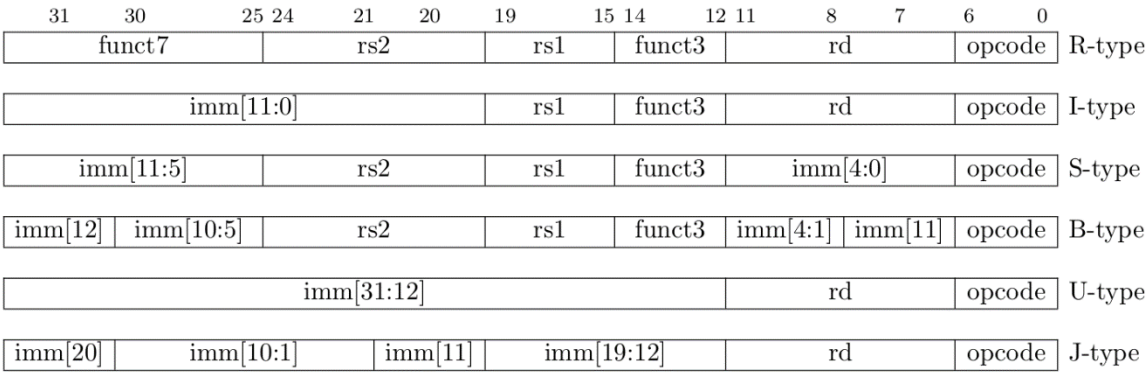


Fig. 3: Formatul instructiunilor aritmetice(R), imediate(I), salt conditionat(B), lucru cu memoria(S), salt neconditionat(J), upper-immediate(U) ale procesorului RISC-V

Pentru a verifica corectitudinea cerintelor, folositi fisierul de test oferit care ar trebui sa aiba urmatoarea simulare:

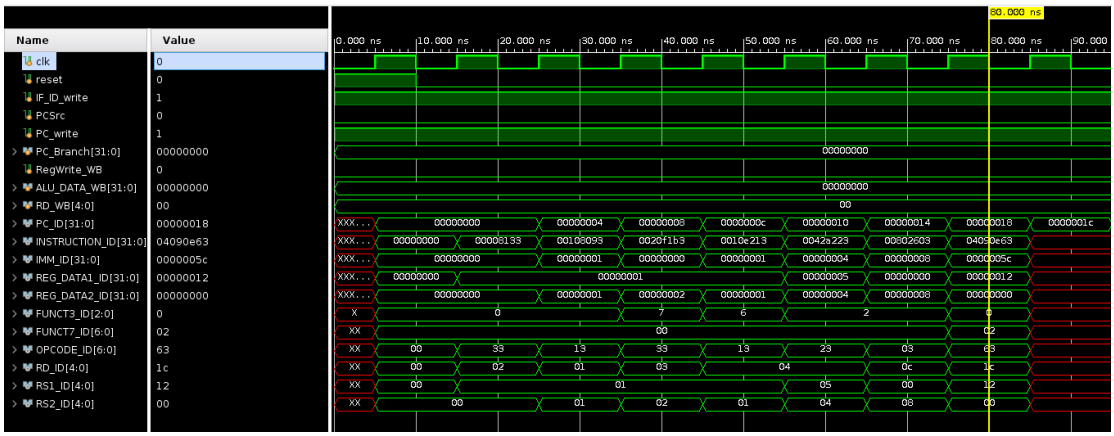


Fig. 4: Formele de unda rezultate in urma simularii intructiunilor specificate la cerinta 8, folosind fisierul de test dat

Bibliografie:

[1] David A. Patterson, John L. Hennessy, *Computer Organization and Design RISC-V edition,* 2018

[2] Andrew Waterman1, Krste Asanoví c*, The RISC-V Instruction Set ManualVolume I: User-Level ISA, 2017*