Lucrarea 3

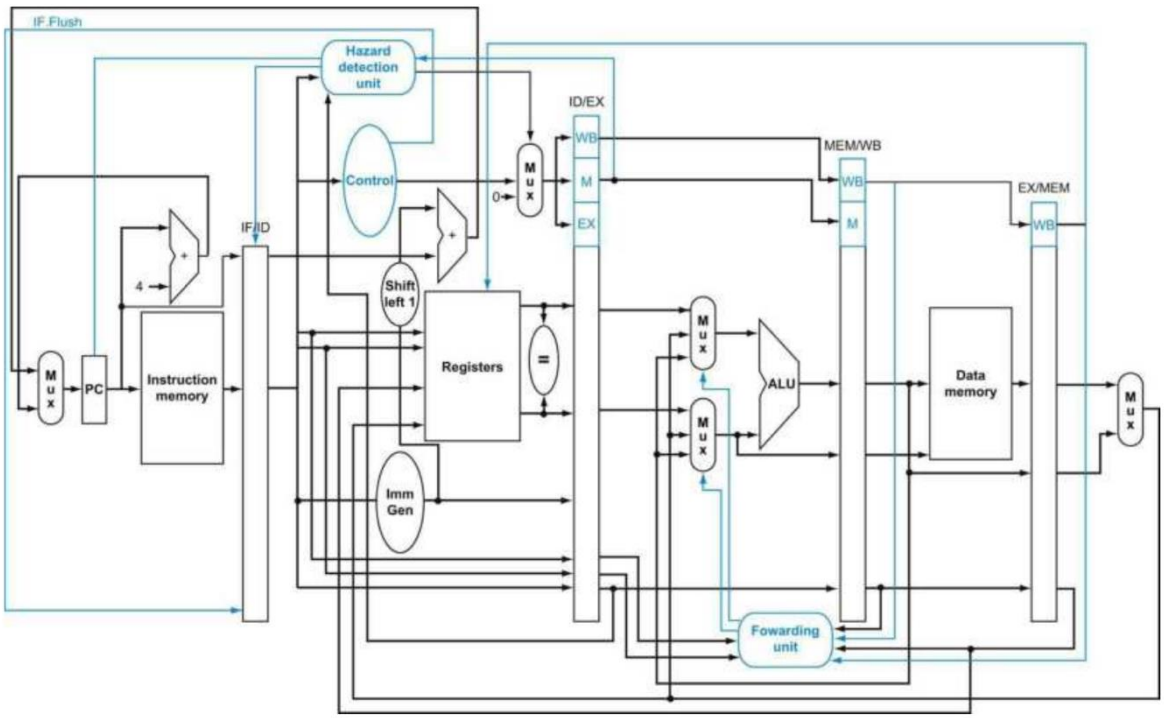


Fig. 1: Procesorul RISC-V – varianta pipeline

Lucrarea curenta isi propune implementarea etapelor EX, MEM, WB, plus doua unitati folosite in detectia hazardurilor si unitatea de forwarding, astfel realizandu-se un prototip complet a procesorului RISC-V.

In etapa EX, unitatea ALU se va ocupa de executarea propriu-zisa a operatiei aritmetico-logice din instructiunea curenta. Un prototip minimal de ALU este scris in Fig. 2, unde in functie de funct3, funct7 si AluOp(generat de unitatea de control), se genereaza semnalele de control corespunzatoare operatiei.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

Fig. 2: Tabelul logic pentru generarea semnalelor de contro a unitatii ALU pentru instructiunile AND, OR, ADD, SUB

Etapa MEM se ocupa de scrierea/citirea in memoria de date de 4KB, similara cu cea de instructiuni. Aceasta suporta, pe langa operatia de citire asincrona, operatia de scriere sincrona.

Etapa WB se ocupa de scrierea rezultatului final in bancul de registri. Rezultatul final este ales de un multiplexor, dintre valoarea citita din memoria de date(daca avem de a face cu instructiunea LW), sau valoarea calculata de ALU(daca avem de a face cu instructiune de tipul R-type sau I-type). Semnalul de control MemtoReg functioneaza ca semnal de selectie pentru multiplexorul mentionat.

Unitatea de Forwarding este responsabila pentru rezolvarea hazardurilor de tip RAW. Aceasta face bypass la valorile calculate de ALU ce nu au fost inca scrise in bancul de registri, si se afla inca in pipeline(in etapele MEM sau WB). Aceasta este prezentata in Fig. 3, si genereaza semnalele ForwardA si ForwardB ce sunt responsabile pentru alegerea operanzilor trimisi spre ALU(mai multe detalii de la pagina 569).

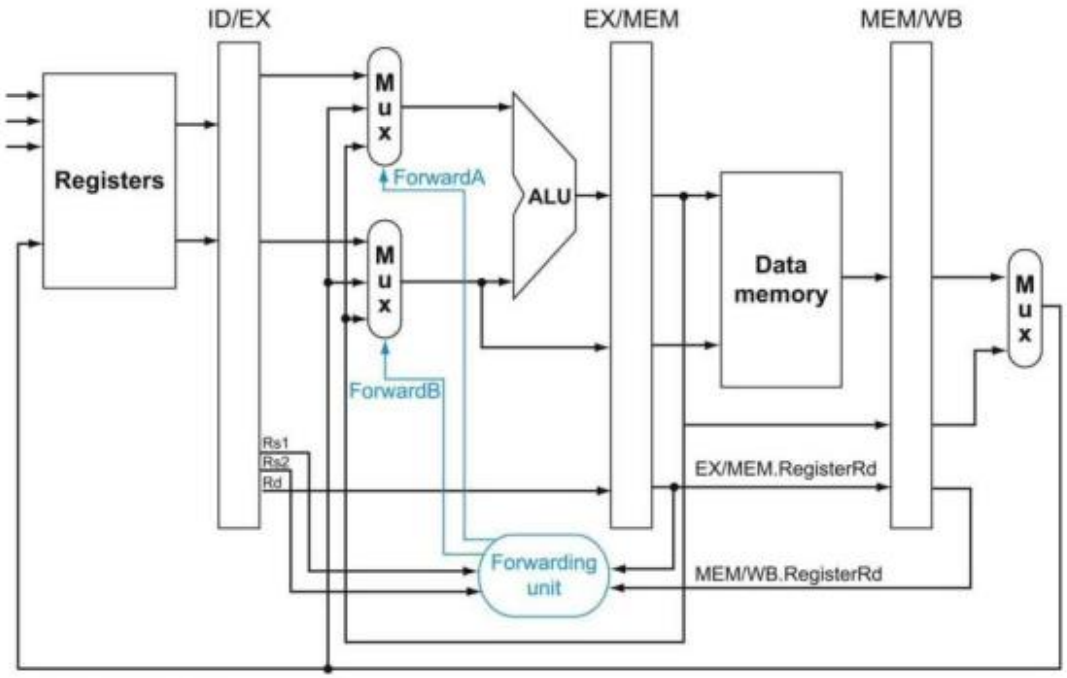


Fig. 3: Unitatea de Forwarding

Multiplexoarele din Fig. 3 selecteaza valorile registrilor sursa ce vor fi trimisi spre ALU(fie acestia din etapa ID, fie bypass-ate din MEM/WB). Multiplexorul nou introdus in diagrama selecteaza valoarea de input pentru ALU, dintre o valoare imediata sau un registru. In Fig. 4 avem logica pentru a putea realiza operatii atat intre 2 registri, cat si intre un registru si o valoare imediata.

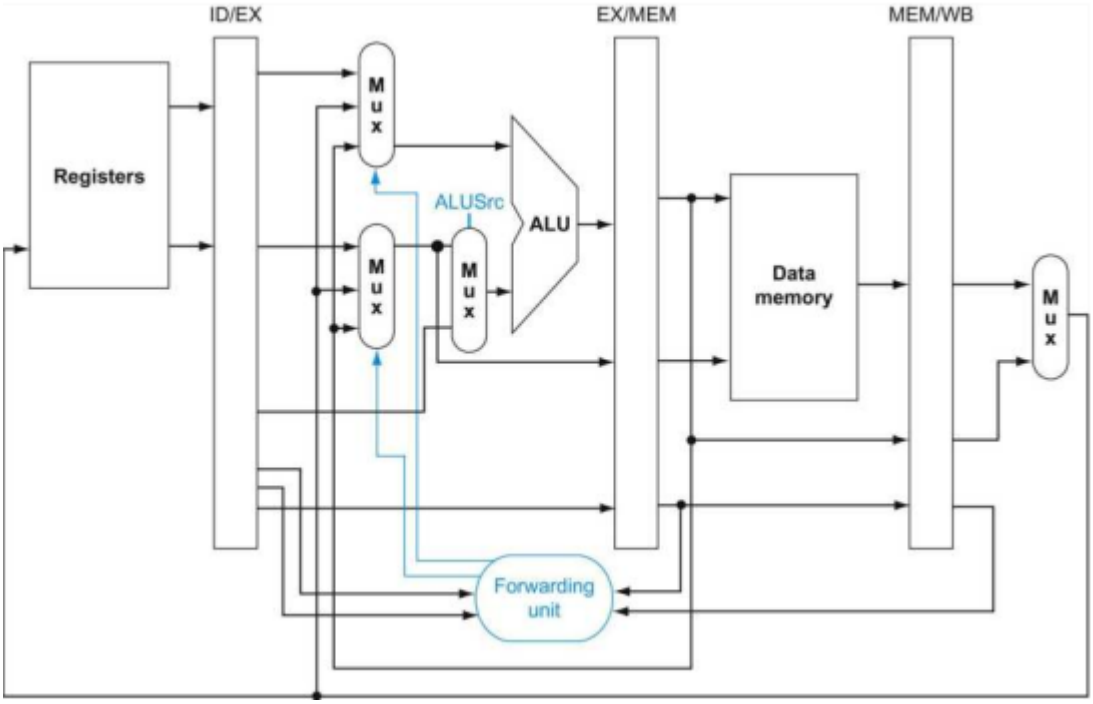


Fig. 4: Etapa EX cu posibilitatea de a executa operatii intre registri si valori immediate.

Pentru a completa etapa EX, este necesara o logica ce calculeaza adresa de salt pentru instructiunile de tip branch. Aceasta adresa este calculata prin adunarea adresei PC curente a instructiunii, si valoarea imediata codificata in aceasta. Fig. 5 prezinta aceasta logica.

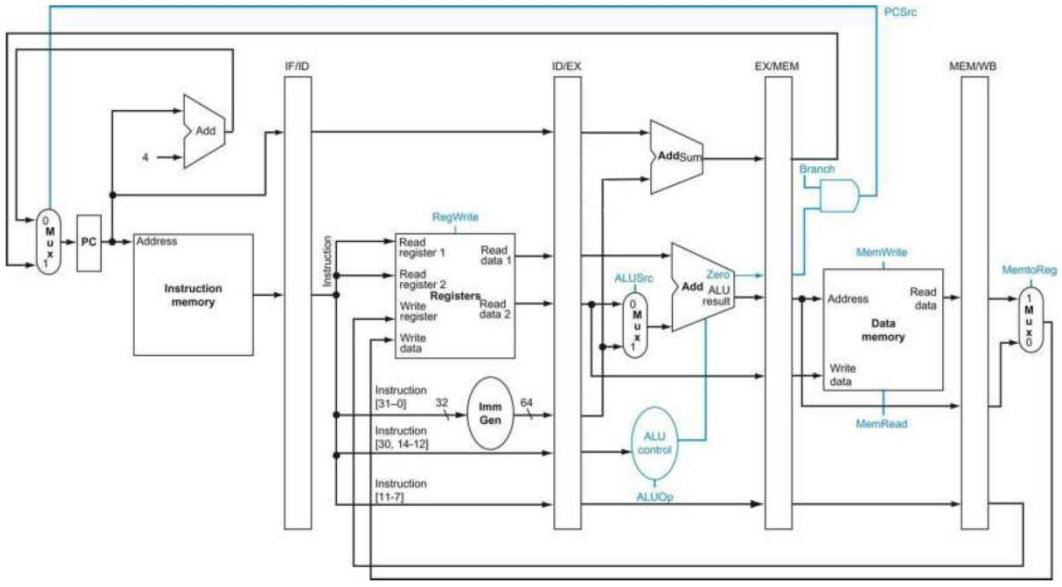


Fig. 5: Logica pentru calculul adresei de salt adaugata in etapa EX, al carui rezultat este propagat in etapa MEM

Unitatea de detectie a hazardurilor(Hazard detection unit) insereaza stall-uri in pipeline atunci cand hazardul RAW nu poate fi rezolvat cu forwarding(mai multe detalii de la pagina 582).

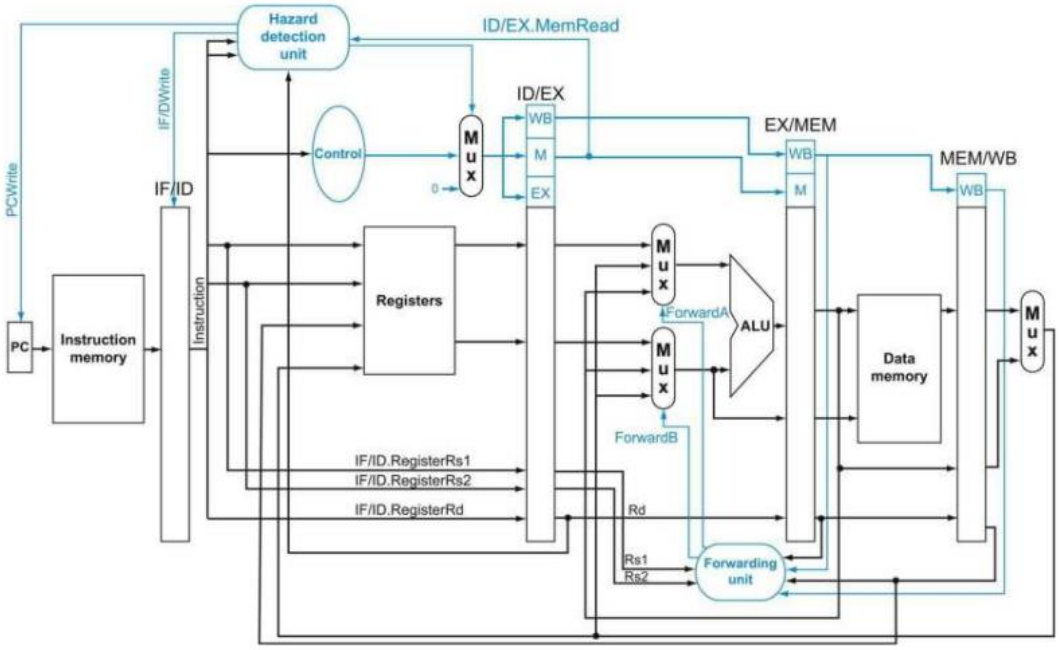


Fig. 6: Procesorul RISC-V complet cu unitatile de forwarding si detectie a hazardurilor

Cerințe:

1. Sa se implementeze unitatea ALUcontrol:

```
module ALUcontrol(input [1:0] ALUop,
                  input [6:0] funct7,
                  input [2:0] funct3,
                  output reg [3:0] ALUinput);
```

Semnalul ALUinput pe 4 biti va desemna operatia executata de ALU, in functie de instructiunea curenta (se vor folosi field-urile funct3 si funct7 si semnalul de control ALUop pentru a identifica ce operatie trebuie executata).

Se va folosi urmatorul tabel pentru generarea lui ALUinput in functie de instructiune:

Instruction	ALUinput code
ld, sd	0010
add	0010
sub	0110
and	0000
or	0001
xor	0011
srl, srli	0101
sll, slli	0100
sra, srai	1001
sltu	0111
slt	1000
beq, bne	0110
blt, bge	1000
bltu, bgeu	0111

2. Sa se implementeze unitatea ALU:

```
module ALU(input [3:0] ALUop,       //ALUinput ce selecteaza operatia
           input [31:0] ina,inb,    //operanzii ce iau parte la operatie
           output zero,             //semnal ce verifica daca rezultatul este 0
           output reg [31:0] out); //rezultatul operatiei
```

3. Sa se implementeze modulul EX ce cuprinde unitatea ALU, ALUcontrol, multiplexoarele necesare pentru alegerea operanzilor si adder-ul ce calculeaza adresa de salt:

```
module EX(input [31:0] IMM_EX,         //valoarea immediate in EX
          input [31:0] REG_DATA1_EX,   //valoarea registrului sursa 1
          input [31:0] REG_DATA2_EX,   //valoarea registrului sursa 2
          input [31:0] PC_EX,          //adresa instructiunii curente in EX
          input [2:0] FUNCT3_EX,       //funct3 pentru instructiunea din EX
          input [6:0] FUNCT7_EX,       //funct7 pentru instructiunea din EX
          input [4:0] RD_EX,           //adresa registrului destinatie
          input [4:0] RS1_EX,          //adresa registrului sursa 1
          input [4:0] RS2_EX,          //adresa registrului sursa 2
          input RegWrite_EX,           //semnal de scriere in bancul de registri
          input MemtoReg_EX,           //...
          input MemRead_EX,            //semnal pentru activarea citirii din memorie
          input MemWrite_EX,           //semnal pentru activarea scrierii in memorie
          input [1:0] ALUop_EX,        //semnalul de control ALUop
          input ALUSrc_EX,             //semnal de selectie intre RS2 si valoarea imediata
          input Branch_EX,             //semnal de identificare a instructiunilor de tip branch
          input [1:0] forwardA,forwardB, //semnalele de selectie pentru multiplexoarele de forwarding

          input [31:0] ALU_DATA_WB,    //valoarea calculata de ALU, prezenta in WB
          input [31:0] ALU_OUT_MEM,    //valoarea calculata de ALU, prezenta in MEM

          output ZERO_EX,              //flag-ul ZERO calculat de ALU
          output [31:0] ALU_OUT_EX,    //rezultatul calculat de ALU in EX
          output [31:0] PC_Branch_EX,  //adresa de salt calculata in EX
          output [31:0] REG_DATA2_EX_FINAL //valoarea registrului sursa 2 selectata dintre
          );                               //valorile prezente in etapele EX, MEM si WB
```

4. Sa se implementeze memoria de date din etapa MEM:

```
module data_memory(input clk,
                   input mem_read,    //semnal de activare a citirii din memorie
                   input mem_write,   //semnal de activare a scrierii in memorie
                   input [31:0] address,    //adresa de scriere/citire
                   input [31:0] write_data, //valoarea scrisa in memorie
                   output reg [31:0] read_data //valoarea citita din memorie
                   );
```

5. Sa se implementeze unitatea de detectie a hazardurilor:

```
module hazard_detection(input [4:0] rd, //adresa registrului destinatie in etapa EX
                        input [4:0] rs1, //adresa registrului sursa 1 decodificat in etapa ID
                        input [4:0] rs2, //adresa registrului sursa 2 decodificat in etapa ID
                        input MemRead,   //semnalul de control MemRead din etapa EX
                        output reg PCwrite, //semnalul PCwrite ce controleaza scrierea in registrul PC
                        output reg IF_IDwrite, //semnal ce controleaza scrierea in registrul de pipeline IF_ID
                        output reg control_sel); //semnal transmis spre unitatea de control
```

Unitatea de detectie a hazardurilor va fi implementata dupa urmatoarea logica:

```
if (ID/EX.MemRead and
   ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
    (ID/EX.RegisterRd = IF/ID.RegisterRs2)))
    stall the pipeline
```

Semnalele de output vor fi puse in locul liniei “stall the pipeline” astfel incat in momentul inserarii unui stall, registrul PC din IF si registrul de pipeline IF_ID sa nu mai permita actualizarea valorilor interne pana cand hazardul nu este rezolvat, iar semnalul control_sel va seta semnalele de control generate de unitatea de control pe 0 logic.

6. Sa se implementeze unitatea de forwarding:

```
module forwarding(input [4:0] rs1,         //adresa registrului sursa 1 in etapa EX
                  input [4:0] rs2,         //adresa registrului sursa 2 in etapa EX
                  input [4:0] ex_mem_rd,   //adresa registrului destinatie in etapa MEM
                  input [4:0] mem_wb_rd,   //adresa registrului destinatie in etapa WB
                  input ex_mem_regwrite,   //semnalul de control RegWrite in etapa MEM
                  input mem_wb_regwrite,   //semnalul de control RegWrite in etapa WB
                  output reg [1:0] forwardA,forwardB); //semnalele de selectie a multiplexoarelor
                                                       //ce vor alege valoarea ce trebuie bypassata
```

Unitatea de forwarding va fi implementatat dupa urmatoarea logica:

1. Pentru hazard in etapa EX:

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10
if  (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10
```

2. Pentru hazard in etapa MEM:

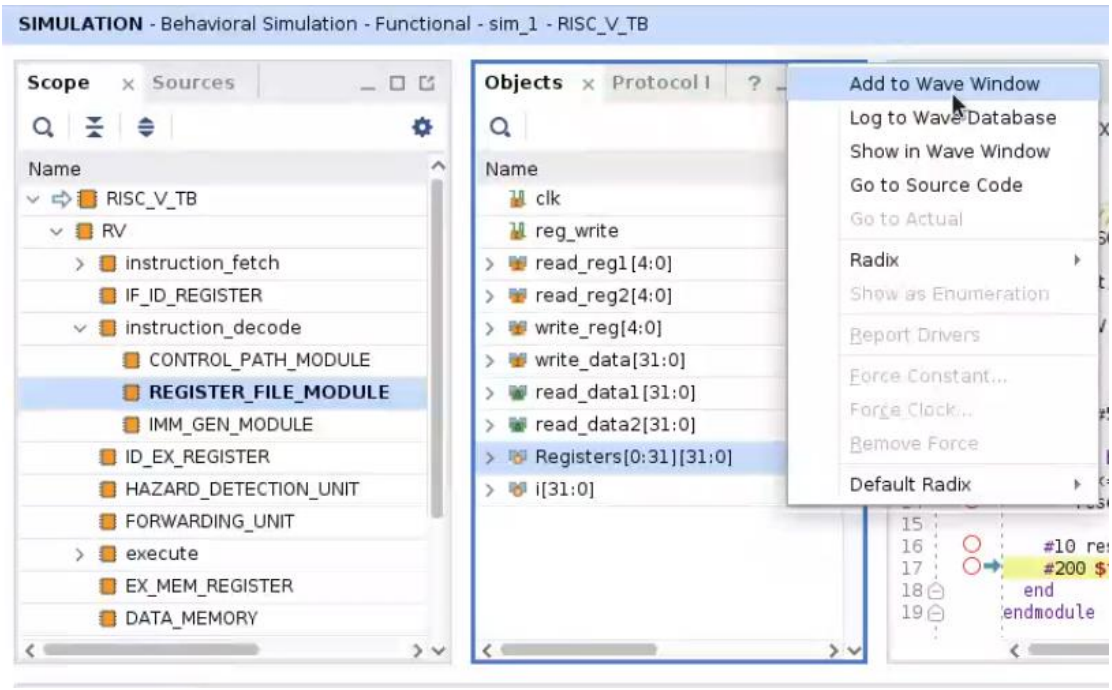
```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

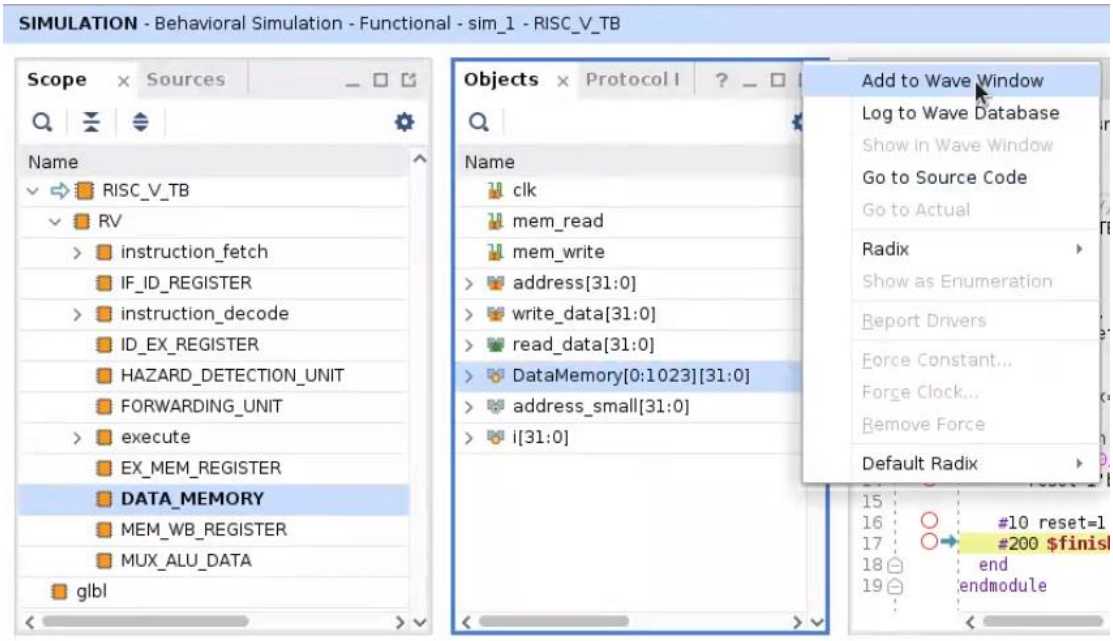
7. Sa se implementeze registrele de pipeline dintre etapele EX si MEM, respectiv MEM si WB

8. Sa se implementeze microprocesorul RISC-V complet ce contine cele 5 stagii de pipeline si cele 2 module pentru detectia hazardurilor, respectiv unitatea de forwarding:

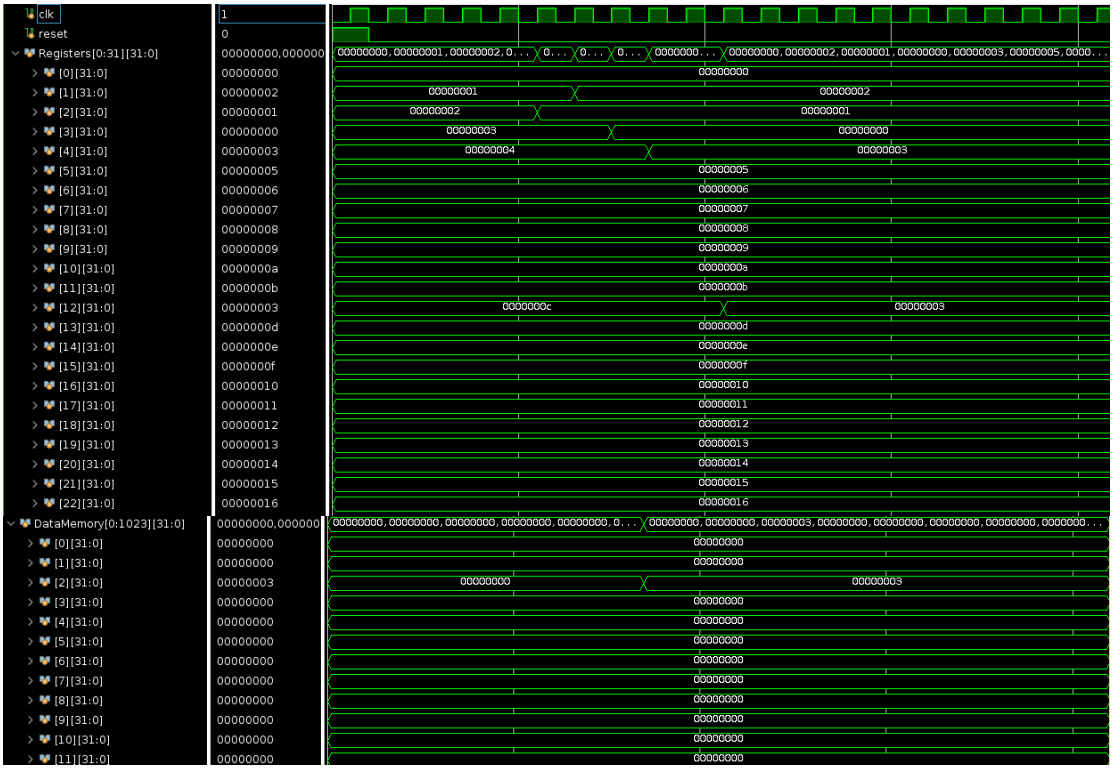
```
module RISC_V(input clk,   //semnalul de ceas global
              input reset, //semnalul de reset global

              output [31:0] PC_EX,         //adresa PC in etapa EX
              output [31:0] ALU_OUT_EX,    //valoarea calculata de ALE in etapa EX
              output [31:0] PC_MEM,        //adresa de salt calculata
              output PCSrc,                //semnal de selectie pentru PC
              output [31:0] DATA_MEMORY_MEM, //valoarea citita din memoria de date in MEM
              output [31:0] ALU_DATA_WB,     //valoarea finala scrisa in etapa WB
              output [1:0] forwardA, forwardB, //semnalele de forwarding
              output pipeline_stall            //semnal de stall la detectia de hazarduri
             );
```

Pentru simulare se va adauga bancul de registri si memoria de date in fereastra pentru formele de unda in urmatorul fel:

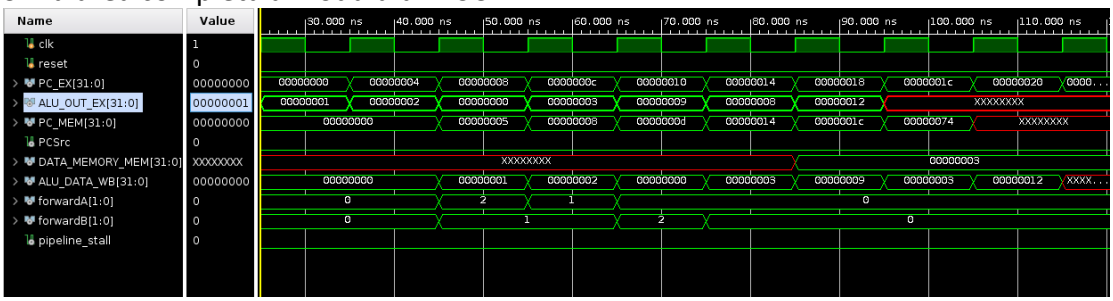




Simularea se va face pe baza codului assembly folosit in lucrarea precedenta, iar valorile finale rezultate in bancul de registri, respectiv memoria de date vor fi urmatoarele:



Simularea completa a modulului RISC-V:



Bibliografie:

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