Organizarea Calculatoarelor
LABORATOR 6 – MIPS PC+ROM

SCOPUL LUCRĂRII

Se va proiecta procesorul MIPS redus, conform Hennessy şi Patterson.

Chestiuni teoretice

Schema completă a procesorului MIPS redus este prezentată în figura 1

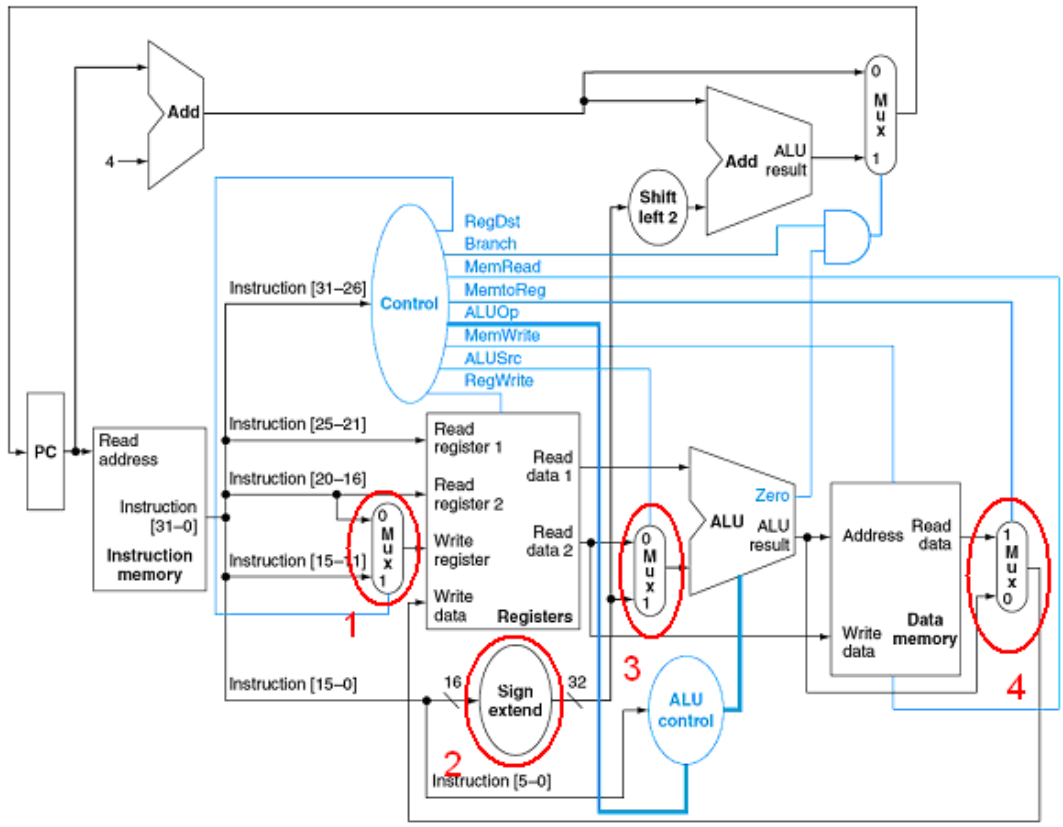


figura 1

La această schemă se va ajunge pe parcursul a 2-3 laboratoare.

Pasul 1: Crearea proiectului.

Creaţi proiectul ISE de tip schematic cu numele **mips**. Apoi adăugaţi la proiect un nou fişier de tip schematic cu numele **mips**. Acest fişier va conţine schema bloc a procesorului şi se va afla în vârful ierarhiei.

Pasul 2: Crearea numărătorului de program

Mai întâi se va implementa porţiunea din MIPS corespunzătoare numărătorului de program ca în figura următoare:

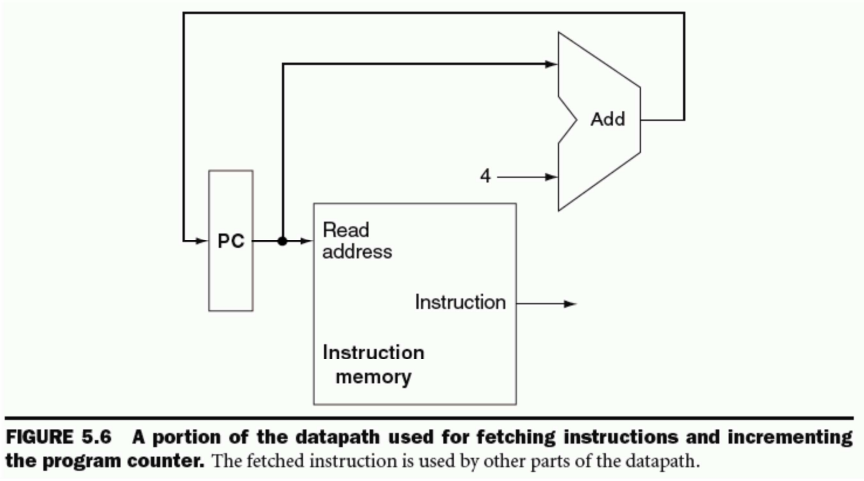


FIGURE 5.6 A portion of the datapath used for fetching instructions and incrementing the program counter. The fetched instruction is used by other parts of the datapath.

În figura 5.6 din Hennessy PC este un registru paralel-paralel. Înscrierea se face pe frontul ridicător al semnalului de ceas. Acest semnal se va numi **Clk**. Pentru a nu supraîncărca desenul, Clk nu mai este figurat, dar în implementarea noastră trebuie să apară. Pentru a implementa PC, procedaţi după cum urmează:

1. Adăugaţi la proiect un nou fişier sursă de tip VHDL. Acesta se va numi **ProgCnt**.
2. Interfaţa modulului este prezentată în continuare. Pentru a genera automat semnalele de la nivelul entităţii, urmaţi indicaţiile din laboratorul 5, pasul 2, figura 2.

```
entity ProgCnt is
   Port(
      Clk      : in  STD_LOGIC;
      New_PC   : in  STD_LOGIC_VECTOR (31 downto 0);
      PC       : out STD_LOGIC_VECTOR (31 downto 0):= x"0000_0000"
   );
end ProgCnt;
```

Declaraţia:

```
PC       : out STD_LOGIC_VECTOR (31 downto 0):= x"0000_0000"
```

precizează că PC este un semnal de tip vector cu elemente de tip std_logic iar valoarea sa iniţială este ‚0' pentru toate elementele. În VHDL **x** are rolul lui **0x** din C. Valoarea iniţială este valoare semnalului după configurarea FPGA. **Este foarte important să precizaţi valoarea iniţială!**

La nivelul arhitecturii trebuie specificat că PC va memora valoarea lui New_PC pe frontul ridicător al lui Clk. Deşi descrierea unui registru paralel-paralel a fost făcută în cursul de analiză şi sinteză a circuitelor digitale, este posibil să nu vă mai aduceţi aminte cum. Cei care îşi aduc aminte, vor proceda conform descrierii din curs. Pentru ceilalţi, se va prezenta o metodă mai puţin „ortodoxă".

În laboratorul precedent s-a prezentat declaraţia VHDL **conditional signal assignment** şi s-a spus că este echivalentă cu **if-elsif-elsif-else** din C. Această declaraţie a fost folosită în laboratorul precedent pentru specificarea multiplexoarelor BNB.

La fel ca în C, ramura **else** nu este obligatorie! Dacă lipseşte, interpretarea este aceeaşi ca în C: semnalul trebuie să-şi păstreze valoarea. Aceasta este cheia pentru implementarea elementelor de memorie.

Dacă scriem:

```
Q <= D when rising_edge(Clk);
```

generăm un element de memorie de tip bistabil.

Folosiţi această idee pentru descrierea registrului paralel-paralel.

Verificaţi sintaxă!

Pasul 3: Crearea noi valorii a PC.

Deoarece la MIPS cuvântul instrucţiune are lungimea de 4 octeţi, PC va avea valorile 0,4,8…. Aceste valori se obţin prin operaţia PC <= PC+4. Cu alte cuvinte este nevoie de un sumator. Acesta este prezentat în figura 5.6 Hennessy. Sumatorul nu se va implementa în acelaşi fişier cu registrul PC deoarece modul de actualizare a PC este mai complicat (vezi figura 1). Pentru a actualizarea PC, procedaţi după cum urmează:

1. Adăugaţi la proiect un nou fişier sursă de tip VHDL. Acesta se va numi **PC_Update.**
2. Interfaţa modulului este prezentată în continuare. Pentru a genera automat semnalele de la nivelul entităţii, urmaţi indicaţiile din laboratorul 5, pasul 2, figura 2.

```
entity PC_Update is
    Port (
      PC        : in  STD_LOGIC_VECTOR (31 downto 0);
      New_PC    : out STD_LOGIC_VECTOR (31 downto 0)
    );
end PC_Update;
```

3. La nivelul arhitecturii, specificaţi valoarea lui New_PC ca PC+4, salvaţi şi verificaţi sintaxa.

Pasul 4: Desenarea celor două blocuri .

1. Creaţi simbolurile pentru cele două module create anterior. Modificaţi simbolul pentru **PC_Update** pentru ca acesta să arate ca în figura 2.
2. Conectaţi cele două blocuri. Va trebui să ajungeţi la schema din figura 2.

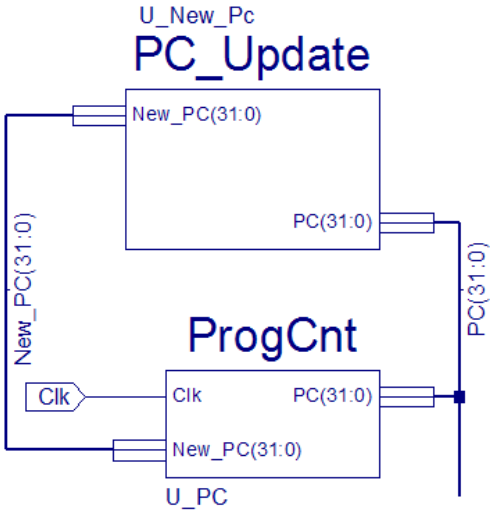


figura 2

Simbolurile **ProgCnt** şi **PC_Update** prezintă terminalele vectoriale **PC(31:0)** şi **New_PC(31:0)**. În general conectarea **directă** a terminalelor vectoriale se face după două reguli:

a) Terminalele care se conectează trebuie să aibă aceeaşi dimensiune.

b) Conectarea se face **de la stânga la dreapta**, indiferent de **direcţia** pinilor care se conectează şi de **limitele** acestora.

De exemplu dacă se conectează terminalul vectorial **EX1(3:7)** cu **EX2(5:1)** se vor realiza următoarele conexiuni scalare, clasice:

```
EX1(3)      EX1(4)      EX1(5)      EX1(6)      EX1(7)
  ↕           ↕           ↕           ↕           ↕
EX2(5)      EX2(4)      EX2(3)      EX2(2)      EX2(1)
```

Magistrala prin care se face conexiunea va primi un nume implicit în momentul desenării şi va avea indecşi lui **EX1** sau **EX2**, în funcţie de poziţia de pe planşa de desenare. Oricum, numele implicit al magistralei poate fi ulterior schimbat. Este obligatoriu însă ca noul nume să specifice un număr de elemente egal cu numărul de elemente al pinilor pe care îi conectează.

De exemplu **EX1** se poate conecta cu **EX2** prin intermediul magistralei **BUSA(0:4)** astfel:

```
EX1(3)      EX1(4)      EX1(5)      EX1(6)      EX1(7)
  ↕BUSA(0)    ↕BUSA(1)    ↕BUSA(2)    ↕BUSA(3)    ↕ BUSA(4)
EX2(5)      EX2(4)      EX2(3)      EX2(2)      EX2(1)
```

sau prin intermediul magistralei **BUSB(9:5)**:

```
EX1(3)      EX1(4)      EX1(5)      EX1(6)      EX1(7)
    ↕BUSB(9)    ↕BUSB(8)    ↕ BUSA(7)    ↕ BUSA(6)    ↕ BUSA(5)
EX2(5)      EX2(4)      EX2(3)      EX2(2)      EX2(1)
```

Regula de conectare este întotdeauna de la stânga la dreapta!

3. Denumiţi magistralele ca în figura 2. Vom avea nevoie de aceste nume în simulare
4. In figura 2 se observă că sub fiecare modul apare un nume: **U_PC** pentru **ProgCnt** şi **U_New_PC** pentru **PC_Update**. Aceste nume se numesc referinţe şi sunt necesare pentru a diferenţia obiectele de acelaşi fel. De exemplu, dacă într-un proiect avem 3 porţi AND2, cum putem să referim una dintre ele fără să apară confuzii? Diferenţierea se face cu ajutorul unui nume suplimentar. De obicei acest nume începe cu **U** de la **Unit** sau IC de la Integrated Circuit.

   La adăugarea unui nou modul în schemă acesta primeşte automat o referinţă de tip *XLNX_no*. Această referinţă este invizibilă.

   Referinţele modulelor apar în simulare. Dacă toate referinţele ar fi de tip XLNX_*no*, ar fi foarte greu de identificat modulele. Din acest motiv referinţele se vor modifică. Pentru a modifica referinţa unui modul faceţi clic dreapta pe respectivul modul şi din meniul contextual ce va apare alegeţi **Object Properties**. Va apare o fereastră de dialog ca în **Error! Reference source not found.**.

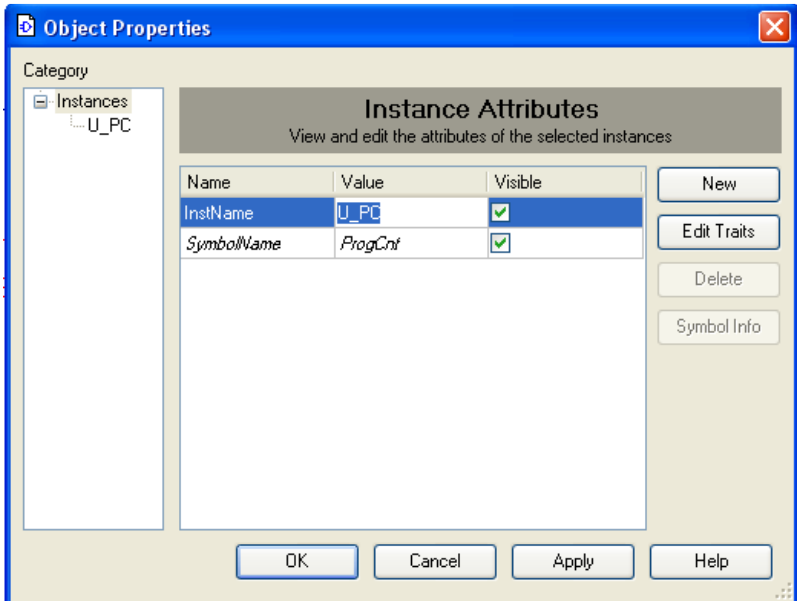


figura 3

Schimbaţi referinţele celor două module, faceţi-le vizibile şi poziţionaţi-le convenabil pe schemă.

Pasul 5: Adăugarea memoriei de program.

Pentru a adăuga memoria de program vom folosi modelele de generare incluse în ISE. Pentru a accesa aceste modele, apăsaţi butonul **Language Templates**. Acesta este încercuit cu roşu în figura 4. Dacă acest buton nu este vizibil, activaţi bara **Language templates** (View → Toolbars).

Apoi selectaţi **ROM Type Declaration**, ca în figura 5:

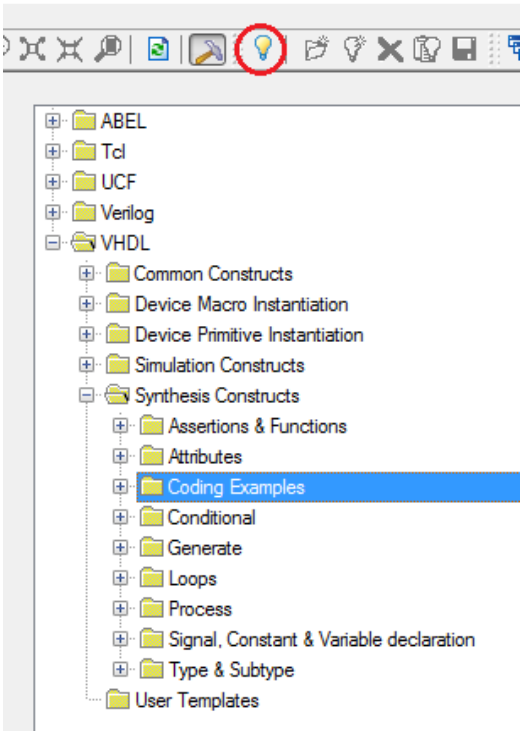


figura 4

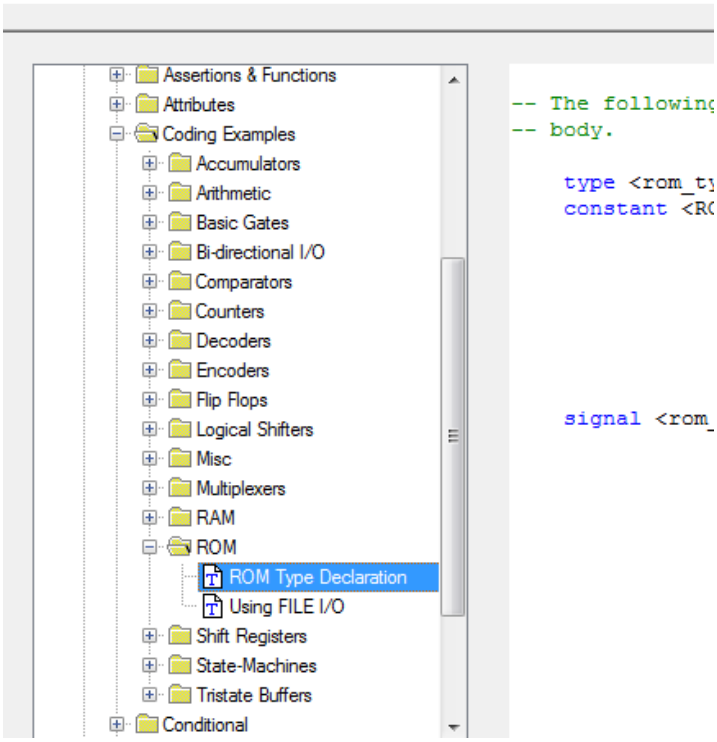


figura 5

Descrierea memoriei ROM ce urmează este făcută după indicaţiile din partea dreaptă din figura 5.

Vom genera o memorie cu 32 de locaţii, fiecare locaţie având 32 de biţi. Astfel memoria va avea organizarea 32x32. În scop didactic 32 de locaţii sunt suficiente.

Pentru a adresa 32 de locaţii este nevoie de 5 terminale de adresă. Conţinutul locaţiei de la adresa specificată prin intermediul terminalelor de adresă devine disponibilă către alte module prin intermediul a 32 de terminale de date. Cele 5 adrese sunt intrări iar cele 32 de date sunt ieşiri.

Pentru adăugarea memoriei de program, procedaţi după cum urmează:

1. Adăugaţi la proiect fişierul **ROM32x32** de tip vhdl. Semnalele pentru adresă se vor numi **Addr** iar cele pentru date, **Data**. Entitatea memoriei ROM este prezentată în continuare. Pentru a genera automat semnalele de la nivelul entităţii, urmaţi indicaţiile din laboratorul 5, pasul 2, figura 2.

```
entity ROM32x32 is
   Port (
      Addr  : in  STD_LOGIC_VECTOR (4 downto 0);
      Data  : out STD_LOGIC_VECTOR (31 downto 0));
end ROM32x32;
```

   Adăugaţi în interfaţa vizuală semnalele **Addr** şi **Data**, astfel încât să obţineţi entitatea de mai sus.

2. O memorie în VHDL seamănă cu un vector unidimensional în C. În C pentru a declara un vector de întregi trebuie se scriem:

```
int intvect[8];
```

Nu este nevoie de nici o declaraţie suplimentară pentru vectori.

În VHDL situaţia este diferită: orice tip de date diferit de tipurile de bază trebuie declarat cu **type**. **type** este în VHDL echivalentul lui **typedef** din C.

O memorie în VHDL este un vector ale cărui elemente sunt de tip scalar sau vectorial. În cazul memoriei noastre cu organizarea 32x32 este nevoie să declarăm un nou tip de date de tip vector (array) cu 32 de elemente, fiecare element fiind la rândul său un **std_logic_vector** cu 32 de elemente. Această declaraţie de tip este marcată cu 1 în secvenţă de cod care urmează. Scrieți în fişierul dumneavoastră această declaraţie.

```
architecture ROM32x32_arch of ROM32x32 is
   type tROM is array (0 to 31) of std_logic_vector(31 downto 0);  1

   constant ROM : tROM :=(
                  --test
                  x"00000001",  --0
                  x"00000002",  --1
                  x"00000004",  --2
                  x"00000008",  --3
                  x"00000010",  --4
                  x"00000020",  --5      2
                  x"00000040",  --6
                  x"00000080",  --7
                  x"00000100",  --8
                  x"00000200",  --9
                  x"00000400",  --10
                  ..........    --11
                  x"40000000",  --30
                  x"80000000"   --31
                  );
begin

   Data  <= ROM(conv_integer(Addr));   3
end ROM32x32_arch;
```

figura 6

Declararea unui obiect de tipul **tROM** se face cu:

```
constant ROM: tROM :=(valori_iniţiale);
```

constant în VHDL este echivalentul lui **const** din C. Declaraţia de mai sus generează obiectul constant numit **ROM** de tipul **tROM**. S-a folosit clasa constant deoarece conţinutul unei memorii ROM nu se poate modifica.

La fel ca în C, vectorii se pot iniţializa la declarare. Pentru început vom iniţializa memoria cu un patern uşor de recunoscut în simulare şi anume „walking 1”. Acest patern se obţine dacă fiecare tetradă ia succesiv valorile 1, 2, 4 şi 8. Scrieți în fişierul dumneavoastră declaraţia marcată cu 2 în secvență de cod anterioară. **Adăugaţi liniile lipsă!**

La nivelul arhitecturii, după **begin**, este descrisă comportarea obiectului ROM. În cazul citirii orice memorie generează la ieşire conţinutul locaţiei selectată de adresă. La modul general ar trebui scris:

```
ieşire=ROM(adresă);
```

În VHDL, în cazul memorie noastre, această declaraţie ar deveni:

```
Data  <= ROM(Addr);
```

Dar indexul oricărui vector trebuie să fie de tip întreg iar **Addr este de tip vector**. Pentru a transforma Addr din vector în întreg se foloseşte funcţia *conv_integer*. Această funcţie primeşte ca argument un obiect de tip **std_logic_vector** şi îl converteşte la **întreg**. Elementul cel mai din dreapta al vectorului este considerat LSB iar mai din stânga este MSB. Gama şi indecşii vectorului nu au importanţă.

Consideraţi următoarele declaraţii:

```
signal v1 : std_logic_vector(3 downto 0);
signal v2 : std_logic_vector(4 to 7);
…..
v1    <= „1100”;
v2    <= „1100”;
```

Deşi gamele şi indecşii minimi şi maximi pentru cei doi vectori diferă, avem:

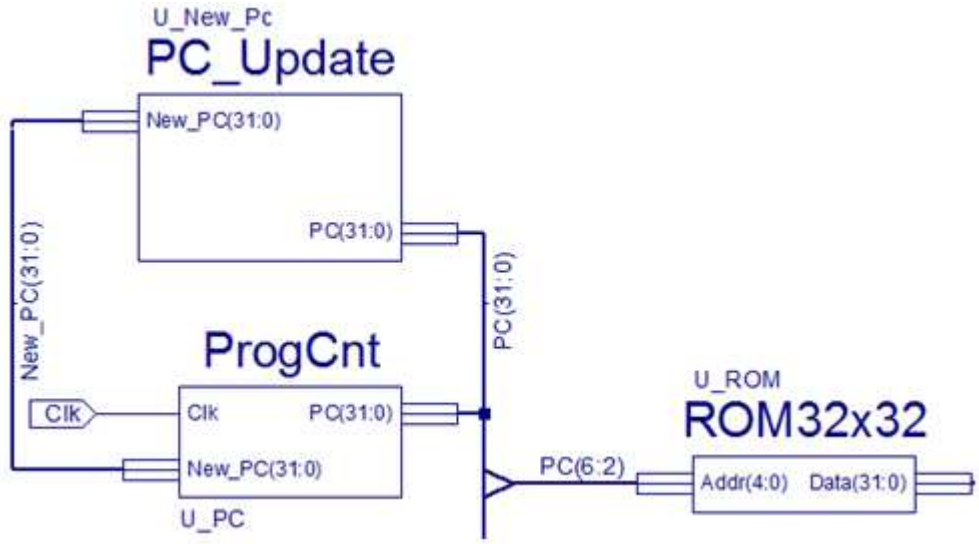
```
conv_integer(v1) = conv_integer(v2) = 12
```

Aplicând `conv_integer` asupra lui Addr, descrierea comportamentului memoriei ROM devine:

```
Data <= ROM( conv_integer(Addr) );
```

Scrieţi în fişierul dumneavoastră declaraţia marcată cu 3 în secvenţă de cod anterioară.

3. Creaţi simbolul asociat entităţii ROM32x32 din fişierul cu acelaşi nume.
4. Plasaţi pe schemă memoria ROM32x32 şi conectaţi-o ca în figura următoare:



5. Nu uitaţi să adăugaţi referinţa memoriei ROM.
5. Salvaţi şi verificaţi schema.

Pasul 6: Multiplexorului rt-rd (marcaj 1 în figura 1).

Chestiuni teoretice

MIPS este un procesor cu lungime fixă a instrucţiunii. Lungimea instrucţiunii este de 32 de biţi. Există 3 moduri de codificare a instrucţiunii. În continuare se vor prezenta modurile **R** şi **I**.

Modul **R** este folosit în principal pentru instrucţiunile cu trei registre. Exemplu tipic este:

ADD rd, rs, rt

Instrucţiunea adună conţinutul registrului *rs* cu cel al registrului *rt* şi depune rezultatul în registrul *rd*. Instrucţiunea se codifică după cum urmează:

Poziţie	31 26	25 21	20 16	15 11	10 6	5 0
Lungime	6	5	5	5	5	6 biţi
	Opcode	**rs**	**rt**	**rd**	**Shamt**	**Function**

Codul instrucţiunii se obţine din câmpurilor **Opcode** şi **Function**. În prima fază instrucţiunile de tip **R** care se vor implementa sunt **ADD**, **SUB**, **AND** şi **OR**.

Modul **I** este folosit în principal pentru instrucţiunile care necesită specificarea unei constante imediate. Codificarea I este următoarea:

Poziţie	31 26	25 21	20 16	15 0
Lungime	6	5	5	16 biţi
	Opcode	**rs**	**rt**	**imm16**

Instrucţiunile de tip I ce se vor implementa sunt **LW** şi **SW**. Ambele instrucţiuni calculează adresa locaţiei de memorie cu care se face transferul cu formula GPR(rs)+SE(imm16). SE înseamnă Sign Extended.

Dacă punem cele două formate unul sub altul obţinem:

Table 1

Poziţie	31 26	25 21	20 16	15 11		0
R	**Opcode**	**rs**	**rt**	**rd**	**Shamt**	**Function**
I	**Opcode**	**rs**	**rt**	**imm16**		

Se observă că în ambele formate câmpurile **Opcode**, **rs** şi **rt** rămân neschimbate. Diferenţa apare la registrul destinaţie: în cazul formatului R acesta este câmpul **rd** al instrucţiunii iar în cazul formatului I este câmpul **rt**. Semnalul care va selecta registrul destinaţie este generat de câmpul **rd** pentru instrucţiunile de tip R şi de câmpul **rt** pentru instrucţiunile I. Acest semnal va fi generat cu un MUX vectorial cu 2 intrări. Intrările sunt vectori cu 5 elemente.

Creaţi un MUX2 pentru vectori de 5 elemente, conform indicaţiilor din laboratoarele precedente. Fişierul vhd care conţine descrierea multiplexorului se va numi **MUX2V5.vhd** iar entitatea asociată este următoarea:

```
entity MUX2V5 is
   port(
      I0    : in  std_logic_vector(4 downto 0);
      I1    : in  std_logic_vector(4 downto 0);
      Sel   : in  std_logic;
      Y     : out std_logic_vector(4 downto 0)
   );
end MUX2V5;
```

Folosind „**conditional signal assignement**” **adăugați** la nivelul arhitecturii descrierea multiplexorului. Verificaţi sintaxa şi creaţi simbolul pentru acest multiplexor.

Adăugaţi pe schema MIPS multiplexorul **rt-rd** conform figurii următoare:

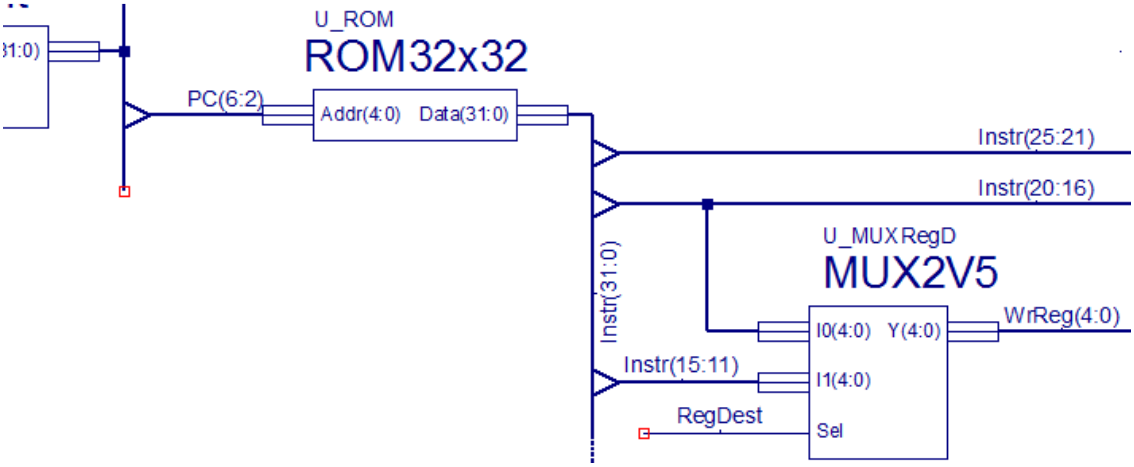


figura 7

Dacă este necesar **editaţi simbolul** multiplexorului pentru ca acesta să **arate** exact ca în figură. Atenţie la ordinea intrărilor de date. De sus în jos ordinea este I0 şi apoi I1. **Notaţi** toate conexiunile ca în figura 7.

Verificaţi schema. Deoarece schema nu este completă, vor apare două avertismente:

```
Start DRC ...
WARNING:DesignEntry:216 - mips.sch Net 'RegDest' is connected to load pins and/or I/O markers, but not connected to any source pin or I/O marker.

WARNING:DesignEntry:221 - mips.sch Bus 'WrReg(4:0)' is connected to source pins and/or I/O markers, but not connected to any load pins or I/O marker.

Totally, 0 error(s) and 2 warning(s) are detected
```

Dacă apar erori sau mai multe avertismente decât cele două de mai sus, corectaţi-le. Dacă nu reuşiţi, chemaţi profesorul.

Pasul 7: Simularea schemei parţiale.

Adăugaţi la proiect un fişier de tip tbw. Deoarece proiectul este de tip secvenţial sincron, selectaţi **Rising Edge** (de obicei este deja selectat) şi faceţi setările din figura 8. **Selectaţi simulare de tip Behavioral**, pentru că schema nu este încă sintetizabilă.

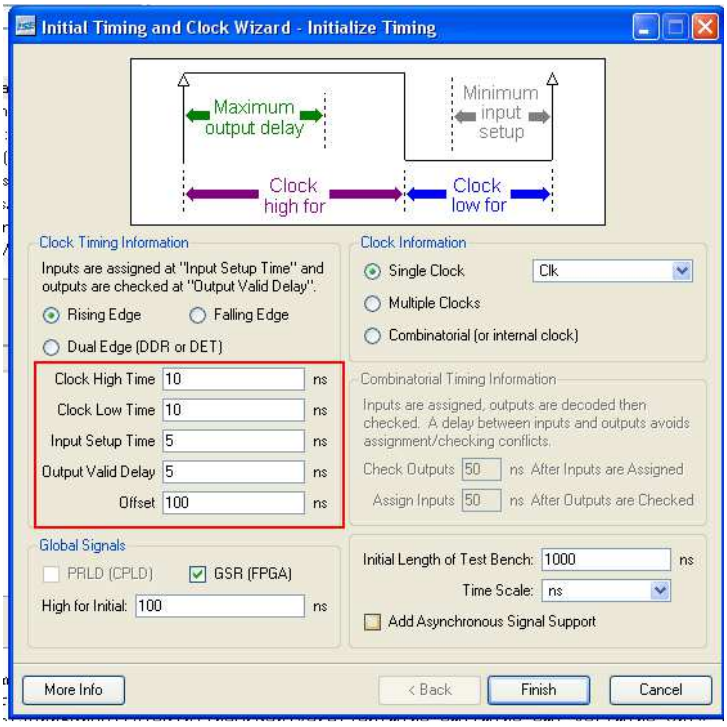


figura 8

Este puţin probabil ca un proiect să funcţioneze de prima dată. În cazul în care ceva nu funcţionează, cum este cazul de cele mai multe ori, este necesar să se examineze comportamentul semnalelor interne definite la nivelul arhitecturii. În mod implicit Modelsim afişează în fereastra Wave doar semnalele de interfaţă ale modulului din vârful ierarhiei.

În momentul de faţă schema are o singura intrare, clk, şi **nici o ieşire**. În acest caz verificare funcţionalităţii se poate face numai prin examinarea semnalelor interne.

Pentru a vizualiza formele de undă ale semnalelor interne se procedează ca în figura 9:

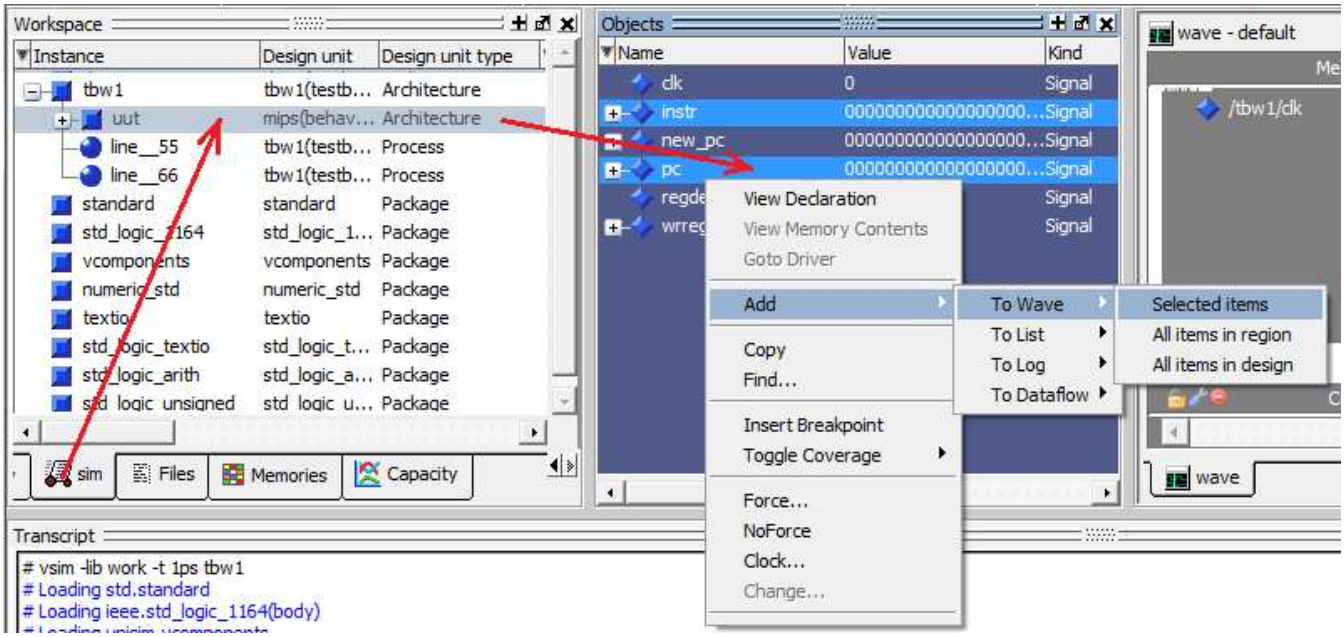


figura 9

1. Dacă nu este deja selectat, se selectează câmpul tab sim
2. In fereastra Workspace se selectează instanţa uut. În orice proiect Modelsim entitatea din vârful ierarhiei se numeşte uut – Unit Under Test. În acest caz uut este mips.
3. Ca urmare a selectării uut, în fereastra Objects se afişează toate semnalele definite atât la nivelul entităţii cât şi la nivelul arhitecturii.
4. În fereastra Objects se selectează toate semnale care se doresc afişate în fereastra Wave.
5. Se face click dreapta oriunde în zona selectată. Din meniul contextual apărut se poate selecta opţiunea Copy sau opţiunea Add →To Wave → Selected Items.
6. Dacă s-a selectat opţiunea Add →To Wave → Selected Items, semnalele selectate se adaugă în fereastra Wave, la sfârşit. Dacă s-a selectat opţiunea Copy, se va afişa fereastra Wave, se va selecta un semnal existent, se va face clic dreapta şi din meniul contextual apărut se va selecta opţiunea Paste. Semnalele selectate anterior se vor insera deasupra semnalului existent.

7. După adăugarea semnalelor PC şi Instr configuraţia semnalelor va fi cea din figura 10.

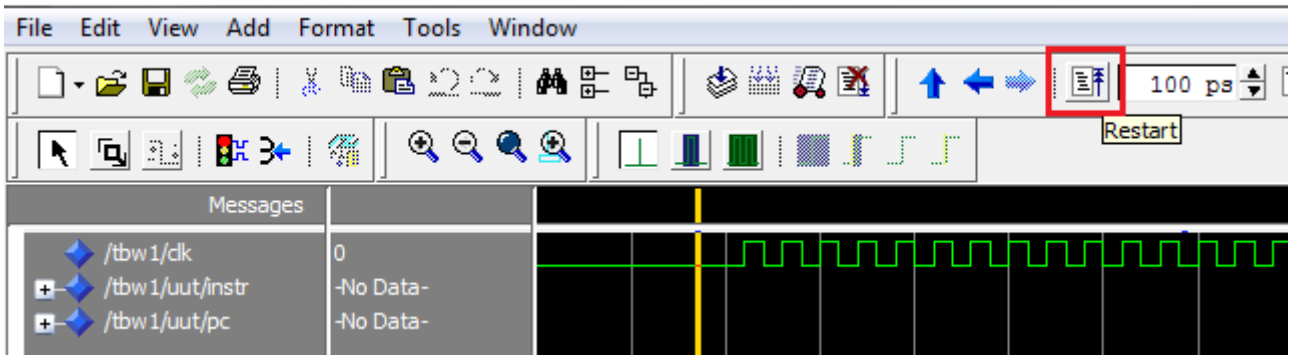
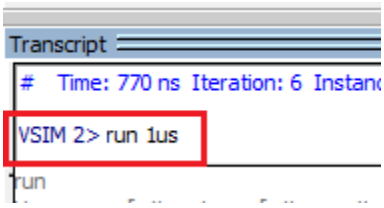


figura 10

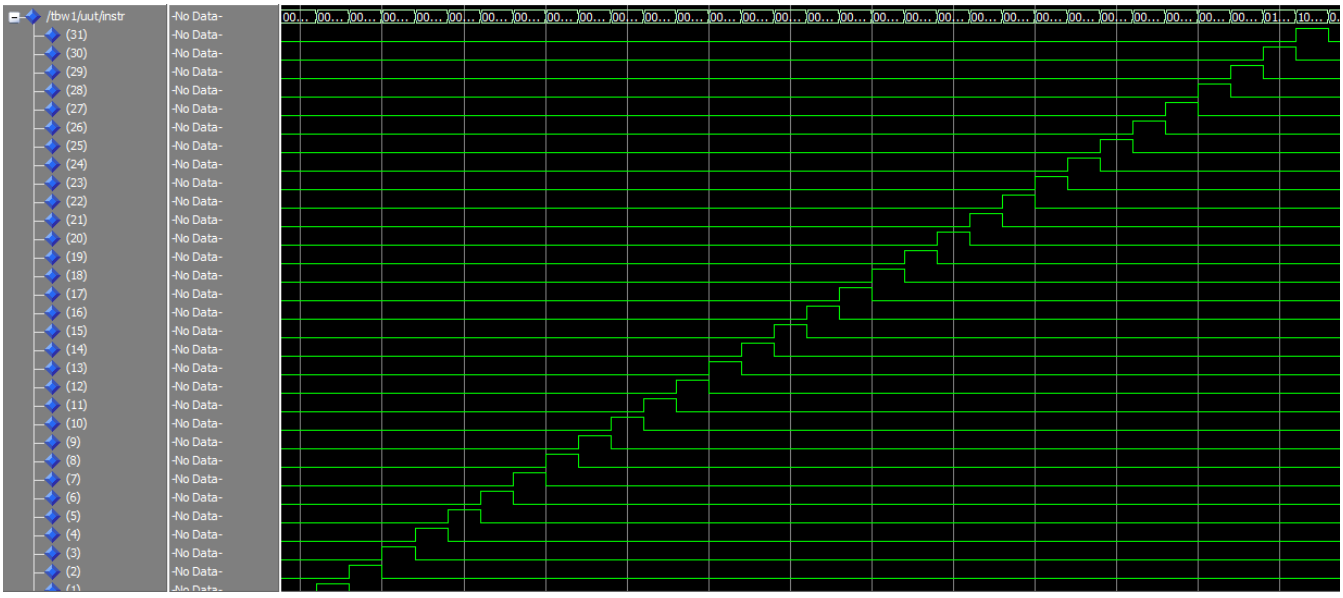
8. Folosind Drag&drop rearanjaţi semnalele astfel încât ordinea acestora să fie Clk, PC şi Instr.
9. Deoarece nu există date disponibile pentru semnale adăugate, apăsaţi butonul Restart, marcat cu roşu în figura 10 iar apoi rulaţi o nouă simulare.

   Pentru a rula o nouă simulare **tastaţi** run 1 us în fereastra principală Modelsim:



Faceţi simularea.

Semnalul PC este un numărător care numără din 4 în 4. La reset starea iniţială este 0; apoi, pe frontul ridicător al lui clk, valorile lui pc vor fi 4, 8, c, 10, etc. Numărătorul de program pc este conectat la adresele memoriei de program aşa că pe ieşirile de date ale memoriei vor apare în ordine valorile din figura 6. Deoarece patternul înscris în memoria de program este un „walking one”, semnalul **instr** va avea alura din figura următoare:



Când consideraţi că funcţionează, chemaţi profesorul pentru validare!