Organizarea Calculatoarelor
LABORATOR 8 – Simularea MIPS

SCOPUL LUCRĂRII

Se va simula procesorul MIPS creat în laboratoarele precedente.

Pasul 1: Generarea codului maşină pentru programul de test

Verificarea funcţionării MIPS se va face cu programul **test1** scris în limbaj de asamblare MIPS:

```
lw    $2, 0x40($0)      # INW0=0xAAAA_AAAB
lw    $3, 0x44($0)      # INW1=0x5555_5555
add   $4, $2, $3
sub   $5, $2, $3
and   $6, $2, $3
or    $7, $2, $3
sw    $2, 0x48($0)
sw    $3, 0x48($0)
sw    $4, 0x48($0)
sw    $5, 0x48($0)
sw    $6, 0x48($0)
sw    $7, 0x48($0)
```

Programul citeşte locaţiile de memorie de la adresele **0x40** şi **0x44**. Aşa cum s-a precizat în laboratorul anterior (la pasul 11) locaţiile de la aceste adrese sunt speciale: ele pot fi doar citite. Instrucţiunea

```
lw    $2, 0x40($0)
```

face ca în registrul 2 să se înscrie valoarea curentă a lui **INW0**, iar instrucţiunea

```
lw    $3, 0x44($0)
```

face ca în registrul 3 să se înscrie valoarea curentă a lui **INW1**.

În continuare vom presupune că **`INW0=0xAAAA_AAAB`** şi **`INW1=0x5555_5555`**, dar se poate seta orice altă valoare. Valorile de 0xAAAA_AAAB şi 0x5555_5555 au fost alese pentru ca rezultatele operaţiilor aritmetice şi logice se calculează uşor manual.

Instrucţiunile următoare fac adunare, scădere, AND şi OR între registrele 2 şi 3 şi depun rezultatele în registrele 4, 5, 6 şi 7. În final conţinutul registrelor 2..7 sunt trimise în locaţia specială de la adresa **0x48** şi pot fi observate în exterior prin intermediul lui **OUTW0**.

Următoarea operaţie constă în obţinerea codului maşină pentru programul de mai sus. Codificarea acestor instrucţiuni se face conform ISA MIPS32 (http://www.cs.ucv.ro/~lemeni/MD00086-2B-MIPS32BIS-AFP-03.02.pdf). În tabelul următor este prezentată codificarea instrucţiunilor **LW** şi **SW**, extrase din ISA MIPS:

Format I

	6	5	5	16
LW	0x23 (OPCODE)	Rs	Rt	offset
SW	0x2b (OPCODE)	Rs	Rt	offset

Instrucţiunea

```
lw    $2, 0x40($0)
```

se codifică conform tabelului următor:

	6	5	5	16
	OPCODE	Rs	Rt	offset
`lw $2,0x40($0)`	0x23	0x0	0x2	0x40
0x8c020040	**100011**	**00000**	**00010**	**0x0040**

În concluzie codul maşină pentru **`lw $2,0x40($0`**) este:

1000_1100_ 0000_ 0010&0x0040=0x8c02_0040.

În continuare ar trebui generat manual codul maşină şi pentru celelalte instrucţiuni din programul de test.

Deşi translatarea manuală a programelor scrise în asamblare este foarte instructivă, este în acelaşi timp generatoare de erori. Din acest motiv vom genera codul maşină automat prin intermediul programului MARS. În acest sens vom proceda după cum urmează:

1. Creaţi un folder numit **asm**. În acest folder creaţi un fişier text denumit **test1.asm**.
2. Copiaţi în fişierul **test1.asm** programul de verificare **test1** prezentat la începutul pasului 1.
3. Extrageţi programul **Mars_4_4.jar** din arhiva lucrării de laborator şi lansaţi-l în execuţie. MARS – **M**IPS **A**ssembler and **R**untime **S**imulator – este un asamblor şi simulator pentru MIPS.
4. Din meniul **File** al programului MARS selectaţi **Open** şi alegeţi fişierul **test1.asm** creat anterior**.**
5. Pentru ca adresa de început a codului maşină ce se va genera de asamblor să fie zero, din meniul **Settings** selectaţi opţiunea **Memory Configuration…**. Va apare fereastra următoare:

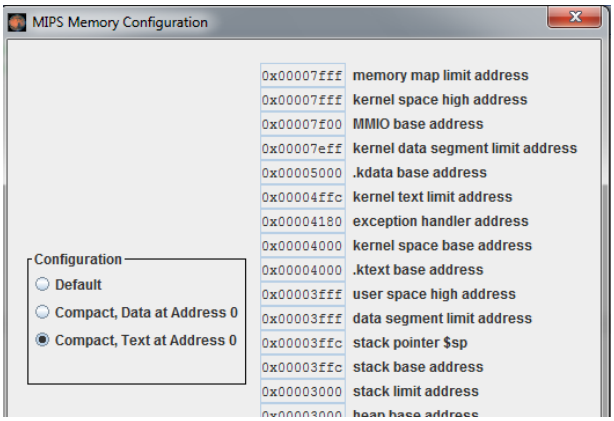


figura 1

Selectaţi configuraţia **Compact, Text at Address 0** şi apoi apăsaţi butonul **Apply and Close**.

6. Asamblaţi acest fişier cu **Run → Assemble**. După executarea acestei operaţii codul din fişierul sursă este asamblat iar codul maşină rezultat este pregătit pentru simulare. Veţi obţine situaţia din figura 2.

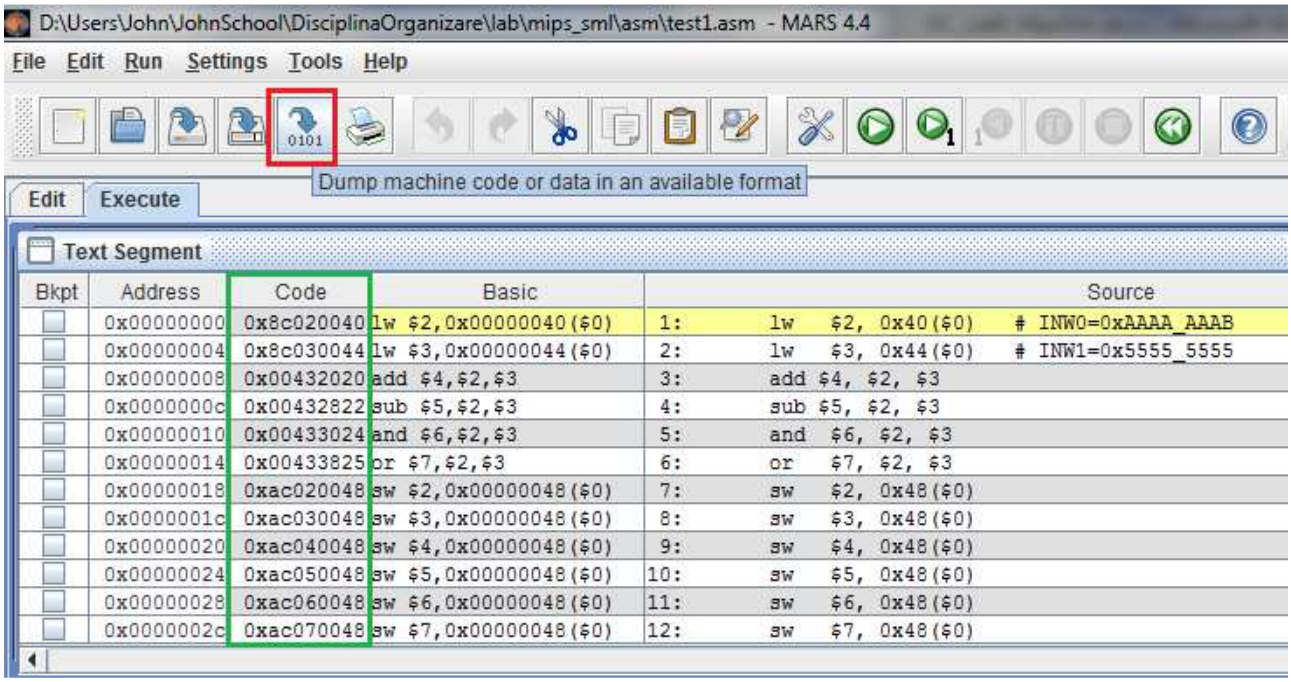
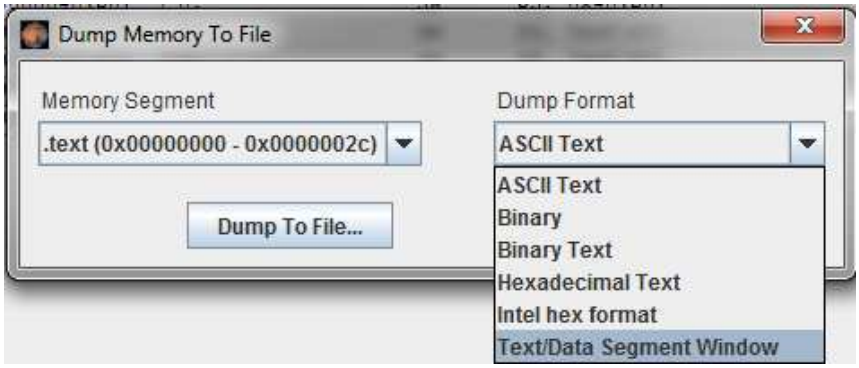


figura 2

7. În coloana **Code** (marcaj verde în figura 2) se află codul maşină pentru programul sursă din fişierul test1.asm. Pentru ca acest cod să fie executat de procesorul MIPS construit în cadrul acestui laborator, el trebuie să devină conţinutul memoriei de cod din fişierul ROM32x32. Mai exact coloana **Code** trebuie copiată în fişierul ROM32x32 astfel încât să devină conţinutul constantei ROM.

   În acest sens, ideal ar fi ca să putem selecta colana Code din fereastra MARS şi apoi să o copiem. Cum această operaţie nu este posibilă, mai întâi vom crea un fişier listing în care se va afla atât sursa cât şi codul maşină rezultat în urma asamblării.



Pentru aceasta apăsaţi butonul **Dump** marcat cu roşu în figura 2. Apoi, în fereastra de dialog din figura alăturată, selectaţi **Text/Data Segment Window** şi apoi apăsaţi butonul **Dump to File…** şi salvaţi fişierul listing sub numele **test1.lst** (sau alt nume dacă acesta nu vă place).

8. În laboratoarele următoare se vor executa mai multe programe. Pentru a păstra toate aceste programe vom aloca un fişier pentru fiecare program.

   Faceţi o copie a fişierului **ROM32x32.vhd**. Denumiţi această copie **ROM32x32_test1.vhd**. în continuare vom copia codul maşină generat din fişierul **test1.lst** în fişierul **ROM32x32_test1.vhd**.

   Deschideţi proiectul mips. Editaţi fişierul **test1.lst** cu editorul de text din ISE. Pentru a edita fişierul **test1.lst** cu editorul de text din ISE selectaţi **File → Open → All files**. **NU adăugaţi** fişierul **test1.lst** la proiect, doar deschideţi-l. Apoi, folosind **modul coloană**, selectaţi coloana ce conţine codul maşină generat de MARS. Pentru a selecta o coloană de text procedaţi după cum urmează:

- Poziționați cursorul mouse-ului într-un colț al dreptunghiului ce va încadra coloana pe care vreți să o selectați.
- Țineți apăsată tasta ALT.
- Apoi apăsați butonul stânga al mouse-ului și țineți-l apăsat.

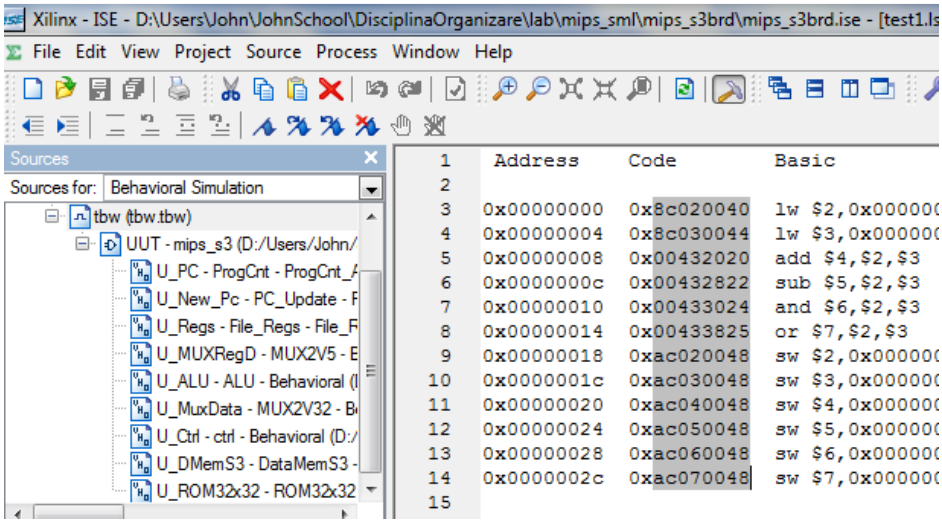


figura 3

- Cu ALT **apăsat** și cu butonul stânga al mouse-ului **apăsat** deplasați cursorul astfel încât să definiți un dreptunghi. Acest dreptunghi trebuie sa încadreze **exact** zona pe care vreți să o selectați. După marcare zona selectată va fi evidențiată prin fundal gri, ca figura 3.
- Eliberați butonul stânga al mouse-ului și tasta ALT, indiferent în ce ordine.
- Faceți clic dreapta pe zona gri și din meniul contextual alegeți **Copy**.

Apoi deschideți pentru editare fișierului **ROM32x32_test1.vhd** în ISE. Pentru a copia coloana selectată anterior procedați după cum urmează:

- **Poziționați** cursorul editorului de text (caret) ca în figura 4 a).
- Faceți **Paste** pentru a copia coloana ce conține codului mașină, marcată anterior pentru copiere. Veți obține situația din figura 4 b). Se observă că vechiul conținut a fost deplasat la dreapta. Coloana selectată a fost inserată în vechiul conținut fără a se adăuga linii noi, așa cum se întâmpla la o copiere non-coloană.
- Selectați în mod coloană vechiul conținut. Trebuie să obțineți situația din figura 4 c).

```
41   type tROM is array (0 to
42
43   constant ROM : tROM :=(
44      x"00000001",  --0
45      x"00000002",  --1
46      x"00000004",  --2
47      x"00000008",  --3
```

a)

```
constant ROM : tROM :=( -- fo
   x"8c02004000000001",  --0
   x"8c03004400000002",  --1
   x"0043202000000004",  --2
   x"0043282200000008",  --3
   x"0043302400500010",  --4
   x"0043382500000020",  --5
   x"ac02004800000040",  --6
   x"ac03004800000080",  --7
   x"ac04004800000100",  --8
   x"ac05004800000200",  --9
   x"ac06004800000400",  --10
   x"ac07004800000800",  --11
   x"00001000",  --12
   x"00002000",  --13
```

b)

```
constant ROM : tROM :=( -- fo
   x"8c020040 00000001",  --0
   x"8c030044 00000002",  --1
   x"00432020 00000004",  --2
   x"00432822 00000008",  --3
   x"00433024 00500010",  --4
   x"00433825 00000020",  --5
   x"ac020048 00000040",  --6
   x"ac030048 00000080",  --7
   x"ac040048 00000100",  --8
   x"ac050048 00000200",  --9
   x"ac060048 00000400",  --10
   x"ac070048 00000800",  --11
   x"00001000",  --12
   x"00002000",  --13
```

c)

```
constant ROM : tROM :=
   x"8c020040",   --0
   x"8c030044",   --1
   x"00432020",   --2
   x"00432822",   --3
   x"00433024",   --4
   x"00433825",   --5
   x"ac020048",   --6
   x"ac030048",   --7
   x"ac040048",   --8
   x"ac050048",   --9
   x"ac060048",   --10
   x"ac070048",   --11
   x"00001000",   --12
   x"00002000",   --13
   x"00004000",   --14
```

d)

```
constant ROM : tROM :=( -- for PC codemem test
   x"8c020040",   --0  lw   $2, 0x40($0)
   x"8c030044",   --1  lw   $3, 0x44($0)
   x"00432020",   --2  add  $4, $2, $3
   x"00432822",   --3  sub  $5, $2, $3
   x"00433024",   --4  and  $6, $2, $3
   x"00433825",   --5  or   $7, $2, $3
   x"ac020048",   --6  sw   $2, 0x48($0)
   x"ac030048",   --7  sw   $3, 0x48($0)
   x"ac040048",   --8  sw   $4, 0x48($0)
   x"ac050048",   --9  sw   $5, 0x48($0)
   x"ac060048",   --10 sw   $6, 0x48($0)
   x"ac070048",   --11 sw   $7, 0x48($0)
   x"00001000",   --12
   x"00002000",   --13
   x"00004000",   --14
```

e)

figura 4

- Ştergeţi coloana selectată anterior (cu DEL). Trebuie să obţineţi situaţia din figura 4 d).
- Tot cu ajutorul modului colană copiaţi programul sursă drept comentarii după adresa locaţiei ROM (--0, --1, --2….). Astfel vom şti ce program este stocat în ROM. **Întotdeauna copiaţi programul sursă drept comentarii.** Trebuie să obţineţi situaţia din figura 4 e).
- Scrieţi conținutul restului de locaţii cu codul instrucţiunii NOP, şi anume x”0000_0000”. Adică în locaţiile de la 13 la 31 acolo unde este 1, 2, 4 sau 8 scrieţi 0. Dacă nu facem această modificare procesorul va considera locaţiile cu 1, 2, 4, 8 coduri maşină şi le va executa. Efectele pot fi imprevizibile. Dacă am avea implementată o instrucţiune de salt, am termina codul cu salt la adresa zero sau salt pe el însuşi şi operația de la acest punct nu ar mai fi necesară. Dar nu avem încă instrucţiune de salt!

Pasul 2: Executarea manuală programului de test (pe hârtie)

Mai întâi trebuie completat un tabel în care să se înregistreze valorile memorate în cele 32 de registre generale pe durata executării programului de test. Va exista câte o coloană pentru fiecare registru şi locaţie de memorie modificate de programul de test. Deoarece programul de test modifică doar registrele 2 până la 7 coloanele se vor numi **$2**, **$3**,…, **$7.** În plus mai este necesară o coloană pentru locaţia de memorie de la adresa 0x48 (OUTW0) deoarece toate instrucţiunile *sw* scriu la această adresă.

Pe linii scriu în ordine instrucţiunile din programul de test. Prima linie corespunde valorilor memorate în registre imediat după iniţializarea (reset) procesorului; toate aceste valori sunt nedefinite. Rezultă următorul tabel:

Tabelul 1

	$2	$3	$4	$5	$6	$7	OUTW0
După Reset	xxxx_xxxx	xxxx_xxxx	xxxx_xxxx	xxxx_xxxx	xxxx_xxxx	xxxx_xxxx	xxxx_xxxx
lw $2, 0x40($0)	AAAA_AAAB	-//-	-//-	-//-	-//-	-//-	-//-
lw $3, 0x44($0)							
add $4, $2, $3							
sub $5, $2, $3							
and $6, $2, $3							
0r $7, $2, $3							
……etc.							

Descrierea instrucţiunilor se află în documentul **ISA MIPS** (şi în curs bineînţeles, dar cine citeşte

cursul…).

Prima instrucţiune din programul de test care se va executa este `lw $2, 0x40($0)`. Descrierea acestei instrucţiuni se află la pagina 152 în **ISA MIPS**. Instrucţiunea `lw $2, 0x40($0)` citeşte locaţia de memorie de la adresa calculată (modul de calcul se va detalia imediat) şi o scrie în registrul general 2.

Adresa locaţiei de memorie care va citi se calculează prin adunarea conţinutului registrului *rs* cu ofsetul extins ca semn. În cazul instrucţiunii `lw $2, 0x40($0)` ofsetul este 0x0040. După extinderea semnului se obţine valoarea SE(0x0040)=0x0000_0040. ALU va aduna SE(0x0040)=0x0000_0040 cu conţinutul registrului zero (care este întotdeauna zero). Rezultatul este 0x0000_0040+0x0000_0000=**0x0000_0040**.

Adresa locaţiei de memorie care va citi este 0x0000_0040. La această adresă se află locaţia specială INW0. Aşa cum s-a specificat la pasul 1, testul se va execută cu **`INW0=0xAAAA_AAAB`** şi **`INW1=0x5555_5555`**. Din acest motiv la ieşirea memoriei trebuie să apară valoarea lui INW0, adică **`0xAAAA_AAAB`.** Această valoare se va scrie în registrul 2.

În concluzie instrucţiunea `lw $2, 0x40($0)` scrie în registrul 2 valoarea 0xAAAA_AAAB. Această valoare se scrie în Tabelul 1 pe linia `lw $2, 0x40($0)` în dreptul registrului 2. Toate celelalte registre plus locaţia OUTW0 îşi păstrează valoarea. Păstrarea valorii anterioare se notează în tabel cu ---//---.

Completaţi Tabelul 1 pentru toate instrucţiunile din programul de test. Toate valorile se vor scrie în hexazecimal. Dacă nu ştiţi să faceţi calcule aritmetice şi logice, folosiţi programul *Calculator* din Windows, Accessories.

Când Tabelul 1 este completat, chemaţi profesorul pentru înregistrarea progresului.

Pasul 3: Simularea MIPS

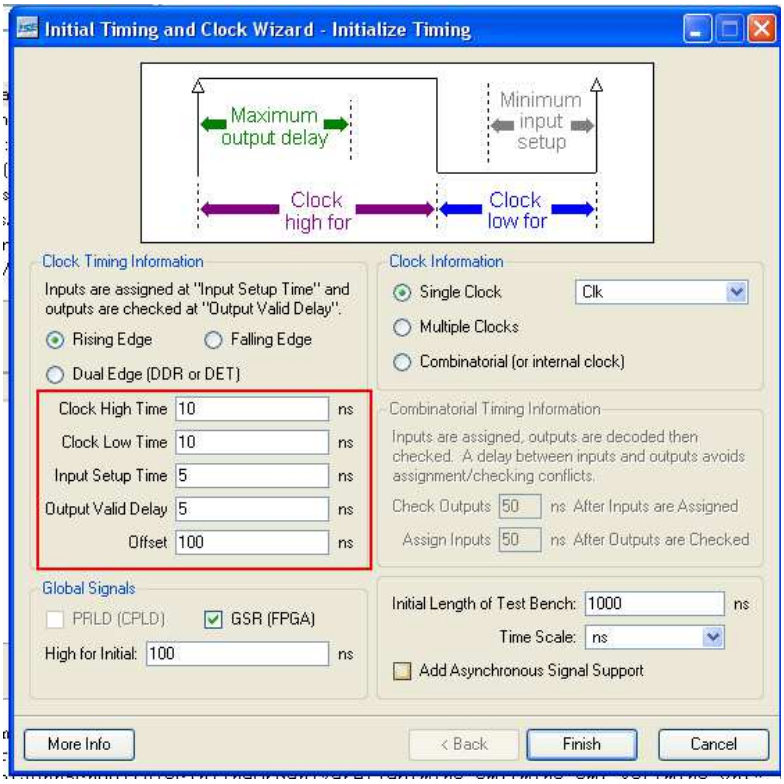


figura 5

În acest moment programul de test este în fişierul **ROM32x32_test1.vhd** iar acest fişier nu este în proiect**.** Pentru a-l adăuga în proiect mai întâi eliminaţi fişierul **ROM32x32.vhd** (cu Remove) iar apoi adăugaţi pe **ROM32x32_test1.vhd.** Eliminarea se face doar din proiect, fişierul eliminat nu este şters şi poate fi adăugat ulterior dacă este cazul.

Pentru simulare adăugaţi la proiect un fişierul de tip tbw cu numele tb_mips.tbw. Deoarece proiectul este de tip secvenţial sincron, selectaţi **Rising Edge** (de obicei este deja selectat) şi faceţi setările din alăturată.

Selectaţi simulare de tip Behavioral, pentru că durează mai puţin şi este uşor de urmărit simularea.

Setaţi valorile citite prin intermediul porturilor INW0 şi INW1 la:

```
INW0=0xAAAA_AAAB          INW1=0x5555_5555
```

Faceţi setarea acolo unde indică săgeata roşie în figura 6. Aceste valori rămân nemodificate pe durata întregii simulări.

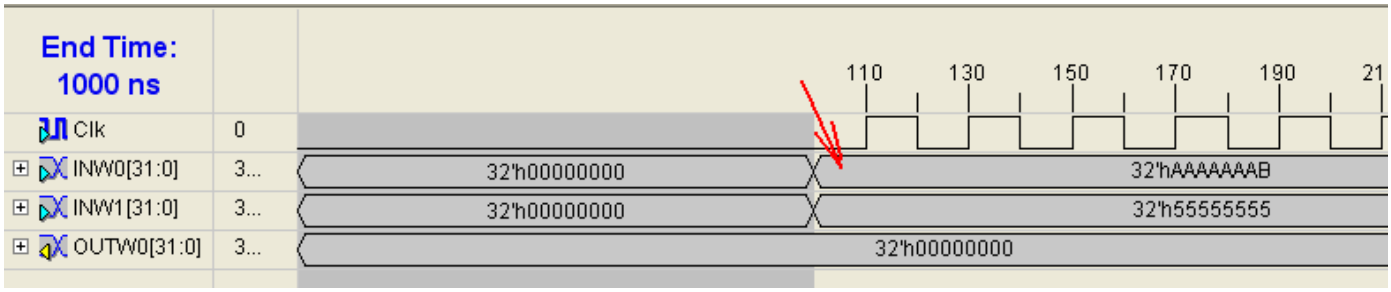


figura 6

Executaţi simularea. După ce setaţi ca toate semnalele să fie afişate în hexazecimal, veţi obţine o situaţie **asemănătoare** cu cea din figura următoare:

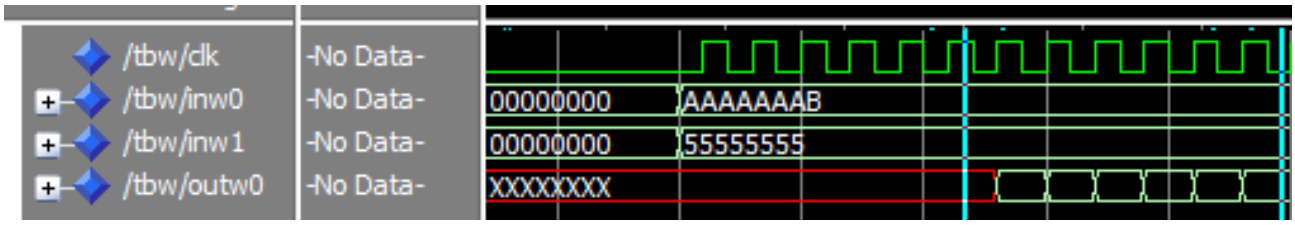


figura 7

În mod obişnuit simulatorul va afişa în fereastra Wave numai porturile (marcherii I/O) modulului din vârful ierarhiei. Aceste semnale sunt cele din figura 7.

Deşi nu apar semnalele interne, semnalul OUTW0 este suficient pentru a vedea că MIPS funcţionează corect. Dacă măriţi zona dintre cursoarele bleu din figura 7, ar trebui să obţineţi aceeaşi succesiune de valori ca cea din coloana OUTW0 din Tabelul 1.

Indiferent de rezultat, de îndată ce ați executat simularea, chemaţi profesorul pentru înregistrarea progresului.

Pasul 4: Pregătirea pentru depanare

În cazul foarte probabil în care vom constata în simulare că una sau mai multe instrucţiuni nu funcţionează corect trebuie să găsim cauza erorii şi apoi să o corectăm. În acest sens vom calcula manual valorile tuturor semnalelor din implementarea MIPS şi le vom compara cu valorile obţinute prin simulare. Locul în care apar diferenţe ne spune ce este greşit în implementare.

Valorile calculate manual ale semnalelor din implementarea MIPS în cazul executării programului de test se vor trece într-un tabel. Pe coloane se vor plasa instrucțiunile din programul de test iar pe linii semnalele din calea de date plus semnalele de control. Pentru a exemplifica acest proces, vom completa acest tabel în cazul primei instrucţiuni.

Prima instrucţiune din programul de test este `lw $2, 0x40($0)`. Execuția acestei instrucţiuni presupune anumite valori ale semnalelor de date şi de control. Pe măsură ce se fac calculele, valorile calculate se scriu tabel:

- Imediat după Reset **PC** este 0. Biţii 6:2 ai PC constituie adresa memoriei de program. Dacă PC este 0 şi biţii săi 6:0 sunt toţi zero. În memoria de program se află cuvântul instrucţiune **Instr**.

Valoarea lui **Instr** pentru `PC=0x0000_0000`, este codul maşină al instrucţiunii `lw $2, 0x40($0)`, adică `0x8c02_0040`. Valoarea **PC** şi a lui **Instr** se trec în tabel:

	`lw $2, 0x40($0)`	`lw $3, 0x44($0)`	`add $4, $2, $3`	`sub $5, $2, $3`
PC	`0x0000_0000`			
Instr	`0x8c02_0040`			

- Registrul sursă *rs* este registrul 0. Conținutul registrului 0 este întotdeauna 0. În implementarea MIPS registrul *rs* este **RdReg1** iar conținutul său **RdData1**. Valorile acestor semnale (0x00 şi 0x0000_0000) se trec în tabel:

	`lw $2, 0x40($0)`	`lw $3, 0x44($0)`	`add $4, $2, $3`	`sub $5, $2, $3`
PC	`0x0000_0000`			
Instr	`0x8c02_0040`			
RdReg1	`0x00`			
RdData1	`0x0000_0000`			

- În cazul instrucţiunii *lw* registrul target *rt* este registru destinaţie. În implementarea MIPS *rt* este **RdReg2** iar conținutul sau **RdData2**. Deoarece *rt* este registru destinaţie valoarea citită din acest registru, adică **RdData2**, nu contează. Din acest motiv valoarea lui **RdData2** în cazul instrucţiunii *lw* este xxxx_xxxx. rt= **RdReg2**=0x02 şi **RdData2**=xxxx_xxxx se trec în tabel:

	`lw $2, 0x40($0)`	`lw $3, 0x44($0)`	`add $4, $2, $3`	`sub $5, $2, $3`
PC	`0x0000_0000`			
Instr	`0x8c02_0040`			
RdReg1	`0x00`			
RdData1	`0x0000_0000`			
RdReg2	`0x02`			
RdData2	`xxxx_xxxx`			

- În continuare ALU va calcula adresa locaţiei de memorie care va citi. Această adresa se calculează prin adunarea conţinutului registrului *rs* cu ofsetul extins ca semn. În cazul instrucţiunii `lw $2, 0x40($0)` ofsetul este 0x0040. După extinderea semnului se obţine valoarea SE(0x0040)=0x0000_0040. Această valoare este al doilea operand al ALU. Valoarea ofsetului se trece în tabel:

	`lw $2, 0x40($0)`	`lw $3, 0x44($0)`	`add $4, $2, $3`	`sub $5, $2, $3`
PC	`0x0000_0000`			
Instr	`0x8c02_0040`			
RdReg1	`0x00`			
RdData1	`0x0000_0000`			
RdReg2	`0x02`			
RdData2	`xxxx_xxxx`			
offset	`0x0040`			

- Pentru ca al doilea operand ALU să fie SE(offset) trebuie ca semnalul de control **ALUsrc** să fie ‚1'. În cazul instrucțiunii *lw* adresa se calculează prin adunarea conţinutului registrului *rs* cu offsetul extins ca semn. Pentru ca ALU să efectueze adunare semnalul **ALUOP** trebuie să fie „00".

- ALU va aduna SE(0x0040)=0x0000_0040 cu conţinutul registrului zero (care este întotdeauna 0x0000_0000). Rezultatul este **0x0000_0040 + 0x0000_0000 = 0x0000_0040**. Aceasta este valoarea semnalului **ALU_out**. Valorile semnalelor **ALUsrc**, **ALUOP** şi **ALU_out** se trec în tabel:

	lw $2, 0x40($0)	lw $3, 0x44($0)	add $4, $2, $3	sub $5, $2, $3
PC	0x0000_0000			
Instr	0x8c02_0040			
RdReg1	0x00			
RdData1	0x0000_0000			
RdReg2	0x02			
RdData2	xxxx_xxxx			
offset	0x0040			
ALUsrc	‚1'			
ALUOP	„00"			
ALU_out	0x0000_0040			

- Ieşirea ALU (**ALU_out**) este adresa memoriei de date. În cazul instrucţiunii *lw* memoria trebuie să fie citită. În cazul citirii semnalul **MemWr** este inactiv, adică are valoarea ‚0'. Citirea se face de la adresa 0x0000_0040. La această adresă se află locaţia specială INW0. Aşa cum s-a specificat la pasul 1, testul se va executa cu **INW0=0xAAAA_AAAB** şi **INW1=0x5555_5555**. Din acest motiv la ieşirea memoriei trebuie să apară valoarea **0xAAAA_AAAB:**

	lw $2, 0x40($0)	lw $3, 0x44($0)	add $4, $2, $3	sub $5, $2, $3
PC	0x0000_0000			
Instr	0x8c02_0040			
RdReg1	0x00			
RdData1	0x0000_0000			
RdReg2	0x02			
RdData2	xxxx_xxxx			
offset	0x0040			
ALUsrc	‚1'			
ALUOP	„00"			
ALU_out	0x0000_0040			
MemOut	0xAAAA_AAAB			
MemWr	‚0'			
INW0	0xAAAA_AAAB			

- Ultima grupă de semnale controlează scrierea registrului destinaţie, adică *rt*=$2. Semnalul care controlează scrierea registrului este **WrReg**. Pentru ca `lw $2, 0x40($0)` scrie registrul 2, acest semnal trebuie să fie ‚1'.

  În general, data care se scrie provine fie de la ieşirea ALU, fie de la ieşirea memoriei de date. Semnalul care selectează care data care se va scrie este **Mem2Reg**. În cazul instrucţiunii `lw $2, 0x40($0)` datele provin de la memoria de date, astfel vă Mem2Reg este ‚1'. Data care se va scrie este **WrData**=0xAAAA_AAAB.

  Registrul în care se scrie data este fie *rt* fie *rd*. Semnalul care selectează ce registru se va scrie este **RegDest**. Valoarea lui RegDest este ‚0' şi selectează rt=**RegWr**=2. Toate valorile calculate anterior se trec în Tabelul 2:

Tabelul 2

	lw $2, 0x40($0)	lw $3, 0x44($0)	add $4, $2, $3	sub $5, $2, $3	Etc…
PC	0x0000_0000				
Instr	0x8c02_0040				
RdReg1	0x00				
RdData1	0x0000_0000				
RdReg2	0x02				
RdData2	xxxx_xxxx				
offset	0x0040				
ALUsrc	‚1'				
ALUOP	„00"				
ALU_out	0x0000_0040				
MemOut	0xAAAA_AAAB				
MemWr	‚0'				
RegWr	‚1'				
Mem2Reg	‚1'				
WrData	0xAAAA_AAAB				
RegDest	‚0'				
rd	xx				
WrReg	0x02				
INW0	0xAAAA_AAAB				
INW1	0x5555_5555				
OUTW0	xxxx_xxxx				

În acest moment valorile semnalelor în cazul instrucţiunii `lw $2, 0x40($0)` sunt determinate complet.

Pasul 5: Depanarea MIPS

Depanarea MIPS presupune compararea valorilor semnalelor din Tabelul 2 cu valorile obţinute prin simulare. Pentru a vizualiza formele de undă ale acestor semnale se procedează ca în figura 8:

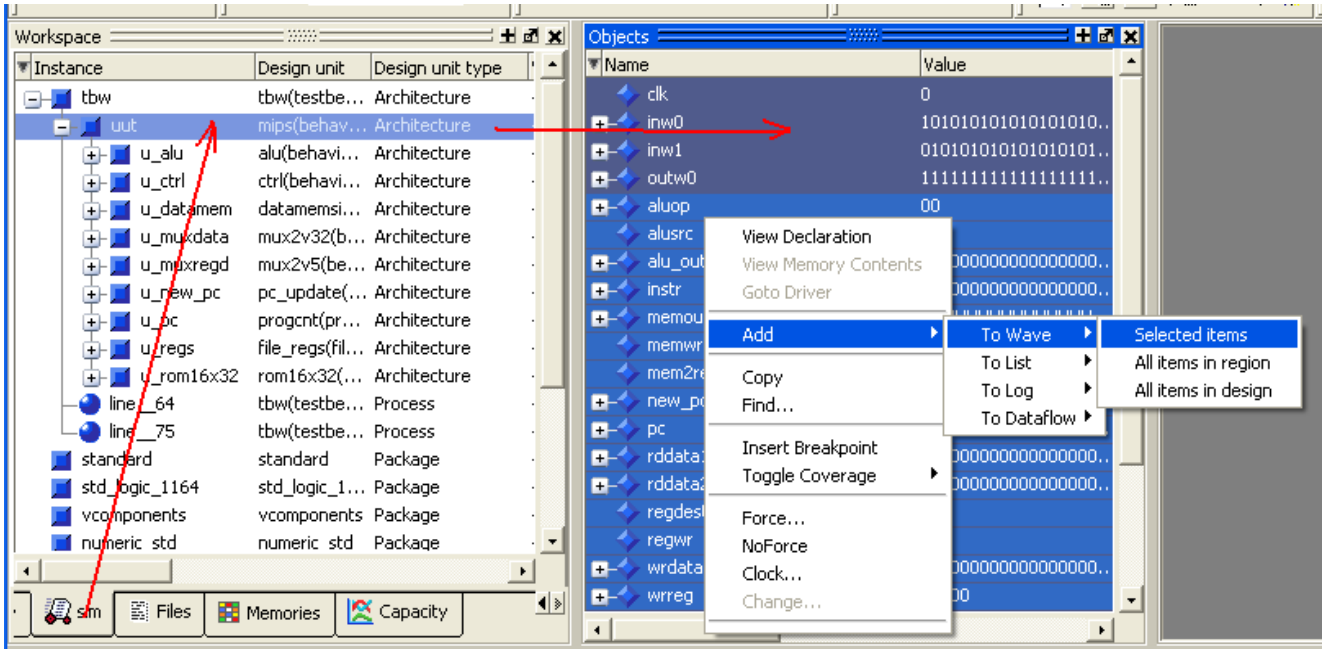


figura 8

1. Dacă nu este deja selectat, se selectează câmpul tab sim
2. În fereastra Workspace se selectează instanţa uut. În orice proiect Modelsim modulul 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. In fereastra Objects se selectează toate semnale care nu sunt deja 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 selectat.
7. Adăugaţi semnalelor selectate prin ce metodă va place mai mult.
8. Folosind Drag&drop rearanjaţi semnalele în ordinea din Tabelul 2. Nu toate semnalele din tabel sunt prezente în semnalele selectate.
9. Semnalele **RdReg1** şi **RdReg2** nu sunt prezente la nivelul *uut* sub acest nume, dar apar în interiorul modulului File_Regs. Pentru a adăuga la Vawe semnalele dintr-un modul diferit de *uut*, selectaţi *u_regs* în fereastra **Instance**. În fereastra **Objects** faceţi selecţiile din figura 9. Apoi adăugați aceste semnale la **Wave** aşa cum s-a explicat mai sus. Folosind Drag&drop rearanjaţi semnalele în ordinea din Tabelul 2. În plus față de semnalele menționate se adaugă și semnalul **s32reg32**, adică cele 32 de registre generale. Acestea sunt necesare pentru verificările rapide conform Tabelul 1.

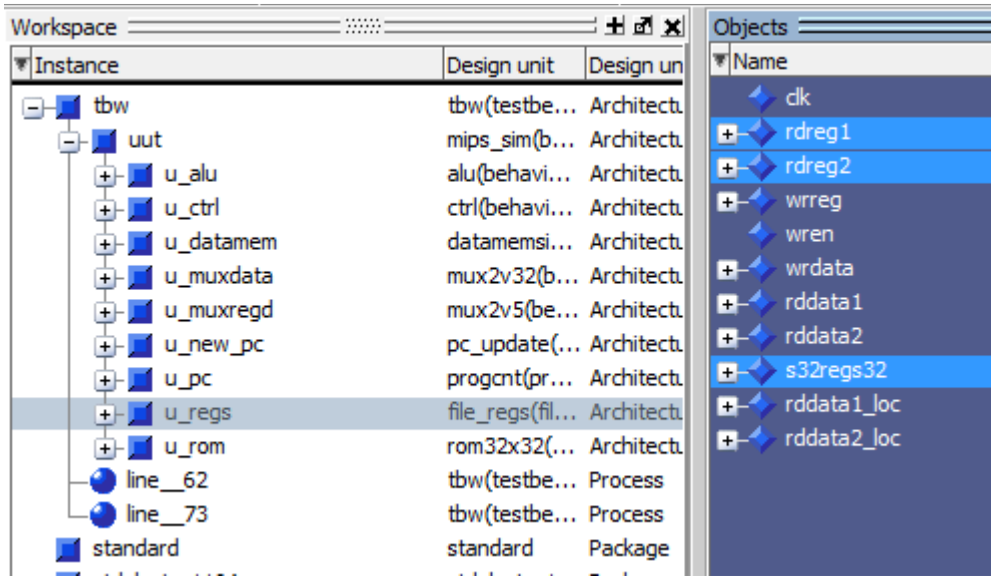


figura 9

10. Adăugaţi ofsetul. Acest semnal este semnalul **Faddr** din interiorul ALU.
11. Adăugaţi *rd*. Acesta este semnalul **I1** din interiorul mux-ului MUX2V5.
12. După aceste adăugări şi reordonări trebuie să obţineţi situaţia din figura 10.

    Observaţie: Spre deosebire de Tabelul 2 semnalele **Clk**, **INW0**, **INW1** şi **OUTW0** se poziţionează la începutul listei de semnale pentru a se putea urmării mai uşor comportamentul intrare-ieşire al procesorului.
13. Setaţi afişarea în hexazecimal pentru toate semnalele.

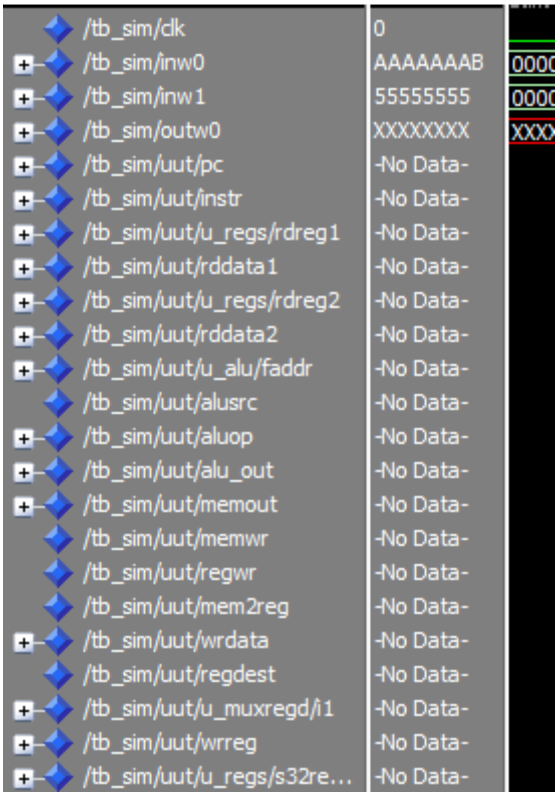


figura 10

14. Pentru a nu repeta procesul de adăugare şi rearanjare de semnale la fiecare nouă lansare a Modelsim, **toate** semnalele afişate în fereastra **Wave** se pot salva. Pentru aceasta din meniul **File** selectaţi opţiunea **Save Format…(**sau apăsaţi iconiţa cu discheta). La o nouă rulare a Modelsim, înainte de a încărca lista de semnale salvată anterior, ştergeţi toate semnalele din fereastra **Wave** iar apoi din meniul **File** selectaţi opţiunea **Load…**
15. Deoarece nu există date disponibile pentru semnale adăugate, apăsaţi butonul Restart. Rulați o nouă simulare **tastând** run 500 ns în fereastra principală Modelsim.

Se cere:

1. Faceţi simularea.
2. Prin intermediul semnalelor **PC**, **Instr**, **s32reg32** şi al portului **OUTW0** urmăriţi execuţia în timp a programului de test. Când o instrucţiune funcţionează corect, **s32reg32** şi **OUTW0** din simularea Modelsim vor avea valorile Tabelul 1.
3. Dacă toate instrucţiunile funcţionează corect, treceți la pasul 6. Dacă nu, completaţi Tabelul 2 pentru prima instrucţiune care nu funcţionează şi continuaţi cu pasul următor.
4. Pentru instrucțiunea care nu funcţionează corect, comparaţi coloana alocată acelei instrucţiuni în Tabelul 2 cu rezultatul simulării şi încercaţi să determinaţi cauza erorii. Pentru a reuşi să faceţi depanarea trebuie să ştiţi modul de funcţionare al MIPS din H&P. Dacă ştiţi teoria şi totuși nu reușiți să determinaţi cauza erorii, chemaţi profesorul.
5. Treceţi la pasul 1.
6. **Profesorul va verifica câte instrucţiuni funcţionează corect. Programul este alcătuit din 6 tipuri de instrucţiuni: LW, SW, ADD, SUB, AND şi OR.**

   Pentru fiecare instrucţiune care funcţionează corect se acordă 1,5 puncte.

Cine nu a implementat corect toate instrucţiunile va continua în laboratorul următor. Însă dacă la acest laborator ați obţinut sub 5, nota finală va fi 5.