CSE 331

Computer Organization

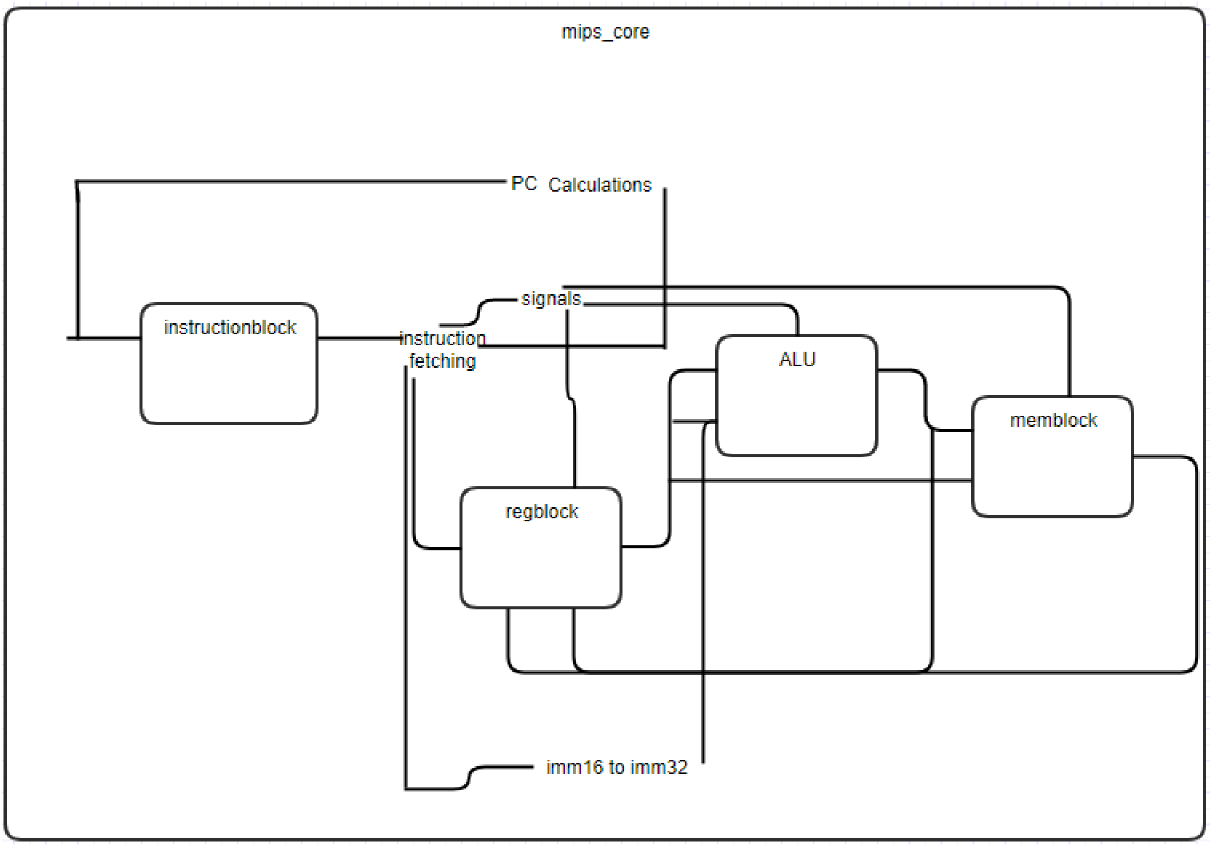
Project 3

Burak Furkan Akşahin

151044094

I. Introduction

1) Big Picture



All modules are in the mips_core modüle, so mips_core is top entity.

Muxes and wires' details are not written in the sketch. The details will be given in the modules part of this report. The wires are representing only inputs and outputs.

Instructionblock,regblock,memblock do file i/o.

2) Life cycle of 1 instruction

PC initialized to 0. Our instruction block catch this change and get the first instruction from instruction file.

This instruction is fetched and signals are set. Therefore regblock's inputs are set in fetch stage, we entered to regblock.

In regblock, read_data_1 and read_data_2 took from register file.

Read_data_1 and read_data_2 changed but ALU can take imm32 too, so we must choose from read_data_2 and imm32. We head to ALUblock.

In ALUblock, some calculations are done with given opcode and function codes and output will be input of memBlock.

memBlock take aluresult and content of rt, read or write by given opcode.

If read will be done, we must take this result as regblocks input again.

Therefore alublock's output and memblock's output can be input of regblock, there must be a choice there.

After writing reg, PC will change again. All this cycle will be done until PC is corrupted.

3) Missing parts, bonus parts, informations

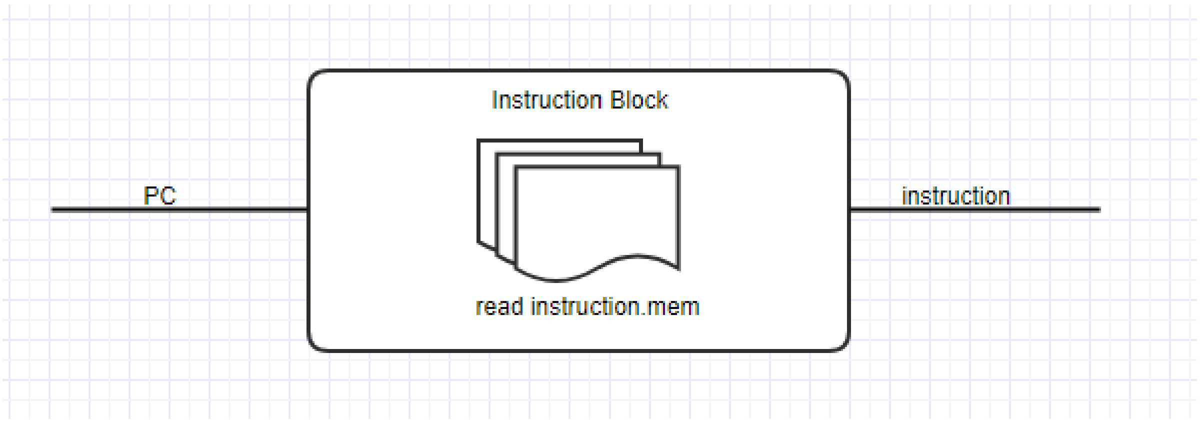
There is no missing part, all the core instructions are implemented.

Some warnings:

For regblock, write_data must be changed if you want to write reg more than 2 times in a row. Because verilog cant understand there is change in write_data, so it doesnt enter regblock for writing. But except that if the instructions are well tought, all programs can work as a charm.

II. Modules

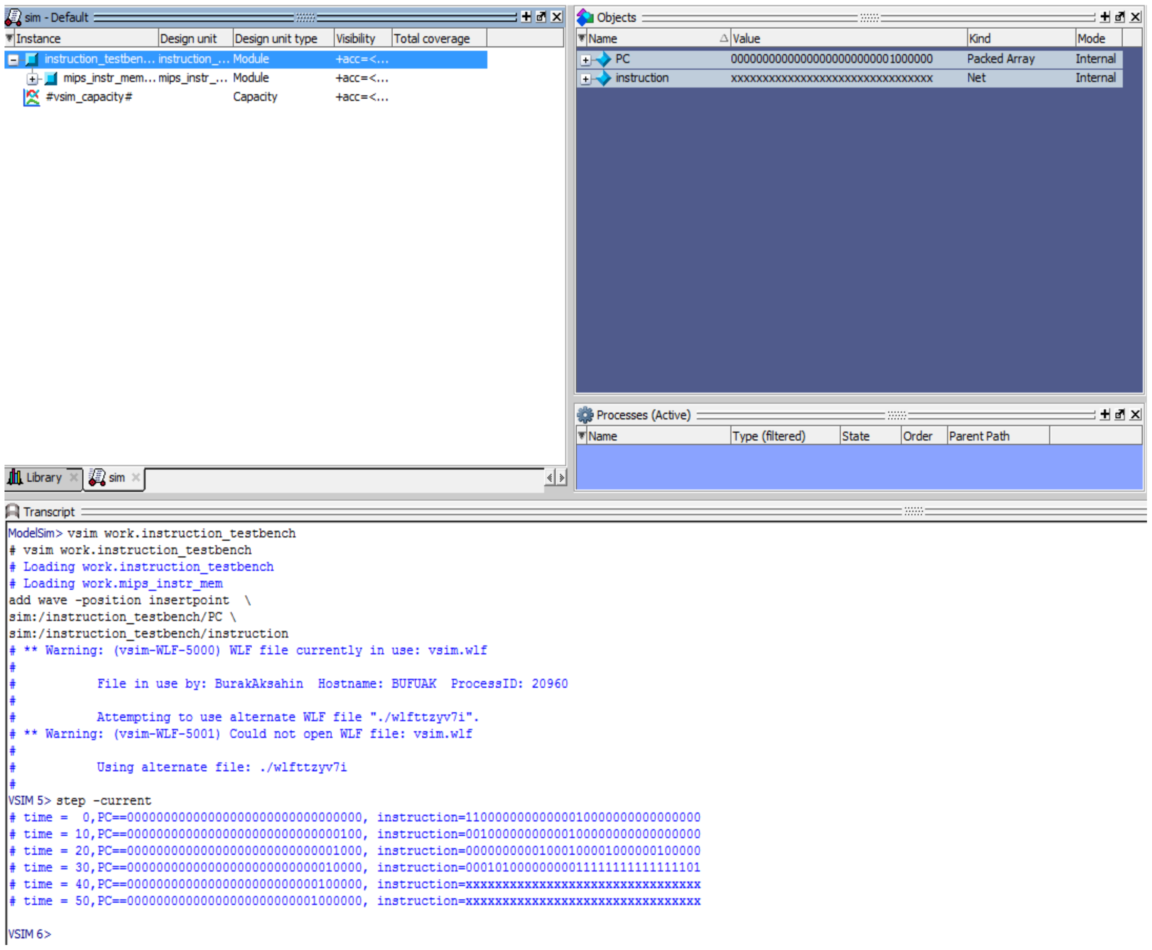
1) InstructionBlock



Inputs: PC

Outputs: instruction

Details: Reads from instruction.mem, and gives instruction as output from PC/4 index. Because PC is incremented by 4.



Instance	Design unit	Design unit type	Visibility	Total coverage
instruction_testben...	instruction_...	Module	+acc=<...	
mips_instr_mem...	mips_instr_...	Module	+acc=<...	
#vsim_capacity#		Capacity	+acc=<...	

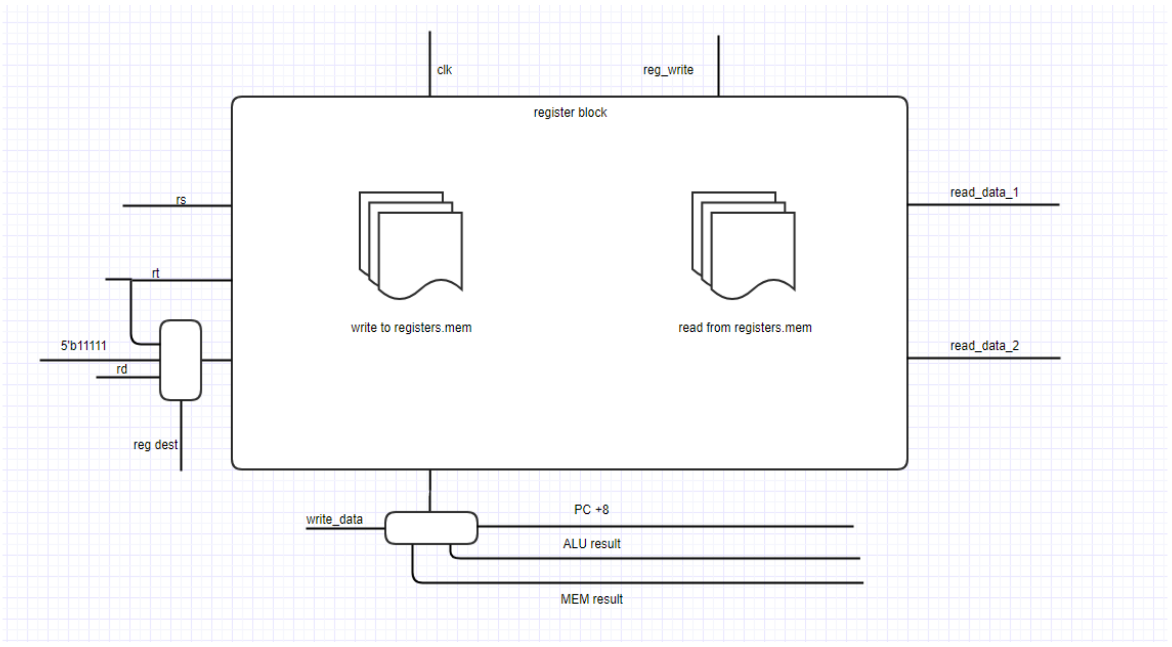
Name	Value	Kind	Mode
PC	00000000000000000000000001000000	Packed Array	Internal
instruction	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Net	Internal

Name	Type (filtered)	State	Order	Parent Path

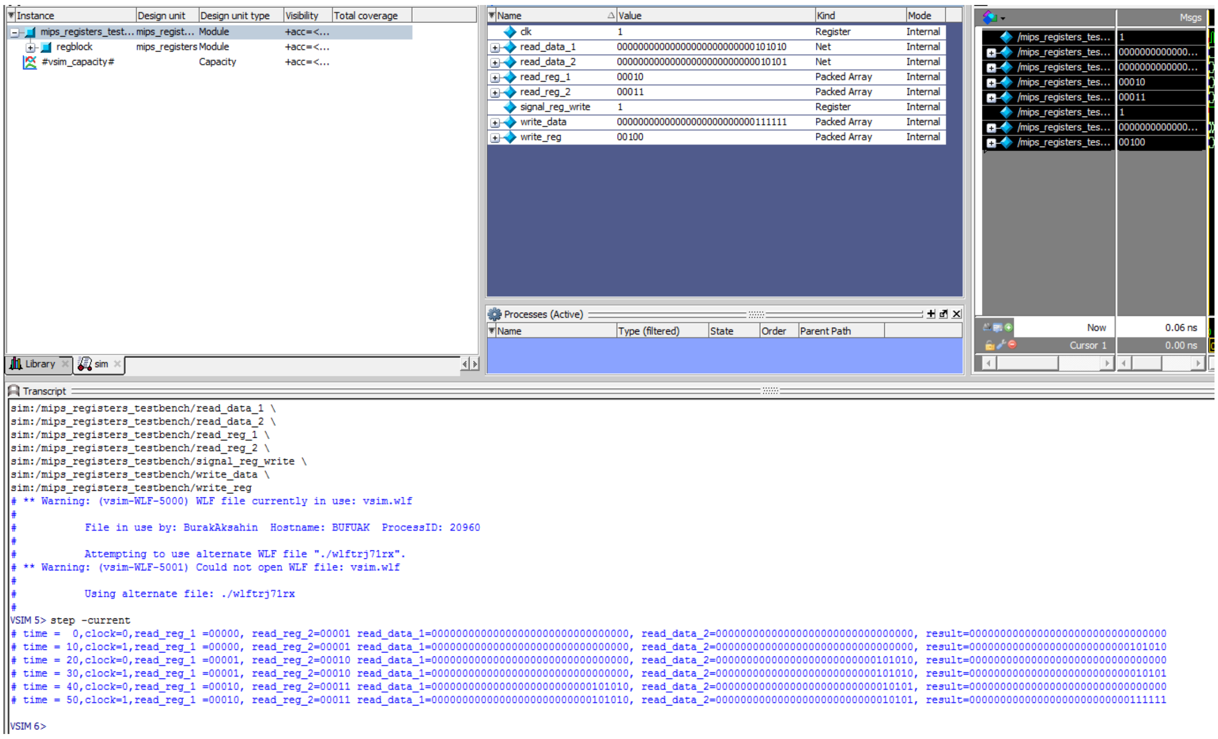
```
ModelSim> vsim work.instruction_testbench
# vsim work.instruction_testbench
# Loading work.instruction_testbench
# Loading work.mips_instr_mem
add wave -position insertpoint  \
sim:/instruction_testbench/PC \
sim:/instruction_testbench/instruction
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#
#           File in use by: BurakAksahin  Hostname: BUFUAK  ProcessID: 20960
#
#           Attempting to use alternate WLF file "./wlfttzyv7i".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#
#           Using alternate file: ./wlfttzyv7i
#
VSIM 5> step -current
# time =  0,PC==00000000000000000000000000000000, instruction=11000000000000010000000000000000
# time = 10,PC==00000000000000000000000000000100, instruction=00100000000000010000000000000000
# time = 20,PC==00000000000000000000000000001000, instruction=00000000010001000010000000100000
# time = 30,PC==00000000000000000000000000010000, instruction=00010100000000011111111111111101
# time = 40,PC==00000000000000000000000000100000, instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# time = 50,PC==00000000000000000000000001000000, instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

VSIM 6>
```

When PC is out of range, instruction will be 32’bx, in mips_core this will be controlled. If Pc will be x, program will be finished.

2) RegisterBlock



Inputs: rs,rt,(rt|5b’11111|rd),(PC+8|ALUresult|MEMresult),clk,reg_write
Outputs: read_data_1,read_data_2
Details: Reads from registers.mem and get contents of given inputs rs,rt to read_data_1 and read_data_2 in clk==0 and write coming content to destination register when clk==1 and reg_write signal is equal to 1.
PS: PC+8 and 5b’11111 are used for jal instruction



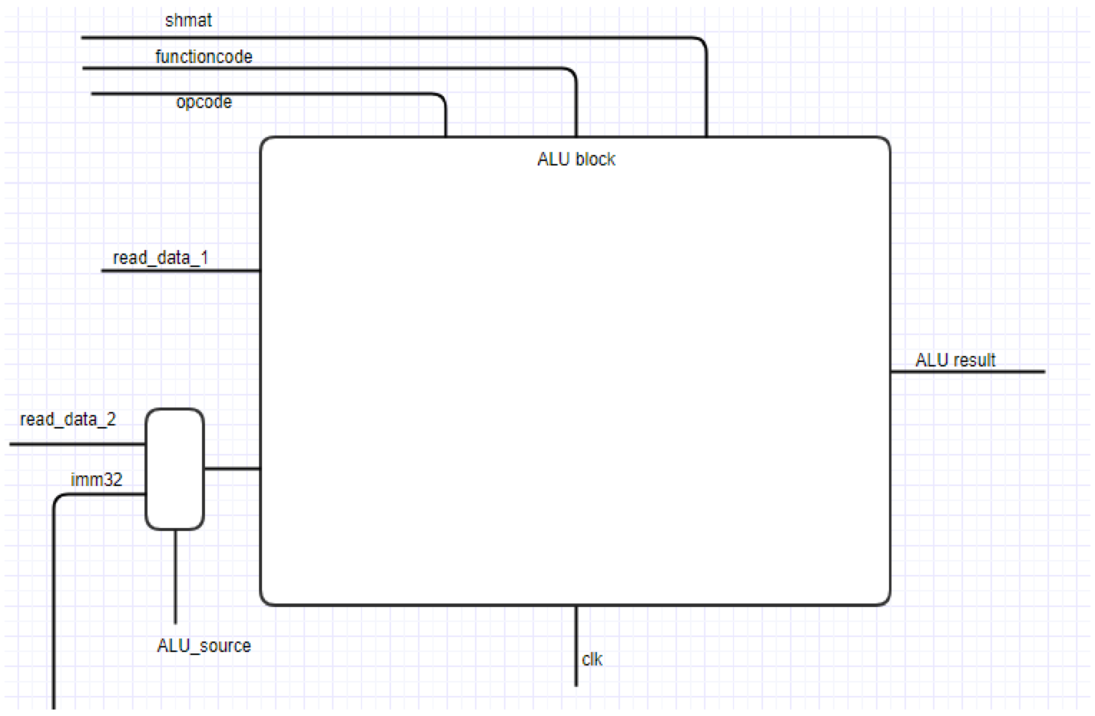
Instance	Design unit	Design unit type	Visibility	Total coverage
mips_registers_test...	mips_regist...	Module	+acc=<...	
regblock	mips_registers	Module	+acc=<...	
#vsim_capacity#		Capacity	+acc=<...	

Name	Value	Kind	Mode
clk	1	Register	Internal
read_data_1	00000000000000000000000000101010	Net	Internal
read_data_2	00000000000000000000000000010101	Net	Internal
read_reg_1	00010	Packed Array	Internal
read_reg_2	00011	Packed Array	Internal
signal_reg_write	1	Register	Internal
write_data	00000000000000000000000000111111	Packed Array	Internal
write_reg	00100	Packed Array	Internal

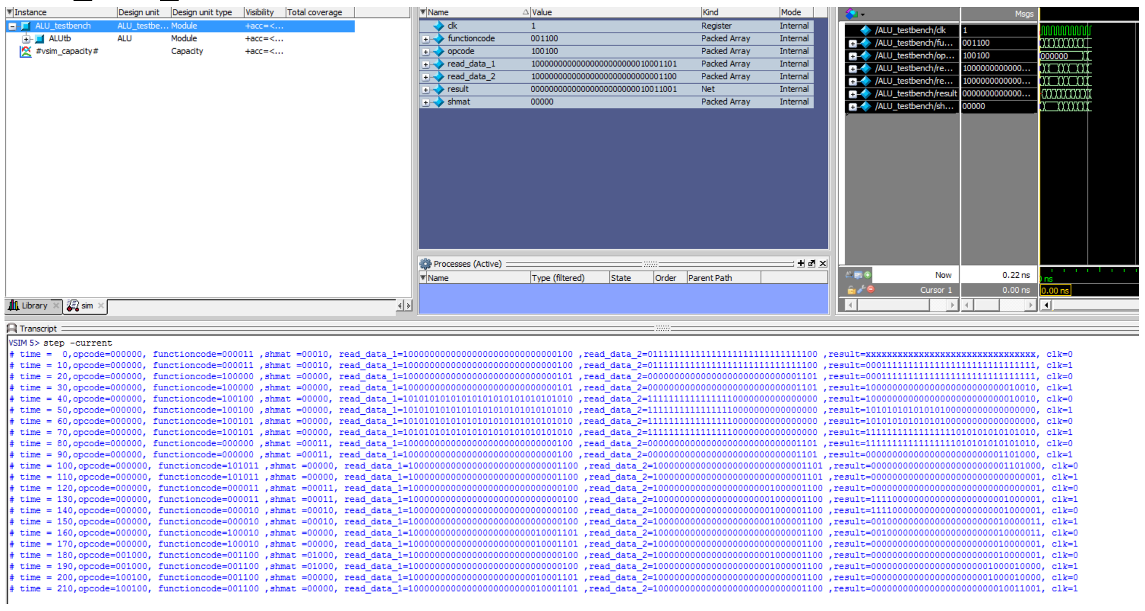
```
sim:/mips_registers_testbench/read_data_1 \
sim:/mips_registers_testbench/read_data_2 \
sim:/mips_registers_testbench/read_reg_1 \
sim:/mips_registers_testbench/read_reg_2 \
sim:/mips_registers_testbench/signal_reg_write \
sim:/mips_registers_testbench/write_data \
sim:/mips_registers_testbench/write_reg
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#
#           File in use by: BurakAksahin  Hostname: BUFUAK  ProcessID: 20960
#
#           Attempting to use alternate WLF file "./wlftrj71rx".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#
#           Using alternate file: ./wlftrj71rx
#
VSIM 5> step -current
# time =  0,clock=0,read_reg_1 =00000, read_reg_2=00001 read_data_1=00000000000000000000000000000000, read_data_2=00000000000000000000000000000000, result=00000000000000000000000000000000
# time = 10,clock=1,read_reg_1 =00000, read_reg_2=00001 read_data_1=00000000000000000000000000000000, read_data_2=00000000000000000000000000000000, result=00000000000000000000000000101010
# time = 20,clock=0,read_reg_1 =00001, read_reg_2=00010 read_data_1=00000000000000000000000000000000, read_data_2=00000000000000000000000000101010, result=00000000000000000000000000000000
# time = 30,clock=1,read_reg_1 =00001, read_reg_2=00010 read_data_1=00000000000000000000000000000000, read_data_2=00000000000000000000000000101010, result=00000000000000000000000000010101
# time = 40,clock=0,read_reg_1 =00010, read_reg_2=00011 read_data_1=00000000000000000000000000101010, read_data_2=00000000000000000000000000010101, result=00000000000000000000000000000000
# time = 50,clock=1,read_reg_1 =00010, read_reg_2=00011 read_data_1=00000000000000000000000000101010, read_data_2=00000000000000000000000000010101, result=00000000000000000000000000111111

VSIM 6>
```

I use two extra registers x and y, Because there was an error to assign values of registers to output registers. And this solved problem.

3) ALUBlock



Inputs:
functioncode,opcode,read_data_1,(read_data_2|imm32),clk,shmat
Outputs: ALU result
Details: Does operation read_data_1, ALU_mux and shmat by opcode and functioncode given. If instruction is signed, some temp signed registers are used for calculation, then this signed result is assigned to ALU result. Works only postedge clk.
If opcode says this instruction is memory. ALU does addition between read_data_1 and imm32.



Instance	Design unit	Design unit type	Visibility	Total coverage
ALU_testbench	ALU_testbe...	Module	+acc=<...	
ALUtb	ALU	Module	+acc=<...	
#vsim_capacity#		Capacity	+acc=<...	

Name	Value	Kind	Mode
clk	1	Register	Internal
functioncode	001100	Packed Array	Internal
opcode	100100	Packed Array	Internal
read_data_1	10000000000000000000000010001101	Packed Array	Internal
read_data_2	10000000000000000000000000001100	Packed Array	Internal
result	00000000000000000000000010011001	Net	Internal
shmat	00000	Packed Array	Internal

	Msgs
/ALU_testbench/clk	1
/ALU_testbench/fu...	001100
/ALU_testbench/op...	100100
/ALU_testbench/re...	1000000000000...
/ALU_testbench/re...	1000000000000...
/ALU_testbench/result	0000000000000...
/ALU_testbench/sh...	00000

Now 0.22 ns
Cursor 1 0.00 ns

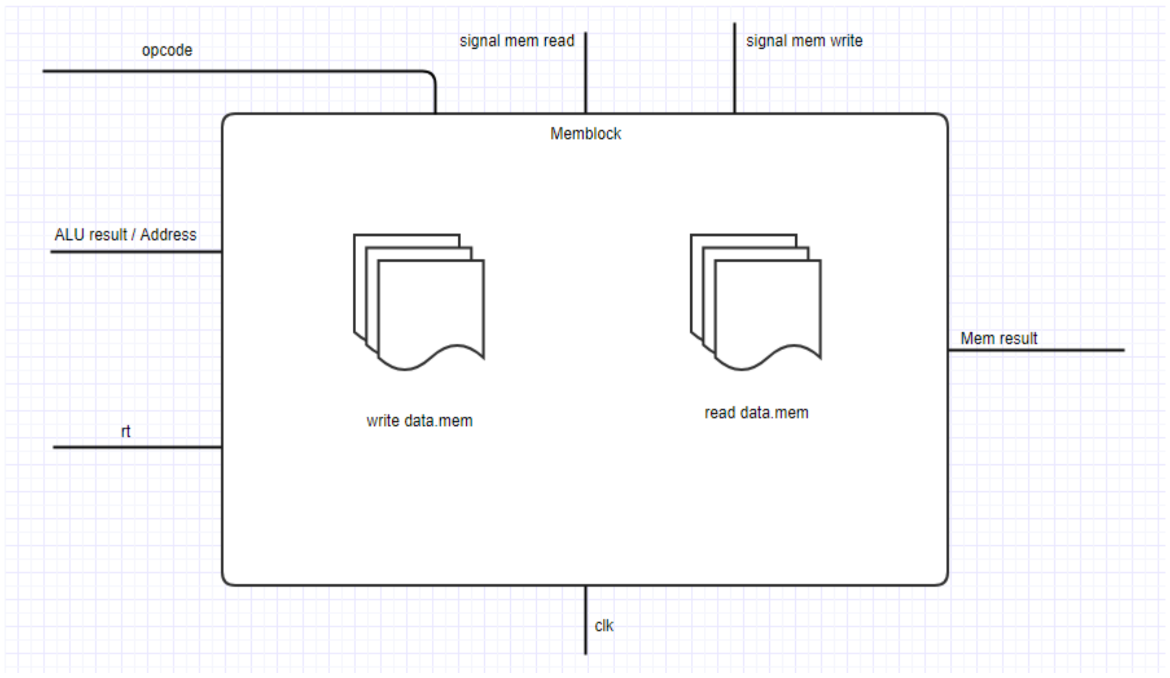
Processes (Active)

Name	Type (filtered)	State	Order	Parent Path

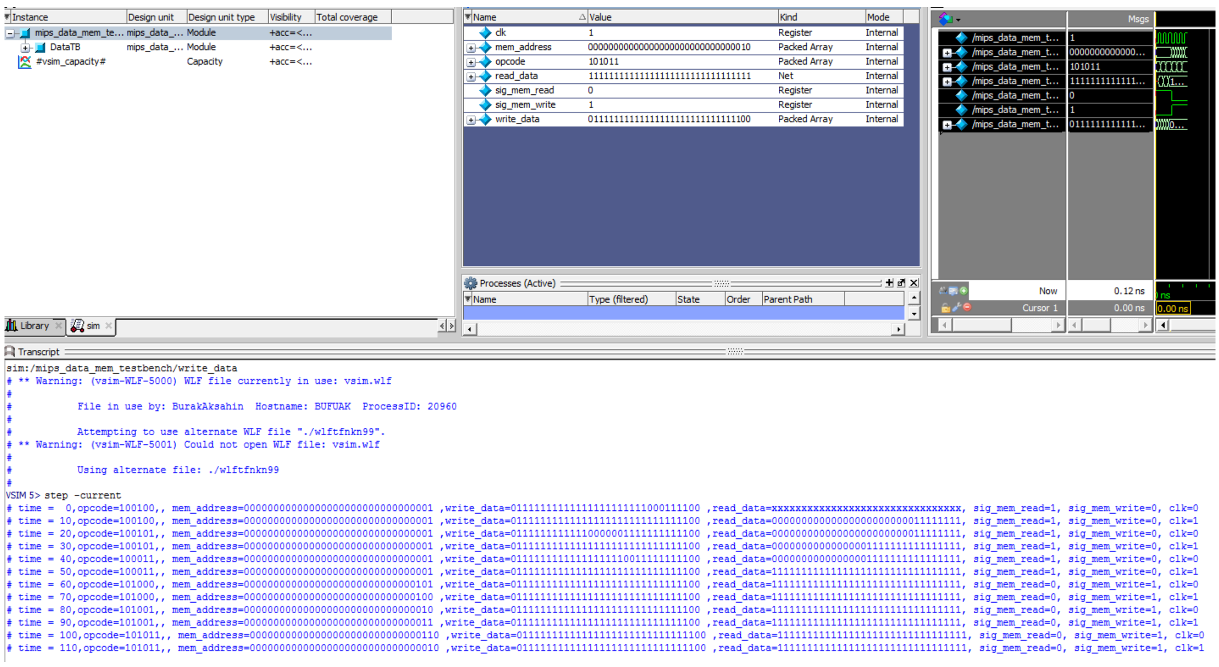
Transcript

```
VSIM 5> step -current
# time =  0,opcode=000000, functioncode=000011 ,shmat =00010, read_data_1=10000000000000000000000000000100 ,read_data_2=01111111111111111111111111111100 ,result=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, clk=0
# time = 10,opcode=000000, functioncode=000011 ,shmat =00010, read_data_1=10000000000000000000000000000100 ,read_data_2=01111111111111111111111111111100 ,result=00011111111111111111111111111111, clk=1
# time = 20,opcode=000000, functioncode=100000 ,shmat =00000, read_data_1=10000000000000000000000000000101 ,read_data_2=00000000000000000000000000001101 ,result=00011111111111111111111111111111, clk=0
# time = 30,opcode=000000, functioncode=100000 ,shmat =00000, read_data_1=10000000000000000000000000000101 ,read_data_2=00000000000000000000000000001101 ,result=10000000000000000000000000010010, clk=1
# time = 40,opcode=000000, functioncode=100100 ,shmat =00000, read_data_1=10101010101010101010101010101010 ,read_data_2=11111111111111110000000000000000 ,result=10000000000000000000000000010010, clk=0
# time = 50,opcode=000000, functioncode=100100 ,shmat =00000, read_data_1=10101010101010101010101010101010 ,read_data_2=11111111111111110000000000000000 ,result=10101010101010100000000000000000, clk=1
# time = 60,opcode=000000, functioncode=100101 ,shmat =00000, read_data_1=10101010101010101010101010101010 ,read_data_2=11111111111111110000000000000000 ,result=10101010101010100000000000000000, clk=0
# time = 70,opcode=000000, functioncode=100101 ,shmat =00000, read_data_1=10101010101010101010101010101010 ,read_data_2=11111111111111110000000000000000 ,result=11111111111111111010101010101010, clk=1
# time = 80,opcode=000000, functioncode=000000 ,shmat =00011, read_data_1=10000000000000000000000000000100 ,read_data_2=00000000000000000000000000001101 ,result=11111111111111111010101010101010, clk=0
# time = 90,opcode=000000, functioncode=000000 ,shmat =00011, read_data_1=10000000000000000000000000000100 ,read_data_2=00000000000000000000000000001101 ,result=00000000000000000000000001101000, clk=1
# time = 100,opcode=000000, functioncode=101011 ,shmat =00000, read_data_1=10000000000000000000000000001100 ,read_data_2=10000000000000000000000000001101 ,result=00000000000000000000000001101000, clk=0
# time = 110,opcode=000000, functioncode=101011 ,shmat =00000, read_data_1=10000000000000000000000000001100 ,read_data_2=10000000000000000000000000001101 ,result=00000000000000000000000000000001, clk=1
# time = 120,opcode=000000, functioncode=000011 ,shmat =00011, read_data_1=10000000000000000000000000000100 ,read_data_2=10000000000000000000001000001100 ,result=00000000000000000000000000000001, clk=0
# time = 130,opcode=000000, functioncode=000011 ,shmat =00011, read_data_1=10000000000000000000000000000100 ,read_data_2=10000000000000000000001000001100 ,result=11110000000000000000000001000001, clk=1
# time = 140,opcode=000000, functioncode=000010 ,shmat =00010, read_data_1=10000000000000000000000000000100 ,read_data_2=10000000000000000000001000001100 ,result=11110000000000000000000001000001, clk=0
# time = 150,opcode=000000, functioncode=000010 ,shmat =00010, read_data_1=10000000000000000000000000000100 ,read_data_2=10000000000000000000001000001100 ,result=00100000000000000000000010000011, clk=1
# time = 160,opcode=000000, functioncode=100010 ,shmat =00000, read_data_1=10000000000000000000000010001101 ,read_data_2=10000000000000000000000000001100 ,result=00100000000000000000000010000011, clk=0
# time = 170,opcode=000000, functioncode=100010 ,shmat =00000, read_data_1=10000000000000000000000010001101 ,read_data_2=10000000000000000000000000001100 ,result=00000000000000000000000010000001, clk=1
# time = 180,opcode=001000, functioncode=001100 ,shmat =01000, read_data_1=10000000000000000000000000000100 ,read_data_2=10000000000000000000001000001100 ,result=00000000000000000000000010000001, clk=0
# time = 190,opcode=001000, functioncode=001100 ,shmat =01000, read_data_1=10000000000000000000000000000100 ,read_data_2=10000000000000000000001000001100 ,result=00000000000000000000000100010000, clk=1
# time = 200,opcode=100100, functioncode=001100 ,shmat =00000, read_data_1=10000000000000000000000010001101 ,read_data_2=10000000000000000000000000001100 ,result=00000000000000000000000100010000, clk=0
# time = 210,opcode=100100, functioncode=001100 ,shmat =00000, read_data_1=10000000000000000000000010001101 ,read_data_2=10000000000000000000000000001100 ,result=00000000000000000000000010011001, clk=1
```

4) MemBlock



Inputs:
Address,write_data,opcode,sig_mem_read,sig_mem_write,clk
Outputs: Mem result
Details: Doesnt work if read and write signals are 0 or clk==0. Reads when signal_mem_read=1 and clk=1. Load byte, load half Word, load Word all these functionalities are working. To work properly, i “and”ed memory content with 32bit number to get byte,half Word and Word content.
When clk is 1 and sig_mem_write=1, this modüle writes given rt content to the memory. It can write byte, half Word, Word again. Alp hoca said sc is not important to implement. So i ignored atomic part of it, it is working like sw.

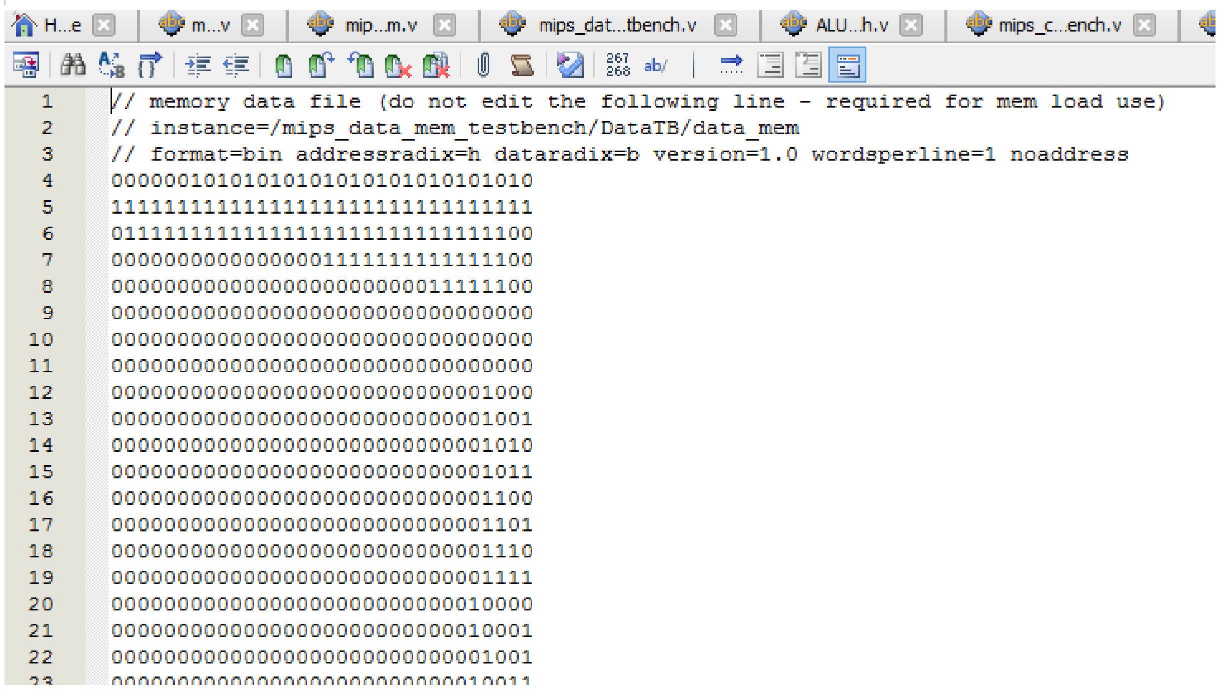


Instance	Design unit	Design unit type	Visibility	Total coverage
mips_data_mem_te...	mips_data_...	Module	+acc=<...	
DataTB	mips_data_...	Module	+acc=<...	
#vsim_capacity#		Capacity	+acc=<...	

Name	Value	Kind	Mode
clk	1	Register	Internal
mem_address	00000000000000000000000000000010	Packed Array	Internal
opcode	101011	Packed Array	Internal
read_data	11111111111111111111111111111111	Net	Internal
sig_mem_read	0	Register	Internal
sig_mem_write	1	Register	Internal
write_data	01111111111111111111111111111100	Packed Array	Internal

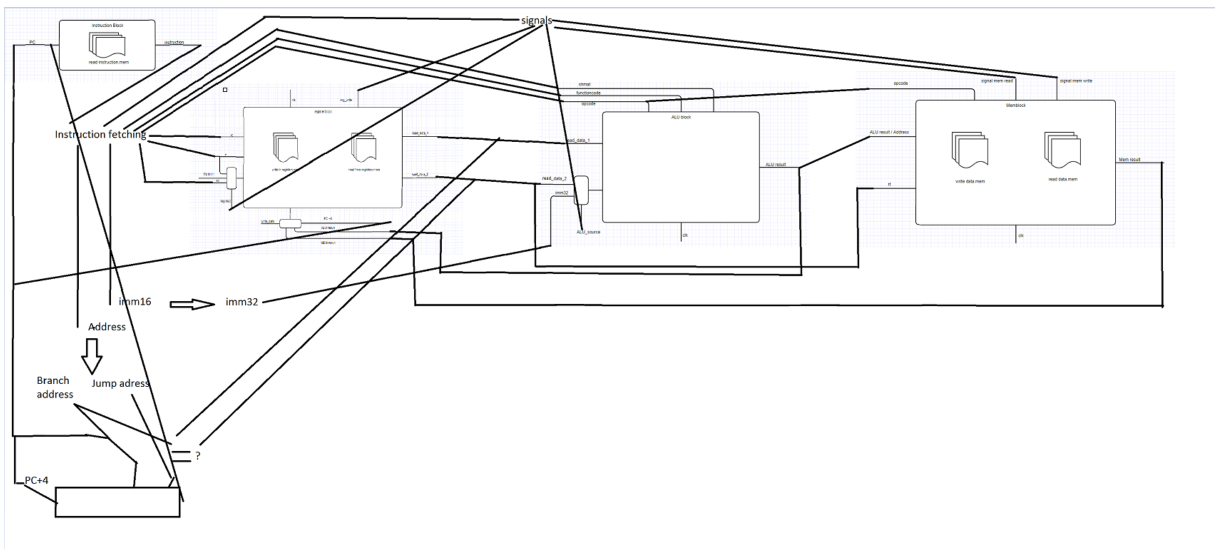
	Msgs
/mips_data_mem_t...	1
/mips_data_mem_t...	0000000000000...
/mips_data_mem_t...	101011
/mips_data_mem_t...	1111111111111...
/mips_data_mem_t...	0
/mips_data_mem_t...	1
/mips_data_mem_t...	0111111111111...
Now	0.12 ns
Cursor 1	0.00 ns

```
sim:/mips_data_mem_testbench/write_data
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#
#           File in use by: BurakAksahin  Hostname: BUFUAK  ProcessID: 20960
#
#           Attempting to use alternate WLF file "./wlftfnkn99".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#
#           Using alternate file: ./wlftfnkn99
#
VSIM 5> step -current
# time =  0,opcode=100100,, mem_address=00000000000000000000000000000001 ,write_data=01111111111111111111111000111100 ,read_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, sig_mem_read=1, sig_mem_write=0, clk=0
# time = 10,opcode=100100,, mem_address=00000000000000000000000000000001 ,write_data=01111111111111111111111111111100 ,read_data=00000000000000000000000011111111, sig_mem_read=1, sig_mem_write=0, clk=1
# time = 20,opcode=100101,, mem_address=00000000000000000000000000000001 ,write_data=01111111111110000001111111111100 ,read_data=00000000000000000000000011111111, sig_mem_read=1, sig_mem_write=0, clk=0
# time = 30,opcode=100101,, mem_address=00000000000000000000000000000001 ,write_data=01111111111111111111111111111100 ,read_data=00000000000000001111111111111111, sig_mem_read=1, sig_mem_write=0, clk=1
# time = 40,opcode=100011,, mem_address=00000000000000000000000000000001 ,write_data=01111111111111111110011111111100 ,read_data=00000000000000001111111111111111, sig_mem_read=1, sig_mem_write=0, clk=0
# time = 50,opcode=100011,, mem_address=00000000000000000000000000000001 ,write_data=01111111111111111111111111111100 ,read_data=11111111111111111111111111111111, sig_mem_read=1, sig_mem_write=0, clk=1
# time = 60,opcode=101000,, mem_address=00000000000000000000000000000101 ,write_data=01111111111111111111111111111100 ,read_data=11111111111111111111111111111111, sig_mem_read=0, sig_mem_write=1, clk=0
# time = 70,opcode=101000,, mem_address=00000000000000000000000000000100 ,write_data=01111111111111111111111111111100 ,read_data=11111111111111111111111111111111, sig_mem_read=0, sig_mem_write=1, clk=1
# time = 80,opcode=101001,, mem_address=00000000000000000000000000000010 ,write_data=01111111111111111111111111111100 ,read_data=11111111111111111111111111111111, sig_mem_read=0, sig_mem_write=1, clk=0
# time = 90,opcode=101001,, mem_address=00000000000000000000000000000011 ,write_data=01111111111111111111111111111100 ,read_data=11111111111111111111111111111111, sig_mem_read=0, sig_mem_write=1, clk=1
# time = 100,opcode=101011,, mem_address=00000000000000000000000000000110 ,write_data=01111111111111111111111111111100 ,read_data=11111111111111111111111111111111, sig_mem_read=0, sig_mem_write=1, clk=0
# time = 110,opcode=101011,, mem_address=00000000000000000000000000000010 ,write_data=01111111111111111111111111111100 ,read_data=11111111111111111111111111111111, sig_mem_read=0, sig_mem_write=1, clk=1
```



```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_data_mem_testbench/DataTB/data_mem
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000010101010101010101010101010
11111111111111111111111111111111
01111111111111111111111111111100
00000000000000011111111111111100
00000000000000000000000011111100
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000001001
00000000000000000000000000010011
```

5) Mips_core



Inputs:

None

Outputs: Result

Details: When pc is changed, instruction block catches that and gives new instruction to mips_core. Instruction is fetched like that.

```
if(instruction==32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)
   $finish;
opcode = instruction[31:26];
rs = instruction[25:21];
rt = instruction[20:16];
rd = instruction[15:11];
shmat = instruction[10:6];
functioncode = instruction[5:0];
imm = instruction[15:0];
adress = instruction[25:0];
```

After that we arrange signals from our opcode and functioncode. Reg_dst_mux is choosen too.(For R type rd,Others rt).

There is a #2 delay to syncnorize all inputs for register block.

After this choice,We entered register block. Register block gives read_data_1 and read_data_2 contents.

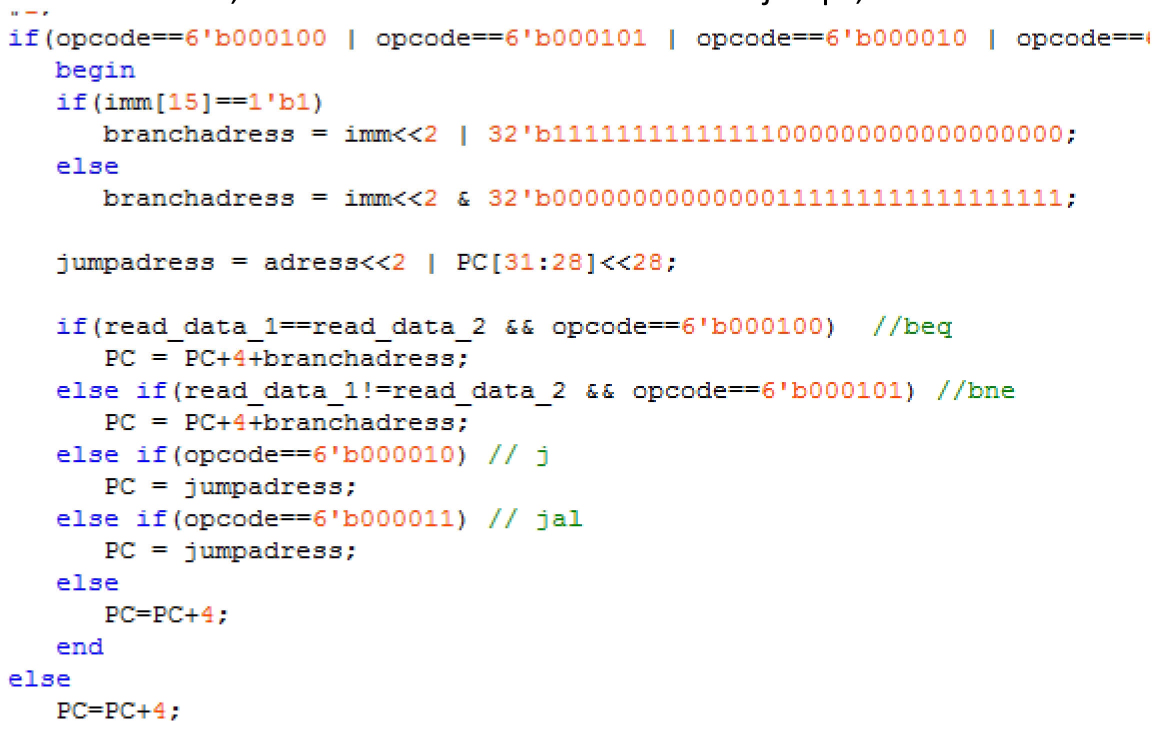
Now we have to choose second input for ALU. In Rtypes this second input is read_data_2 but for I and J types input is imm32. After ALU inputs are given, we have to make clk 1 to start ALU's code.(ALU works only in postedge clk we have shown in ALU testbench).

ALU block gives a result, this result will be write_data of register block in Rtypes, adress of mem block in I types. Then we have a #7 delay to syncronize this inputs again. In this #7 delay, we entered memory block.

Memory block has 2 data entry, one of them is ALU result, other is rt content. Rt is ready after register block segment so i have to say that memory block must work in only clk=1. Because in clk=1, alu result is ready too. If sig_mem_read there is a mem result which will be written to register.

So our reg block can have 3 write_data possiblities, MEM_result(I type),ALU_result(R type),PC+8(jal). To choose that i give a #1 delay to sncyronize this input to register block.

After all of that, we have to look for branches and jumps,



```
if(opcode==6'b000100 | opcode==6'b000101 | opcode==6'b000010 | opcode==
   begin
   if(imm[15]==1'b1)
      branchadress = imm<<2 | 32'b11111111111111100000000000000000;
   else
      branchadress = imm<<2 & 32'b00000000000000011111111111111111;

   jumpadress = adress<<2 | PC[31:28]<<28;

   if(read_data_1==read_data_2 && opcode==6'b000100)  //beq
      PC = PC+4+branchadress;
   else if(read_data_1!=read_data_2 && opcode==6'b000101) //bne
      PC = PC+4+branchadress;
   else if(opcode==6'b000010) // j
      PC = jumpadress;
   else if(opcode==6'b000011) // jal
      PC = jumpadress;
   else
      PC=PC+4;
   end
else
   PC=PC+4;
```

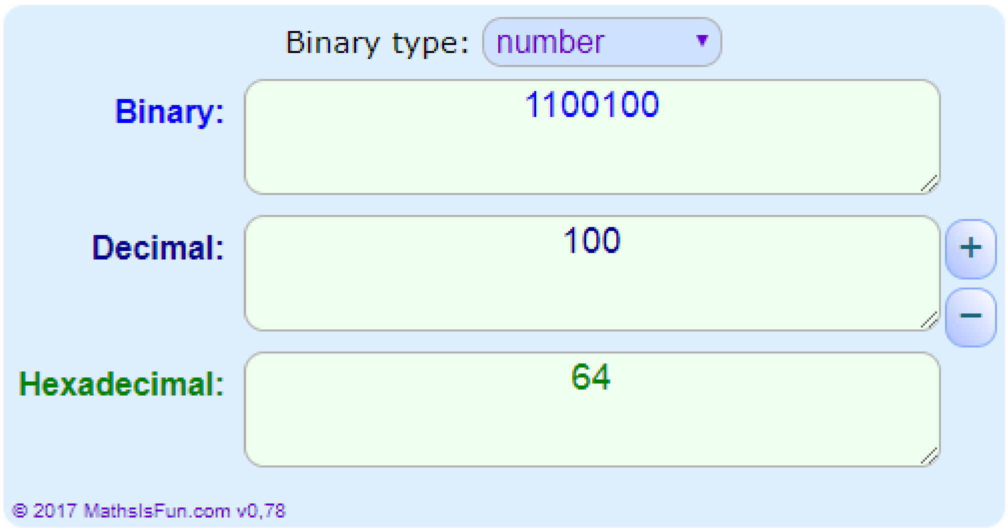
All program counter calculations are the same with mips instruction set paper. PC is incremented by 4 if there is no jump or branch, branch and jump addresses calculated correctly and same with mips processor.

After PC is calculated, clk will be 0. And mips_core waits for new instruction.

There is an example program execute in my mips_core. Which gives addition of 0 to given number in data.mem[0] and writes result to data.mem[1].

$$\text{Sum from 1 to n} = \frac{n(n+1)}{2}$$

$$\text{Sum from 1 to 100} = \frac{100(100+1)}{2} = (50)(101) = 5050$$


Binary type: number

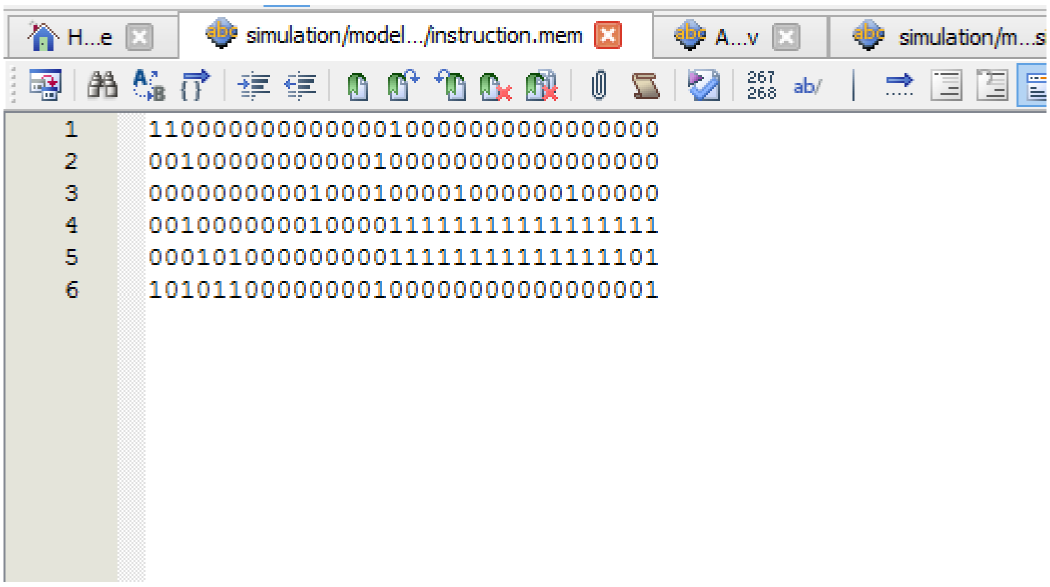
Binary: 1100100

Decimal: 100

Hexadecimal: 64

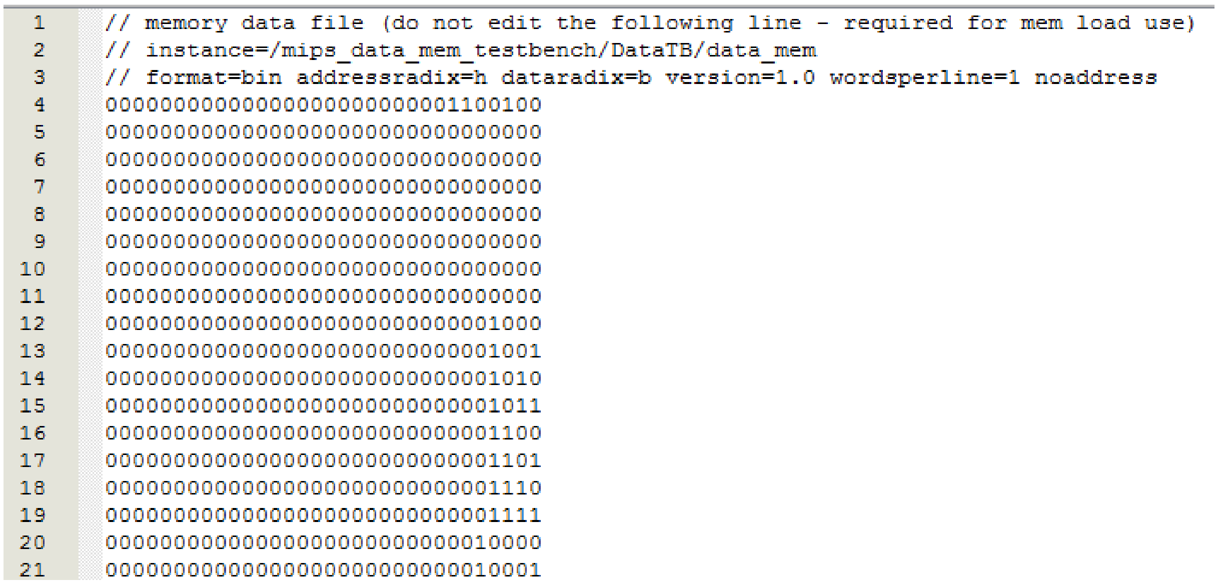
© 2017 MathsIsFun.com v0,78

Instruction.mem



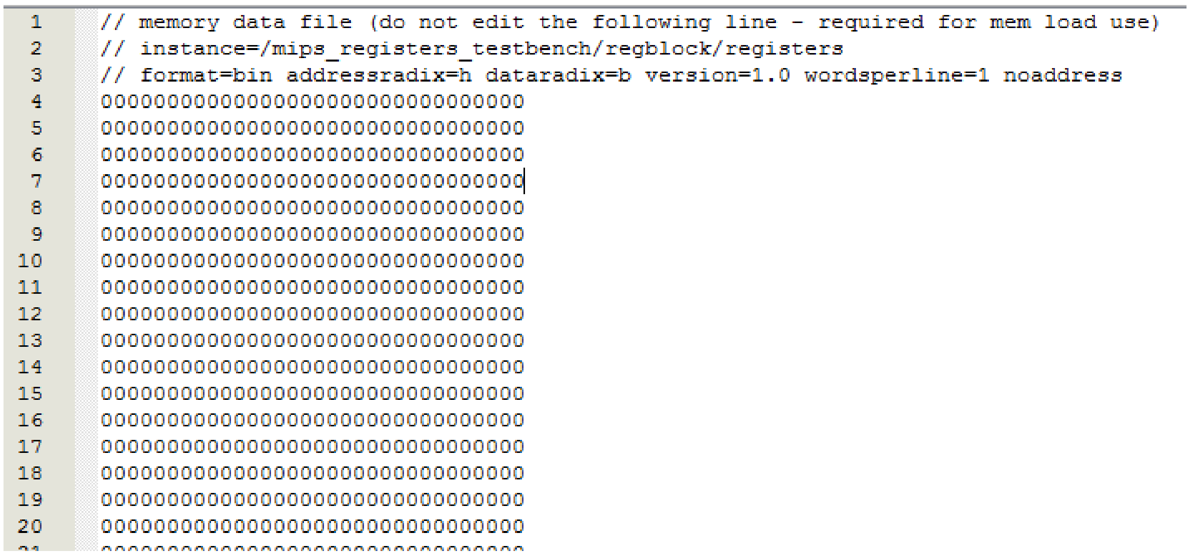
```
11000000000000010000000000000000
00100000000000100000000000000000
00000000001000100001000000100000
00100000001000011111111111111111
00010100000000011111111111111101
10101100000000100000000000000001
```

Data.mem before executing, data.mem[0] is our n.



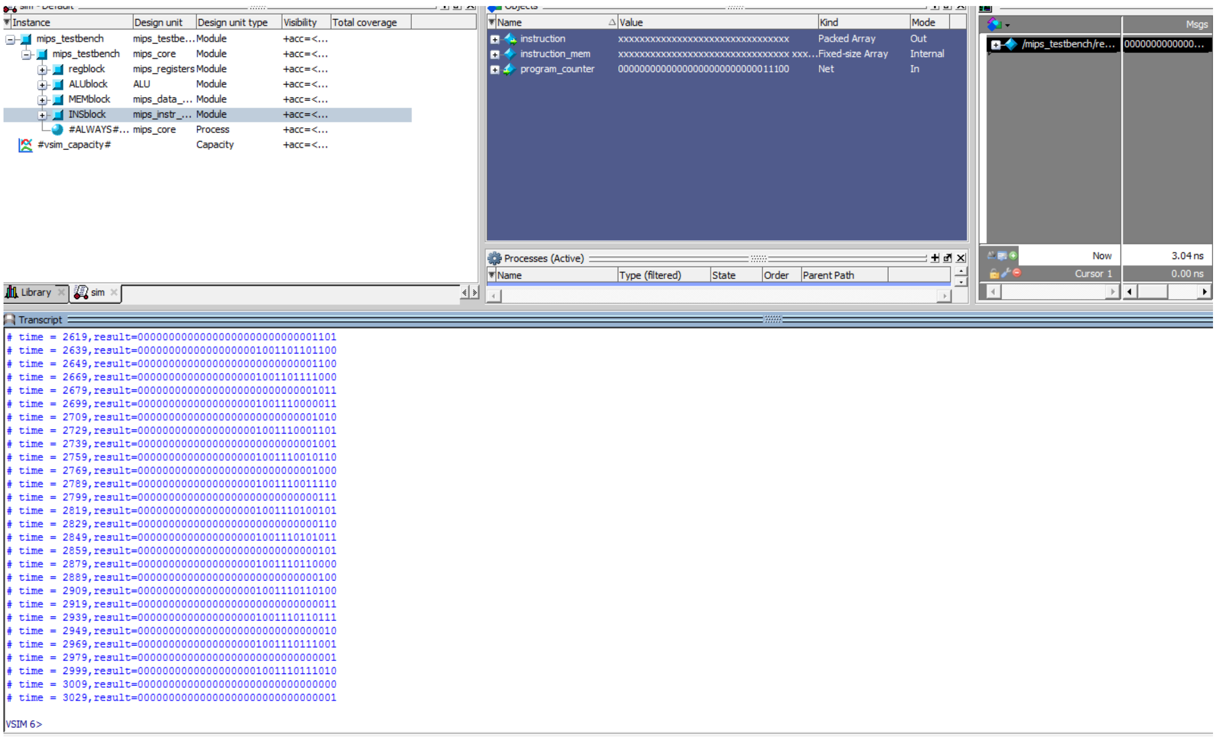
```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_data_mem_testbench/DataTB/data_mem
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000001100100
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
```

Registers.mem before executing. First 3 register must be 0.



```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_registers_testbench/regblock/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Execution. Last result is 01 because we are writing to that address.

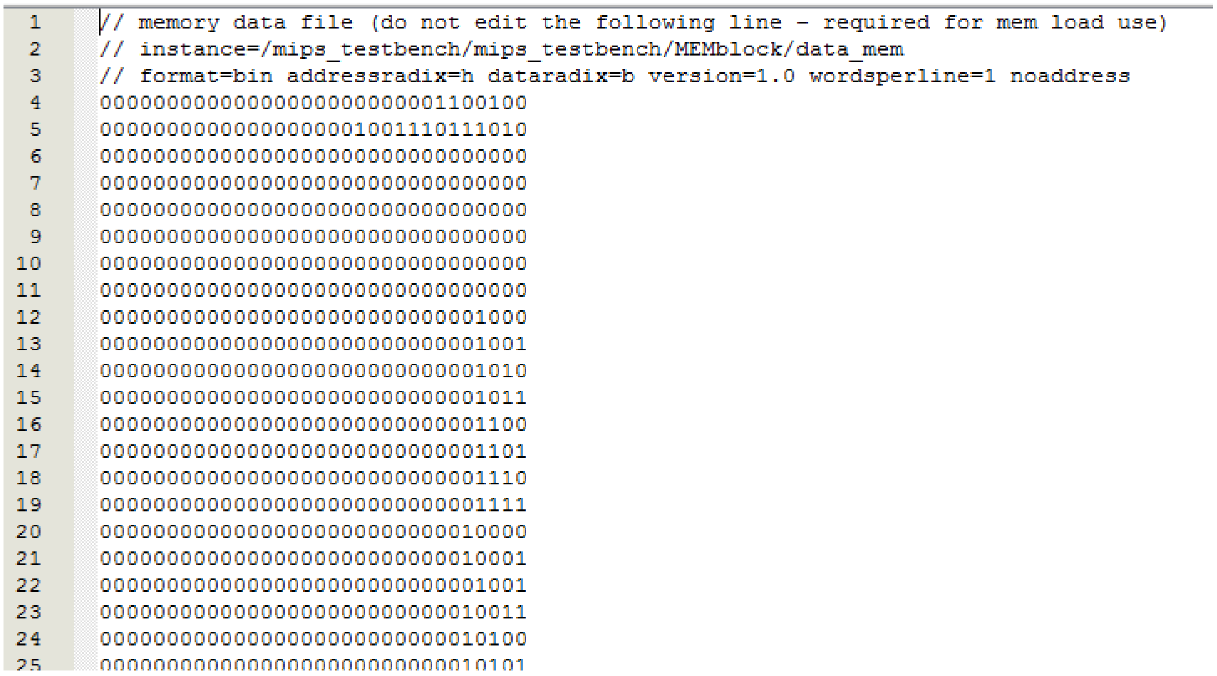


Instance	Design unit	Design unit type	Visibility	Total coverage
mips_testbench	mips_testbe…	Module	+acc=<…	
mips_testbench	mips_core	Module	+acc=<…	
regblock	mips_registers	Module	+acc=<…	
ALUblock	ALU	Module	+acc=<…	
MEMblock	mips_data_…	Module	+acc=<…	
INSblock	mips_instr_…	Module	+acc=<…	
#ALWAYS#…	mips_core	Process	+acc=<…	
#vsim_capacity#		Capacity	+acc=<…	

Name	Value	Kind	Mode
instruction	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Packed Array	Out
instruction_mem	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxx…	Fixed-size Array	Internal
program_counter	00000000000000000000000000011100	Net	In

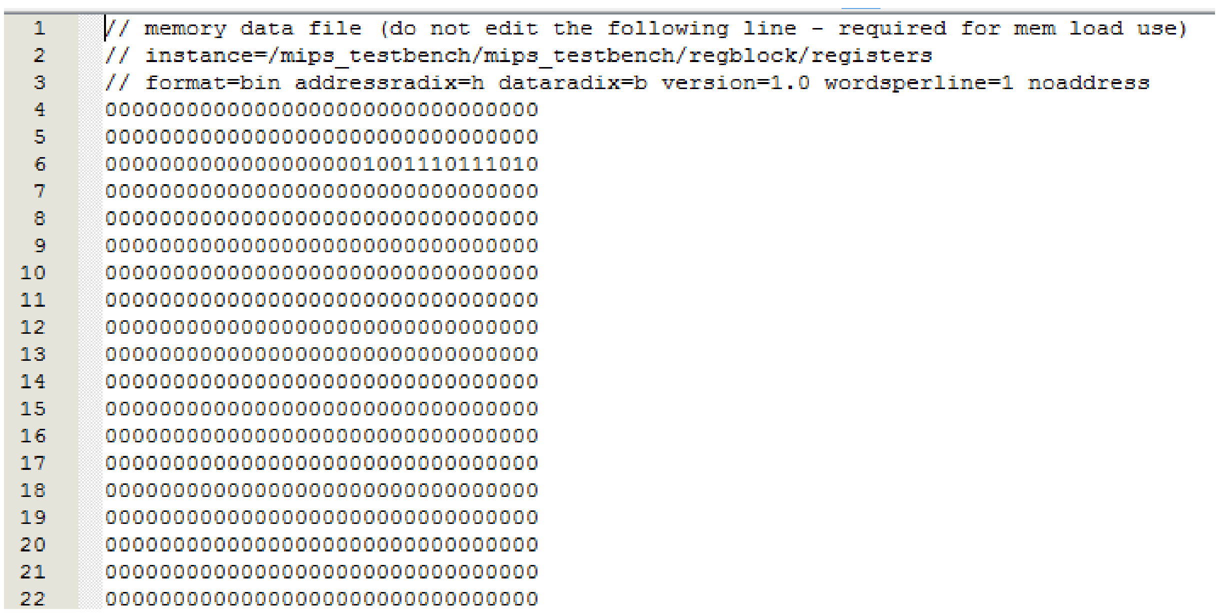
```
# time = 2619,result=00000000000000000000000000001101
# time = 2639,result=00000000000000000001001101101100
# time = 2649,result=00000000000000000000000000001100
# time = 2669,result=00000000000000000001001101111000
# time = 2679,result=00000000000000000000000000001011
# time = 2699,result=00000000000000000001001110000011
# time = 2709,result=00000000000000000000000000001010
# time = 2729,result=00000000000000000001001110001101
# time = 2739,result=00000000000000000000000000001001
# time = 2759,result=00000000000000000001001110010110
# time = 2769,result=00000000000000000000000000001000
# time = 2789,result=00000000000000000001001110011110
# time = 2799,result=00000000000000000000000000000111
# time = 2819,result=00000000000000000001001110100101
# time = 2829,result=00000000000000000000000000000110
# time = 2849,result=00000000000000000001001110101011
# time = 2859,result=00000000000000000000000000000101
# time = 2879,result=00000000000000000001001110110000
# time = 2889,result=00000000000000000000000000000100
# time = 2909,result=00000000000000000001001110110100
# time = 2919,result=00000000000000000000000000000011
# time = 2939,result=00000000000000000001001110110111
# time = 2949,result=00000000000000000000000000000010
# time = 2969,result=00000000000000000001001110111001
# time = 2979,result=00000000000000000000000000000001
# time = 2999,result=00000000000000000001001110111010
# time = 3009,result=00000000000000000000000000000000
# time = 3029,result=00000000000000000000000000000001

VSIM 6>
```

Data.mem after executing.



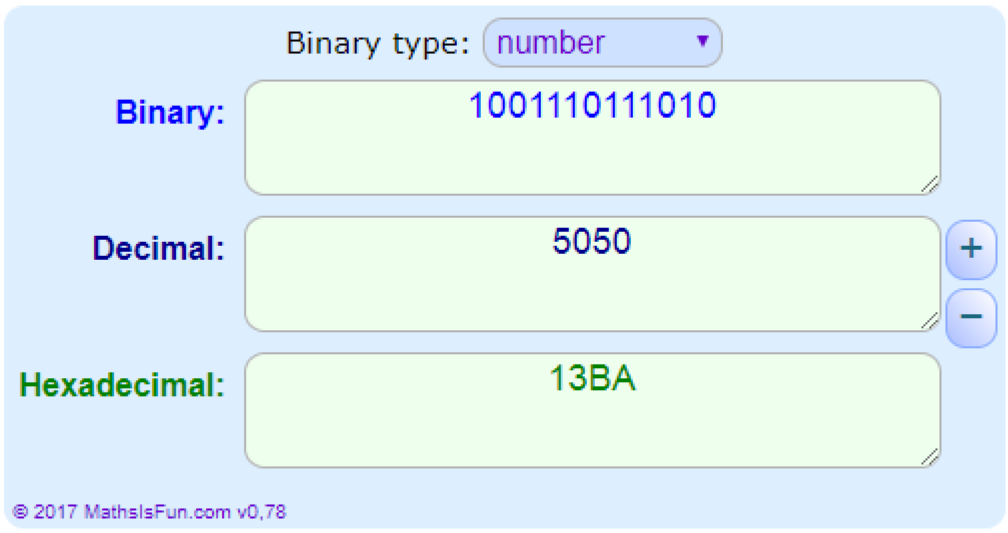
```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_testbench/mips_testbench/MEMblock/data_mem
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000001100100
00000000000000000001001110111010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000001001
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
```

Registers.mem after executing



```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_testbench/mips_testbench/regblock/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000001001110111010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Proof that our result is correct.



Binary type: number

Binary: 1001110111010

Decimal: 5050

Hexadecimal: 13BA

© 2017 MathsIsFun.com v0.78