Fatih Selim Yakar 161044054

COMPUTER ORGANIZATION HW4 REPORT

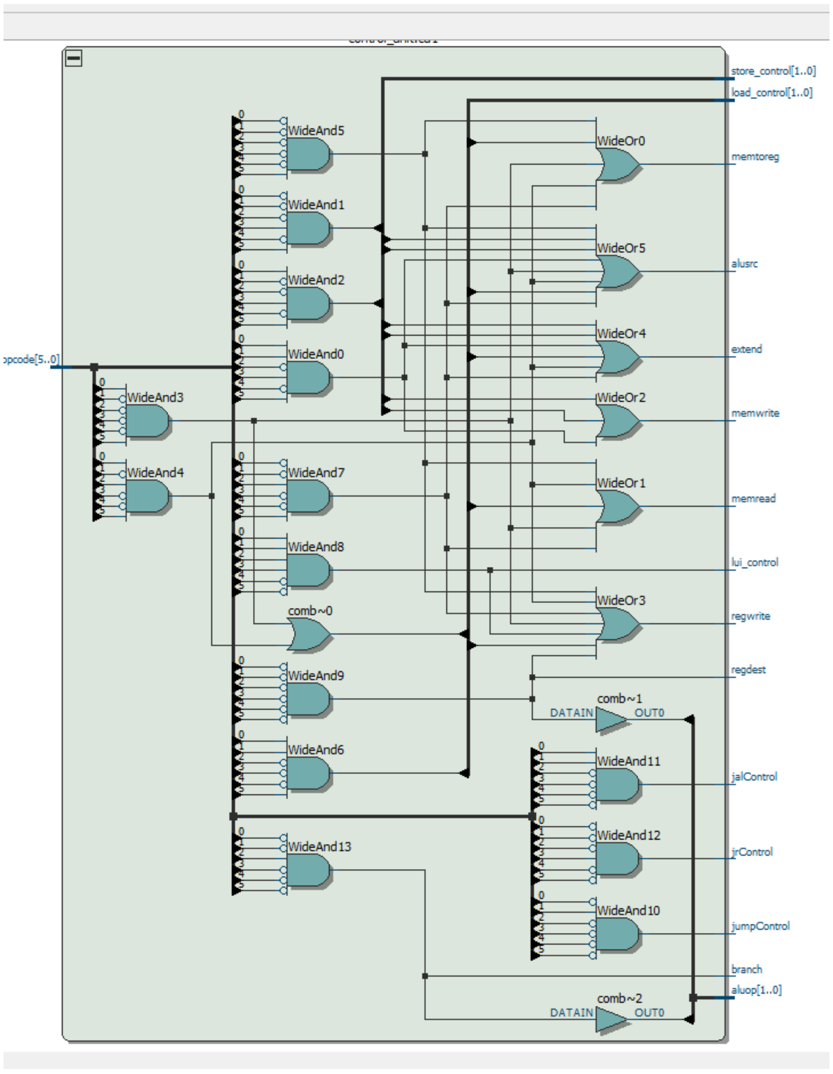
Modules(from the previous project):

- *module xor_gate(out,a,b);*
- *module mux(out,c0,c1,c2,c3,sel[1:0]);*
- *module full_adder(cout,out,cin,a,b);*
- *module one_bit_alu(b,a,less,cin,aluop[2:0],cout,out);*
- *module msb_one_bit_alu(b,a,less,cin,aluop[2:0],cout,out,set,v);*
- *module big_alu(b[31:0],a[31:0], aluop[2:0],cout,out[31:0],v);*
- *module mux2to1(out,c0,c1,sel);*
- *module mux2to1_32bit (out[31:0],c0[31:0],c1[31:0],sel);*
- *module mux4to1_32bit(out[31:0],c0[31:0],c1[31:0], c2[31:0], c3[31:0],sel[1:0);*
- *module mips_instruction(instruction[31:0], read_address[31:0]);*
- *module mips_memory(read_data[31:0],address [31:0],write_data [31:0], store_control_signal[1:0],input signal_mem_read,signal_mem_write,clk);*
- *module mips_registers(read_data_1[31:0], read_data_2[31:0], write_data[31:0], read_reg_1[4:0], read_reg_2[4:0], write_reg[4:0], signal_reg_write, clk);*
- *module extend(out[31:0],in[15:0],extend_signal);*
- *module load_unit(writeData[31:0],readDataMem[31:0],load_signal[1:0]);*
- *module lui_unit(writeData[31:0],immediate[15:0],memoryOut[31:0],lui_control_signal);*

Modules(from the new project):

- *module control_unit(memread,memtoreg,memwrite,regwrite,store_control[1:0],extend,lui_control,load_control[1:0],regdest,branch,alusrc,aluop[1:0],jumControl,jrControl, opcode[5:0]);*

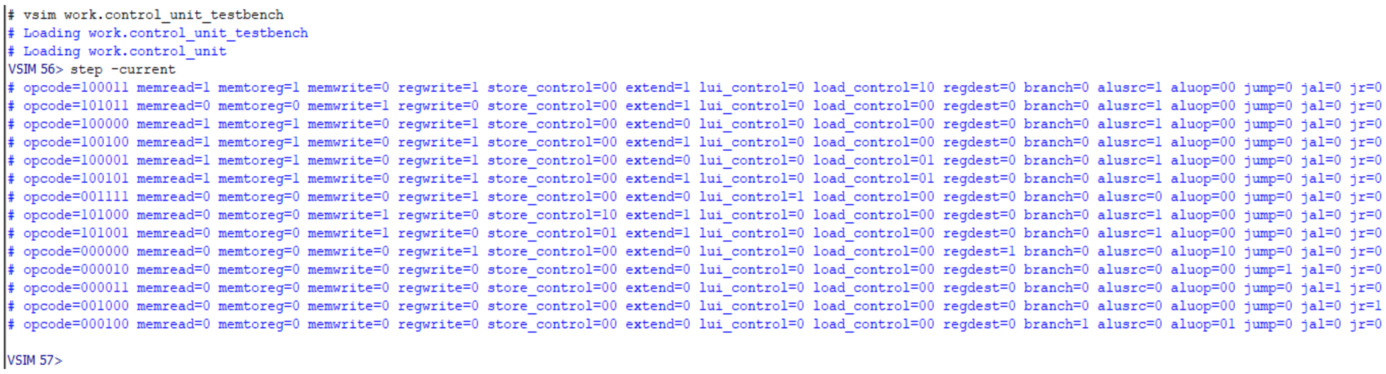
I modify the control unit additionally the I types, for R type instructions and J type instructions. So new control signals added and architecture are rearranged .



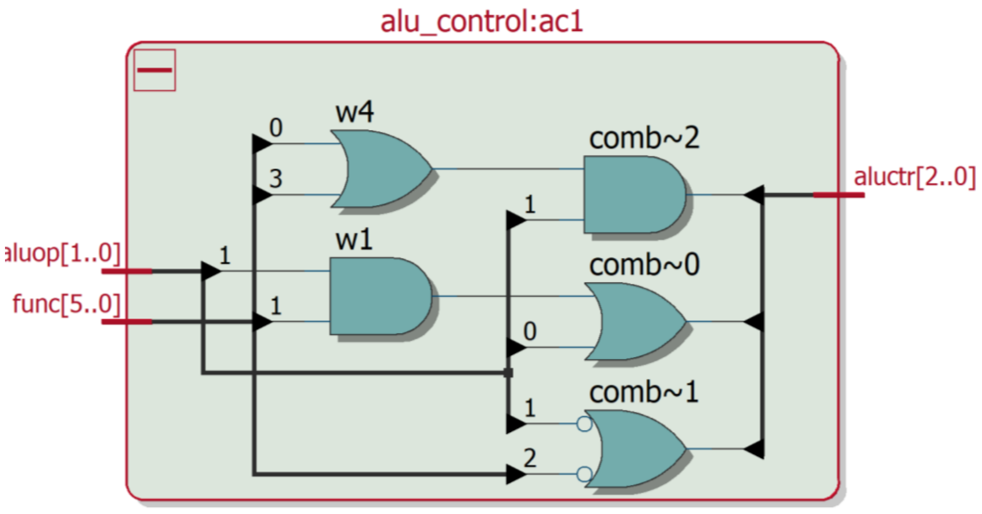
Şekil 1-Rearranged control unit

- *module alu_control(output [2:0] aluctr,input [1:0] aluop,input [5:0] func);*

In this Project I used it to decide what Alu did(sub,add,slt,or,and).I implement the structurally three equation for aluctr bits by looking aluop and function field.

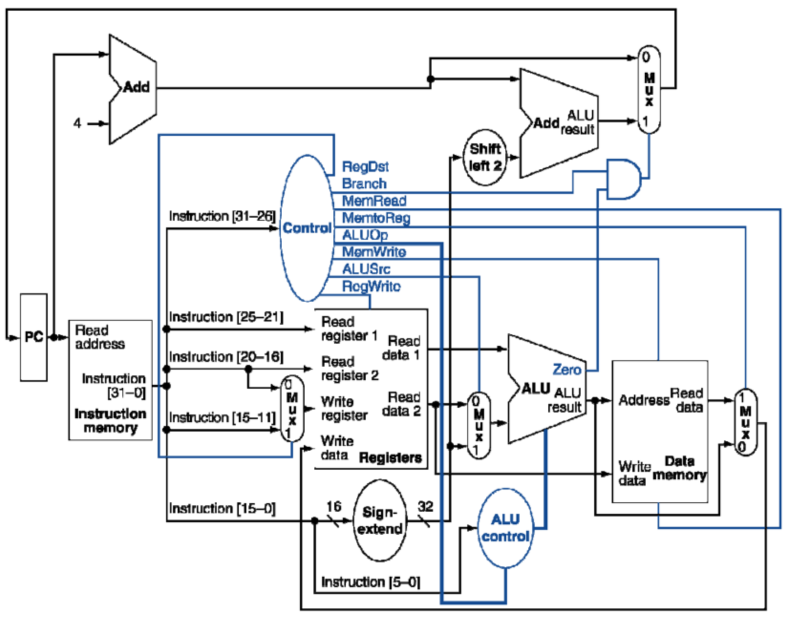

```
# vsim work.control_unit_testbench
# Loading work.control_unit_testbench
# Loading work.control_unit
VSIM 56> step -current
# opcode=100011 memread=1 memtoreg=1 memwrite=0 regwrite=1 store_control=00 extend=1 lui_control=0 load_control=10 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=101011 memread=0 memtoreg=0 memwrite=1 regwrite=0 store_control=00 extend=1 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=100000 memread=1 memtoreg=1 memwrite=0 regwrite=1 store_control=00 extend=0 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=100100 memread=1 memtoreg=1 memwrite=0 regwrite=1 store_control=00 extend=1 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=100001 memread=1 memtoreg=1 memwrite=0 regwrite=1 store_control=00 extend=0 lui_control=0 load_control=01 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=100101 memread=1 memtoreg=1 memwrite=0 regwrite=1 store_control=00 extend=1 lui_control=0 load_control=01 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=001111 memread=0 memtoreg=0 memwrite=0 regwrite=1 store_control=00 extend=0 lui_control=1 load_control=00 regdest=0 branch=0 alusrc=0 aluop=00 jump=0 jal=0 jr=0
# opcode=101000 memread=0 memtoreg=0 memwrite=1 regwrite=0 store_control=10 extend=1 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=101001 memread=0 memtoreg=0 memwrite=1 regwrite=0 store_control=01 extend=1 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=1 aluop=00 jump=0 jal=0 jr=0
# opcode=000000 memread=0 memtoreg=0 memwrite=0 regwrite=1 store_control=00 extend=0 lui_control=0 load_control=00 regdest=1 branch=0 alusrc=0 aluop=10 jump=0 jal=0 jr=0
# opcode=000010 memread=0 memtoreg=0 memwrite=0 regwrite=0 store_control=00 extend=0 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=0 aluop=00 jump=1 jal=0 jr=0
# opcode=000011 memread=0 memtoreg=0 memwrite=0 regwrite=0 store_control=00 extend=0 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=0 aluop=00 jump=0 jal=1 jr=0
# opcode=001000 memread=0 memtoreg=0 memwrite=0 regwrite=0 store_control=00 extend=0 lui_control=0 load_control=00 regdest=0 branch=0 alusrc=0 aluop=00 jump=0 jal=0 jr=1
# opcode=000100 memread=0 memtoreg=0 memwrite=0 regwrite=0 store_control=00 extend=0 lui_control=0 load_control=00 regdest=0 branch=1 alusrc=0 aluop=01 jump=0 jal=0 jr=0

VSIM 57>
```

Şekil 2-Testbench of controlunit



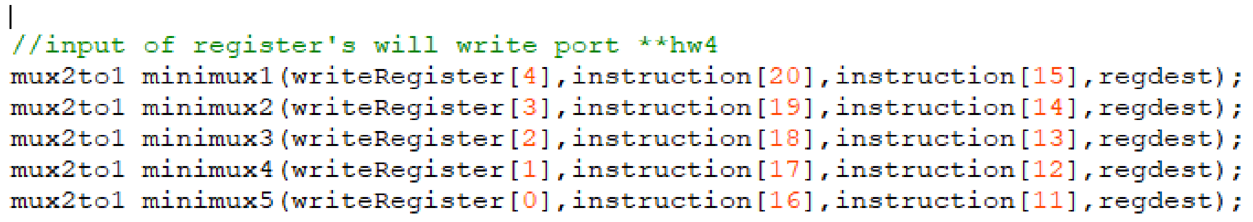
Şekil 3-Control unit's architecture

- **All other changes I have, are in the mips32 module.**



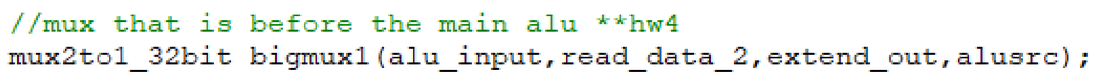
Şekil 4-It is not exactly true datapath. My datapath includes additionally lui_unit, load_unit and sign_unit for realizing other instructions.

1. I use these number of 1 bits muxes for selecting rd or rt register to entered register unit. So this selects rd register if regdest signal is 1, And selects rt register if regdest signal is 0. **In the datapath I implement the left of the register's mux.**



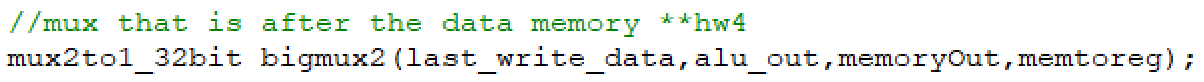
```
//input of register's will write port **hw4
mux2to1 minimux1(writeRegister[4],instruction[20],instruction[15],regdest);
mux2to1 minimux2(writeRegister[3],instruction[19],instruction[14],regdest);
mux2to1 minimux3(writeRegister[2],instruction[18],instruction[13],regdest);
mux2to1 minimux4(writeRegister[1],instruction[17],instruction[12],regdest);
mux2to1 minimux5(writeRegister[0],instruction[16],instruction[11],regdest);
```

2. I use these for selection alu_input, If alusrc is 0 then selects the rt_data else, then selects the sign extented immediate field. **In the datapath I implement the right of the register's mux.**



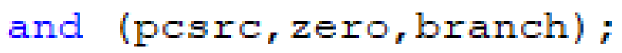
```
//mux that is before the main alu **hw4
mux2to1_32bit bigmux1(alu_input,read_data_2,extend_out,alusrc);
```

3. I use these for selecting the Alu output or memory read data(actually output of load_unit) by looking mem to reg signal. If signal is 1 then selects memory read data, else selects alu's output data. In the datapath I implement the right of the register's mux.



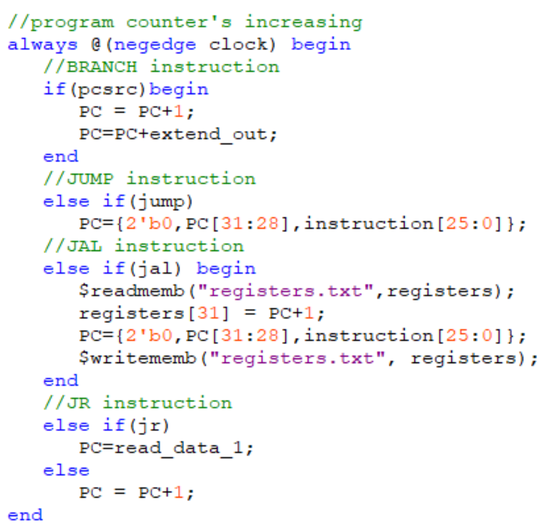
```
//mux that is after the data memory **hw4
mux2to1_32bit bigmux2(last_write_data,alu_out,memoryOut,memtoreg);
```

4. I use these for detecting branch process.



```
and (pcsrc,zero,branch);
```

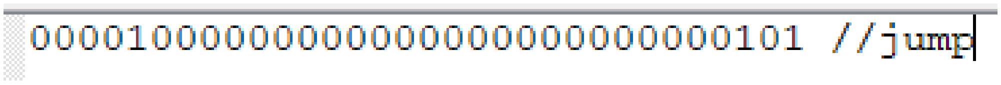
5. I use these for PC increasing and implementing jump,branch,jal,jr instructions.



```
//program counter's increasing
always @(negedge clock) begin
   //BRANCH instruction
   if(pcsrc)begin
      PC = PC+1;
      PC=PC+extend_out;
   end
   //JUMP instruction
   else if(jump)
      PC={2'b0,PC[31:28],instruction[25:0]};
   //JAL instruction
   else if(jal) begin
      $readmemb("registers.txt",registers);
      registers[31] = PC+1;
      PC={2'b0,PC[31:28],instruction[25:0]};
      $writememb("registers.txt", registers);
   end
   //JR instruction
   else if(jr)
      PC=read_data_1;
   else
      PC = PC+1;
end
```

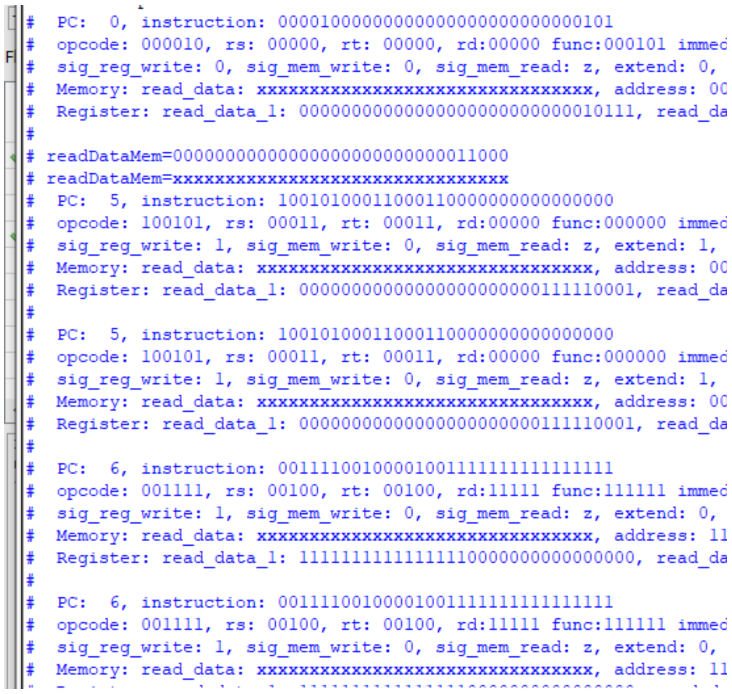
Instruction testbenchs

1-Jump instruction testbench



```
00001000000000000000000000000101 //jump
```

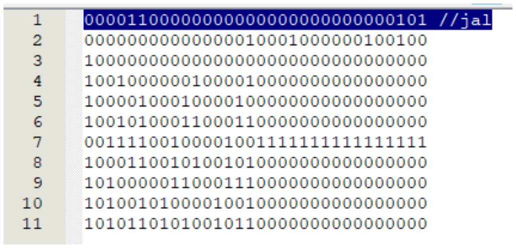
Attention: PC is increased



```
#  PC:  0, instruction: 00001000000000000000000000000101
#  opcode: 000010, rs: 00000, rt: 00000, rd:00000 func:000101 immed
#  sig_reg_write: 0, sig_mem_write: 0, sig_mem_read: z, extend: 0,
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00
#  Register: read_data_1: 00000000000000000000000000010111, read_da
#
# readDataMem=00000000000000000000000000011000
# readDataMem=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#  PC:  5, instruction: 10010100011000110000000000000000
#  opcode: 100101, rs: 00011, rt: 00011, rd:00000 func:000000 immed
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1,
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00
#  Register: read_data_1: 00000000000000000000000011110001, read_da
#
#  PC:  5, instruction: 10010100011000110000000000000000
#  opcode: 100101, rs: 00011, rt: 00011, rd:00000 func:000000 immed
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1,
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00
#  Register: read_data_1: 00000000000000000000000011110001, read_da
#
#  PC:  6, instruction: 00111100100001001111111111111111
#  opcode: 001111, rs: 00100, rt: 00100, rd:11111 func:111111 immed
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0,
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 11
#  Register: read_data_1: 11111111111111110000000000000000, read_da
#
#  PC:  6, instruction: 00111100100001001111111111111111
#  opcode: 001111, rs: 00100, rt: 00100, rd:11111 func:111111 immed
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0,
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 11
```

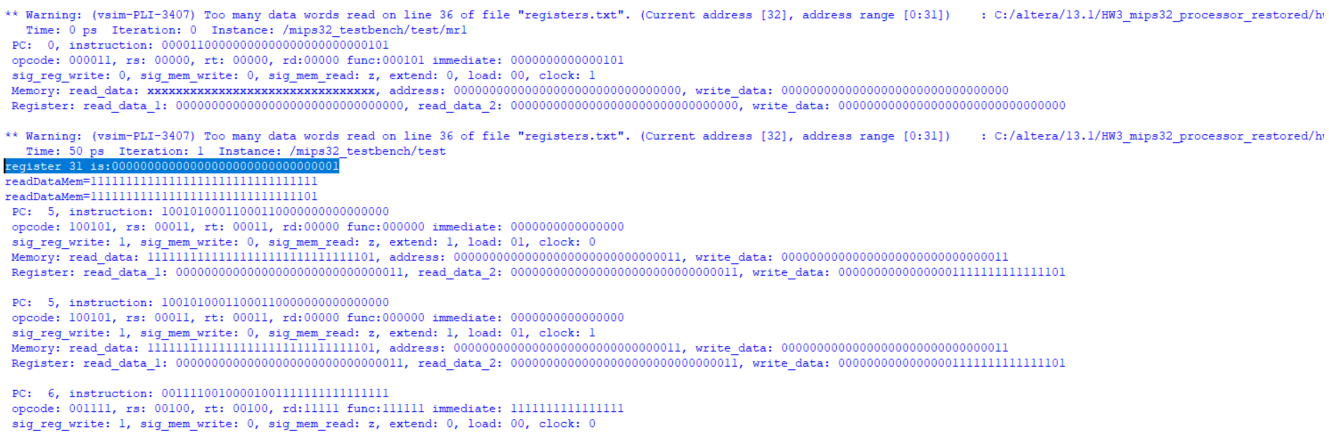
Şekil 5-There are not load instruction because of that read_data are xx.xx

2-Jal instruction testbench



```
00001100000000000000000000000101 //jal
00000000000000001000100000100100
10000000000000000000000000000000
10010000001000010000000000000000
10000100010000100000000000000000
10010100011000110000000000000000
00111100100001001111111111111111
10001100101001010000000000000000
10100000110001110000000000000000
10100101000010010000000000000000
10101101010010110000000000000000
```

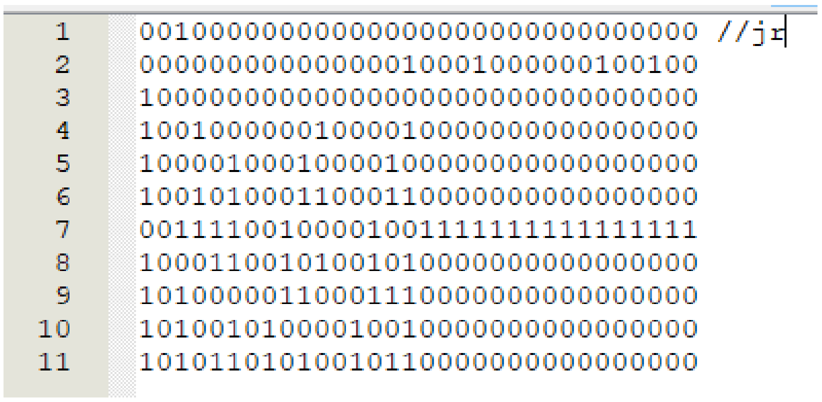
Attention: PC is increased and register 31 is PC+1



```
** Warning: (vsim-PLI-3407) Too many data words read on line 36 of file "registers.txt". (Current address [32], address range [0:31])    : C:/altera/13.1/HW3_mips32_processor_restored/h
   Time: 0 ps  Iteration: 0  Instance: /mips32_testbench/test/mrl
 PC:  0, instruction: 00001100000000000000000000000101
 opcode: 000011, rs: 00000, rt: 00000, rd:00000 func:000101 immediate: 0000000000000101
 sig_reg_write: 0, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
 Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00000000000000000000000000000000, write_data: 00000000000000000000000000000000
 Register: read_data_1: 00000000000000000000000000000000, read_data_2: 00000000000000000000000000000000, write_data: 00000000000000000000000000000000

** Warning: (vsim-PLI-3407) Too many data words read on line 36 of file "registers.txt". (Current address [32], address range [0:31])    : C:/altera/13.1/HW3_mips32_processor_restored/h
   Time: 50 ps  Iteration: 1  Instance: /mips32_testbench/test
register 31 is:00000000000000000000000000000001
readDataMem=11111111111111111111111111111111
readDataMem=11111111111111111111111111111101
 PC:  5, instruction: 10010100011000110000000000000000
 opcode: 100101, rs: 00011, rt: 00011, rd:00000 func:000000 immediate: 0000000000000000
 sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1, load: 01, clock: 0
 Memory: read_data: 11111111111111111111111111111101, address: 00000000000000000000000000000011, write_data: 00000000000000000000000000000011
 Register: read_data_1: 00000000000000000000000000000011, read_data_2: 00000000000000000000000000000011, write_data: 00000000000000001111111111111101

 PC:  5, instruction: 10010100011000110000000000000000
 opcode: 100101, rs: 00011, rt: 00011, rd:00000 func:000000 immediate: 0000000000000000
 sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1, load: 01, clock: 1
 Memory: read_data: 11111111111111111111111111111101, address: 00000000000000000000000000000011, write_data: 00000000000000000000000000000011
 Register: read_data_1: 00000000000000000000000000000011, read_data_2: 00000000000000000000000000000011, write_data: 00000000000000001111111111111101

 PC:  6, instruction: 00111100100001001111111111111111
 opcode: 001111, rs: 00100, rt: 00100, rd:11111 func:111111 immediate: 1111111111111111
 sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 0
```

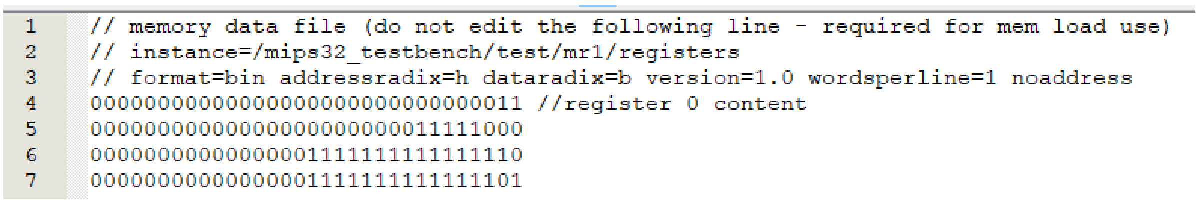
3-Jr instruction testbench

Gets the register 0’s content and jumps this



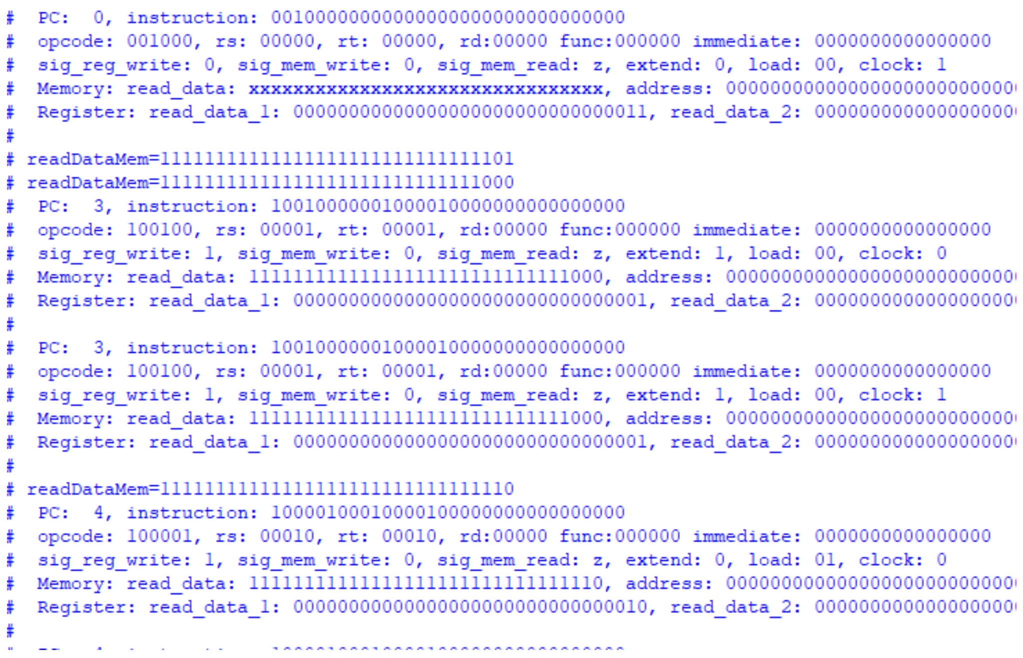
```
00100000000000000000000000000000 //jr
00000000000000010001000000100100
10000000000000000000000000000000
10010000001000010000000000000000
10000100010000100000000000000000
10010100011000110000000000000000
00111100100001001111111111111111
10001100101001010000000000000000
10100000110001110000000000000000
10100101000010010000000000000000
10101101010010110000000000000000
```

Register 0’s content is 3 so PC jumps the 3



```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000011 //register 0 content
00000000000000000000000011111000
00000000000000001111111111111110
00000000000000001111111111111101
```

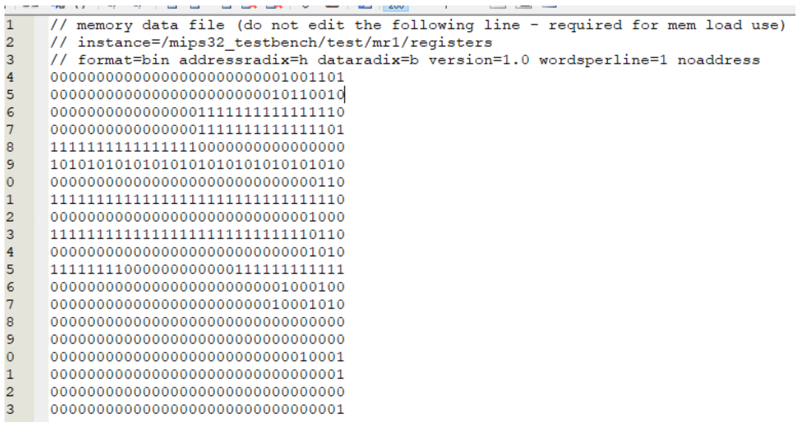
Attention: PC is 3.



```
#  PC:  0, instruction: 00100000000000000000000000000000
#  opcode: 001000, rs: 00000, rt: 00000, rd:00000 func:000000 immediate: 0000000000000000
#  sig_reg_write: 0, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 000000000000000000000000000
#  Register: read_data_1: 00000000000000000000000000000011, read_data_2: 000000000000000000
#
# readDataMem=11111111111111111111111111111101
# readDataMem=11111111111111111111111111111000
#  PC:  3, instruction: 10010000001000010000000000000000
#  opcode: 100100, rs: 00001, rt: 00001, rd:00000 func:000000 immediate: 0000000000000000
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1, load: 00, clock: 0
#  Memory: read_data: 11111111111111111111111111111000, address: 000000000000000000000000000
#  Register: read_data_1: 00000000000000000000000000000001, read_data_2: 000000000000000000
#
#  PC:  3, instruction: 10010000001000010000000000000000
#  opcode: 100100, rs: 00001, rt: 00001, rd:00000 func:000000 immediate: 0000000000000000
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1, load: 00, clock: 1
#  Memory: read_data: 11111111111111111111111111111000, address: 000000000000000000000000000
#  Register: read_data_1: 00000000000000000000000000000001, read_data_2: 000000000000000000
#
# readDataMem=11111111111111111111111111111110
#  PC:  4, instruction: 10000100010000100000000000000000
#  opcode: 100001, rs: 00010, rt: 00010, rd:00000 func:000000 immediate: 0000000000000000
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 01, clock: 0
#  Memory: read_data: 11111111111111111111111111111110, address: 000000000000000000000000000
#  Register: read_data_1: 00000000000000000000000000000010, read_data_2: 000000000000000000
#
```

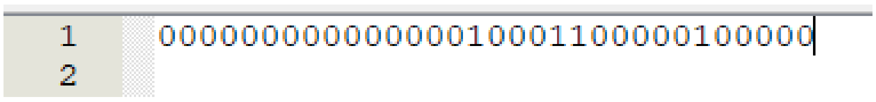
4-add instruction testbench

Before the add memory:

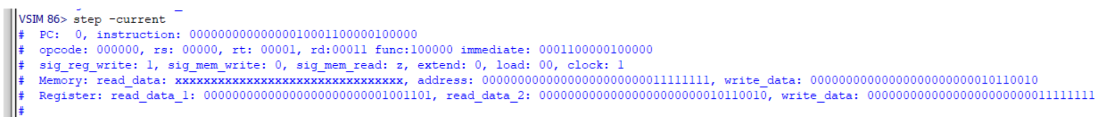


```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000001001101
00000000000000000000000010110010
00000000000000001111111111111110
00000000000000001111111111111101
11111111111111110000000000000000
10101010101010101010101010101010
00000000000000000000000000000110
11111111111111111111111111111110
00000000000000000000000000001000
11111111111111111111111111110110
00000000000000000000000000001010
11111111000000000000111111111111
00000000000000000000000001000100
00000000000000000000000010001010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000010001
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000001
```

Add instruction(sums 0th and 1st registers content and writes 3th register):

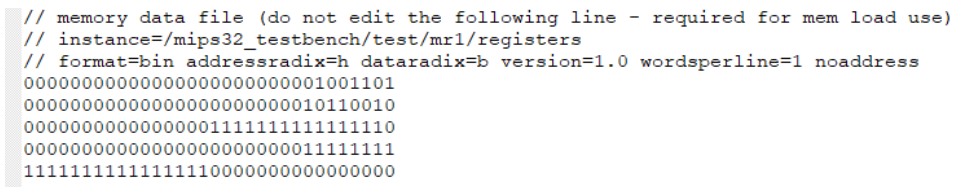


```
00000000000000010001100000100000
```



```
VSIM 86> step -current
#  PC:  0, instruction: 00000000000000010001100000100000
#  opcode: 000000, rs: 00000, rt: 00001, rd:00011 func:100000 immediate: 0001100000100000
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00000000000000000000000011111111, write_data: 00000000000000000000000010110010
#  Register: read_data_1: 00000000000000000000000001001101, read_data_2: 00000000000000000000000010110010, write_data: 00000000000000000000000011111111
#
```

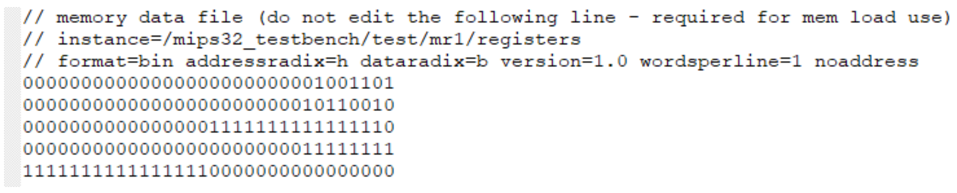
After the add:



```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000001001101
00000000000000000000000010110010
00000000000000001111111111111110
00000000000000000000000011111111
11111111111111110000000000000000
```

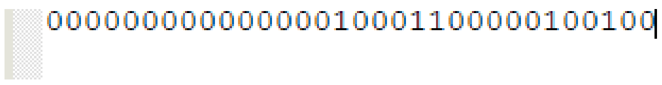
5-and instruction testbench

Before the and:

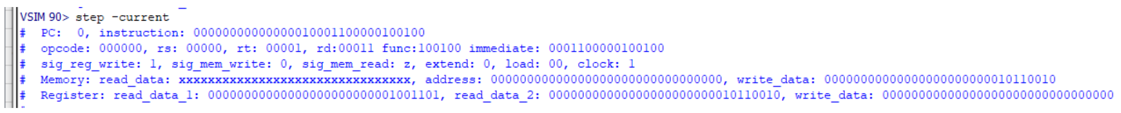


```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000001001101
00000000000000000000000010110010
00000000000000001111111111111110
00000000000000000000000011111111
11111111111111110000000000000000
```

And instruction(ands 0th and 1st registers content and writes 3th register):

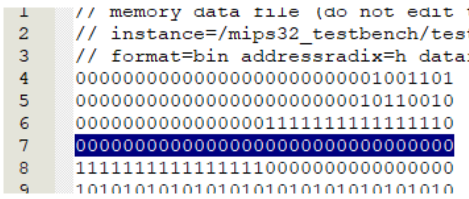


```
00000000000000010001100000100100
```



```
VSIM 90> step -current
#  PC:  0, instruction: 00000000000000010001100000100100
#  opcode: 000000, rs: 00000, rt: 00001, rd:00011 func:100100 immediate: 0001100000100100
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00000000000000000000000000000000, write_data: 00000000000000000000000010110010
#  Register: read_data_1: 00000000000000000000000001001101, read_data_2: 00000000000000000000000010110010, write_data: 00000000000000000000000000000000
```

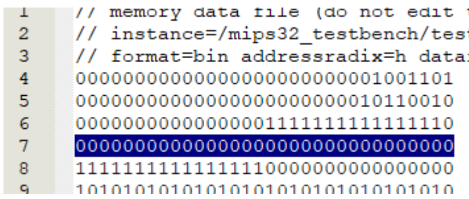
After and instruction:



```
// memory data file (do not edit the following line -
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0
00000000000000000000000001001101
00000000000000000000000010110010
00000000000000001111111111111110
00000000000000000000000000000000
11111111111111110000000000000000
10101010101010101010101010101010
```

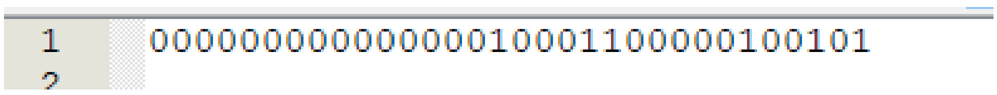
6-or instruction testbench

Before or instruction:

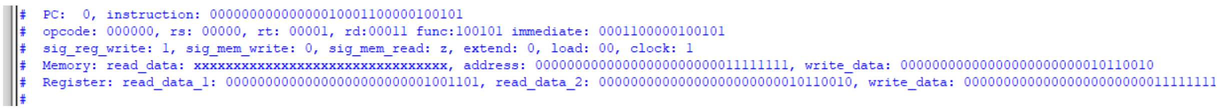


```
// memory data file (do not edit the following line -
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0
00000000000000000000000001001101
00000000000000000000000010110010
00000000000000001111111111111110
00000000000000000000000000000000
11111111111111110000000000000000
10101010101010101010101010101010
```

Or instruction(ors 0th and 1st registers content and writes 3th register):

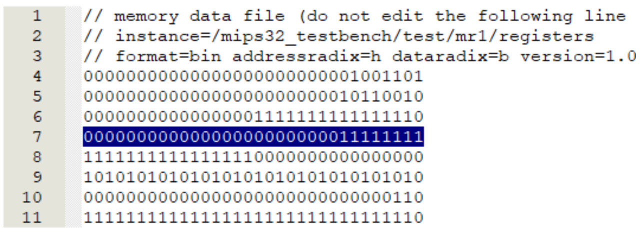


```
00000000000000010001100000100101
```



```
#  PC:  0, instruction: 00000000000000010001100000100101
#  opcode: 000000, rs: 00000, rt: 00001, rd:00011 func:100101 immediate: 0001100000100101
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00000000000000000000000011111111, write_data: 00000000000000000000000010110010
#  Register: read_data_1: 00000000000000000000000001001101, read_data_2: 00000000000000000000000010110010, write_data: 00000000000000000000000011111111
#
```

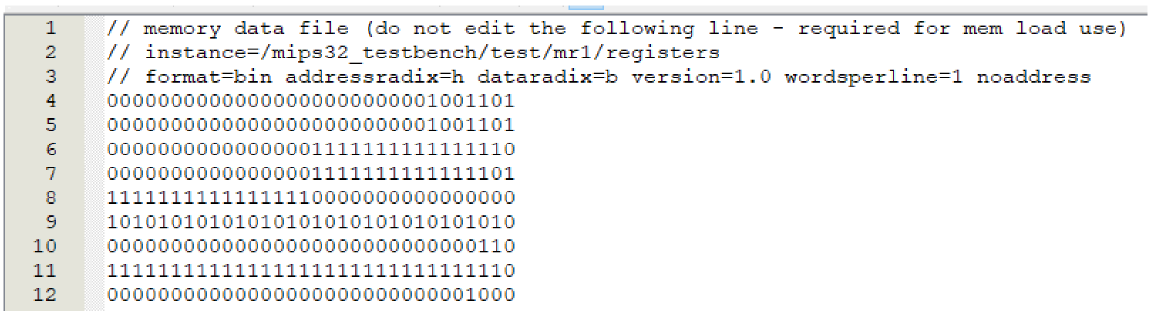
After or instruction:



```
// memory data file (do not edit the following line
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0
00000000000000000000000001001101
00000000000000000000000010110010
00000000000000001111111111111110
00000000000000000000000011111111
11111111111111110000000000000000
10101010101010101010101010101010
00000000000000000000000000000110
11111111111111111111111111111110
```

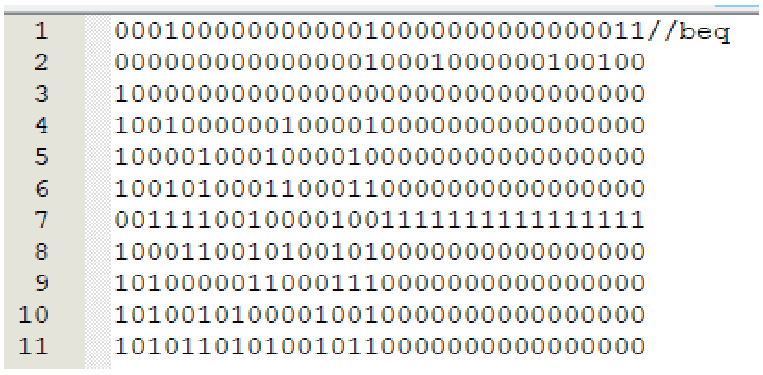
7-beq instruction testbench

Before the beq instruction:



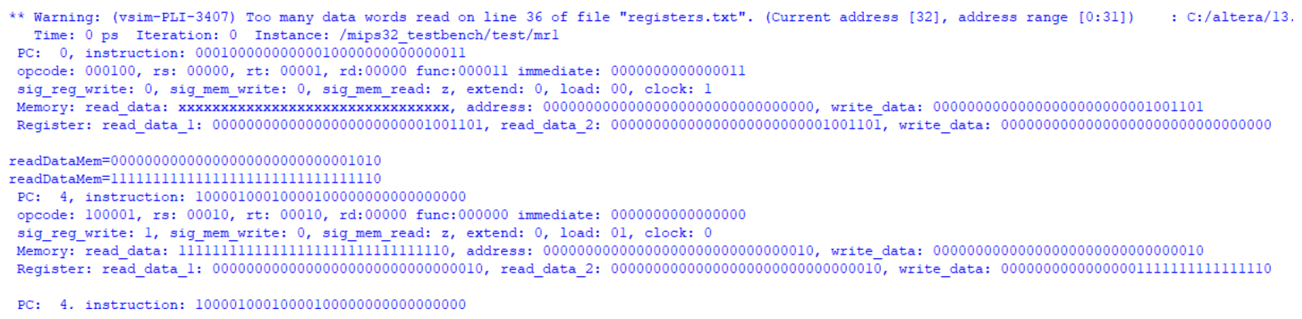
```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/test/mr1/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000001001101
00000000000000000000000001001101
00000000000000001111111111111110
00000000000000001111111111111101
11111111111111110000000000000000
10101010101010101010101010101010
00000000000000000000000000000110
11111111111111111111111111111110
00000000000000000000000000001000
```

Beq instruction(if 0th and 1th register is equal jumps the 11 + 1(dec 4):



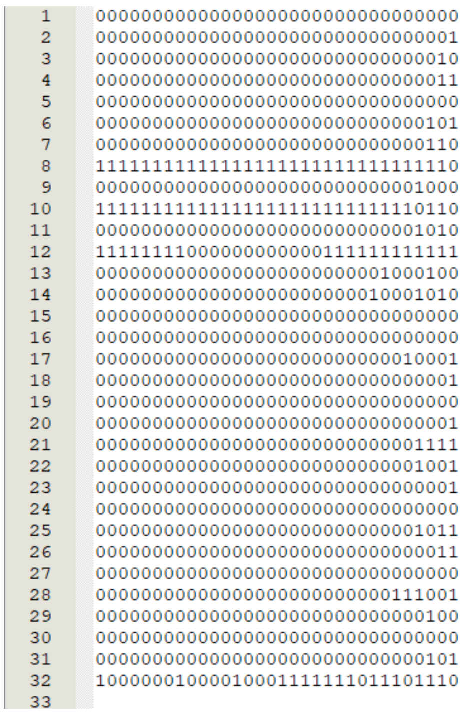
```
00010000000000001000000000000011//beq
00000000000000001000100000100100
10000000000000000000000000000000
10010000001000010000000000000000
10000100010000100000000000000000
10010100011000110000000000000000
00111100100001001111111111111111
10001100101001010000000000000000
10100000110001110000000000000000
10100101000010010000000000000000
10101101010010110000000000000000
```

Attention: pc is increasing because of the equivalence of 0th 1st register



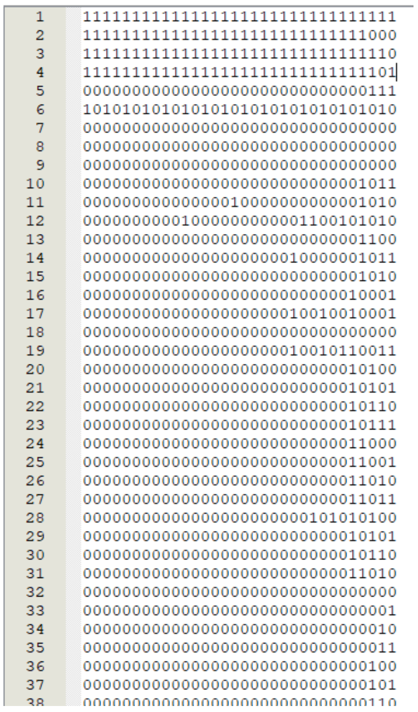
```
** Warning: (vsim-PLI-3407) Too many data words read on line 36 of file "registers.txt". (Current address [32], address range [0:31])    : C:/altera/13.
   Time: 0 ps  Iteration: 0  Instance: /mips32_testbench/test/mrl
 PC:  0, instruction: 00010000000000001000000000000011
 opcode: 000100, rs: 00000, rt: 00001, rd:00000 func:000011 immediate: 0000000000000011
 sig_reg_write: 0, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
 Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00000000000000000000000000000000, write_data: 00000000000000000000000001001101
 Register: read_data_1: 00000000000000000000000001001101, read_data_2: 00000000000000000000000001001101, write_data: 00000000000000000000000000000000

readDataMem=00000000000000000000000000001010
readDataMem=11111111111111111111111111111110
 PC:  4, instruction: 10000100010000100000000000000000
 opcode: 100001, rs: 00010, rt: 00010, rd:00000 func:000000 immediate: 0000000000000000
 sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 01, clock: 0
 Memory: read_data: 11111111111111111111111111111110, address: 00000000000000000000000000000010, write_data: 00000000000000000000000000000010
 Register: read_data_1: 00000000000000000000000000000010, read_data_2: 00000000000000000000000000000010, write_data: 00000000000000001111111111111110

 PC:  4. instruction: 10000100010000100000000000000000
```

Remaining testbenchs:



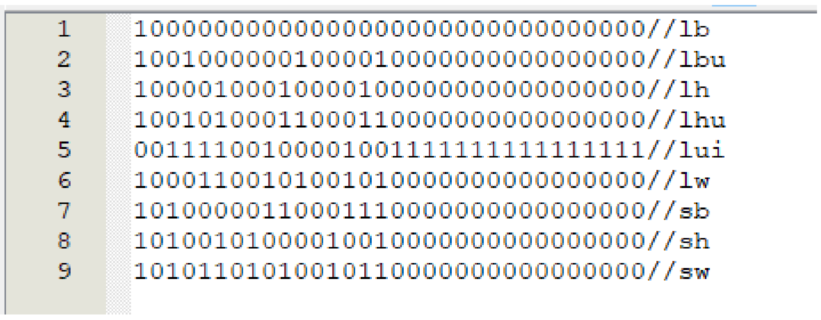
```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000000
00000000000000000000000000000101
00000000000000000000000000000110
11111111111111111111111111111110
00000000000000000000000000001000
11111111111111111111111111110110
00000000000000000000000000001010
11111111000000000000111111111111
00000000000000000000000001000100
00000000000000000000000010001010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000010001
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000001111
00000000000000000000000000001001
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000001011
00000000000000000000000000000011
00000000000000000000000000000000
00000000000000000000000000111001
00000000000000000000000000000100
00000000000000000000000000000000
00000000000000000000000000000101
10000001000010001111111011101110
```

Şekil 6-before the run register



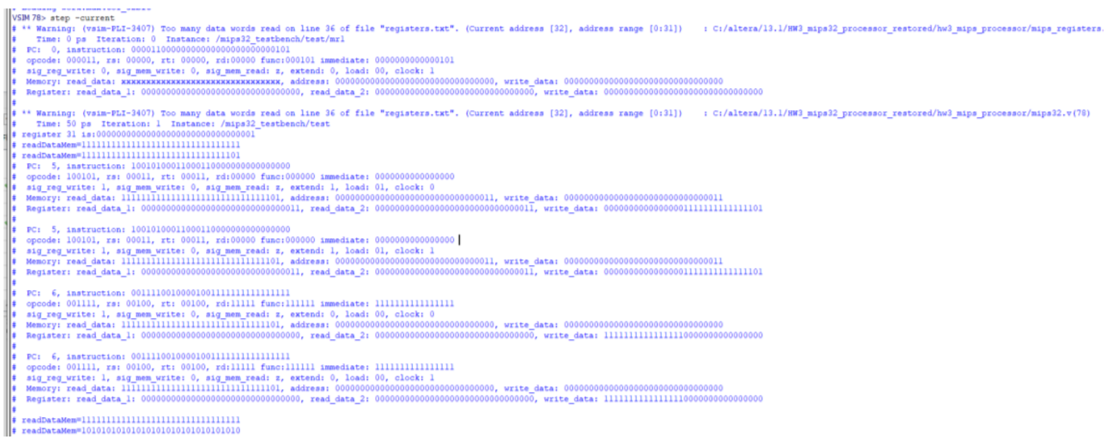
```
11111111111111111111111111111111
11111111111111111111111111111000
11111111111111111111111111111110
11111111111111111111111111111101
00000000000000000000000000000111
10101010101010101010101010101010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001011
00000000000000100000000000001010
000000000010000000000001100101010
00000000000000000000000000001100
00000000000000000000100000001011
00000000000000000000000000001010
00000000000000000000000000010001
0000000000000000000000010010010001
00000000000000000000000000000000
0000000000000000000000010010110011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000101010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000011010
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
```

Şekil 7-before the run register



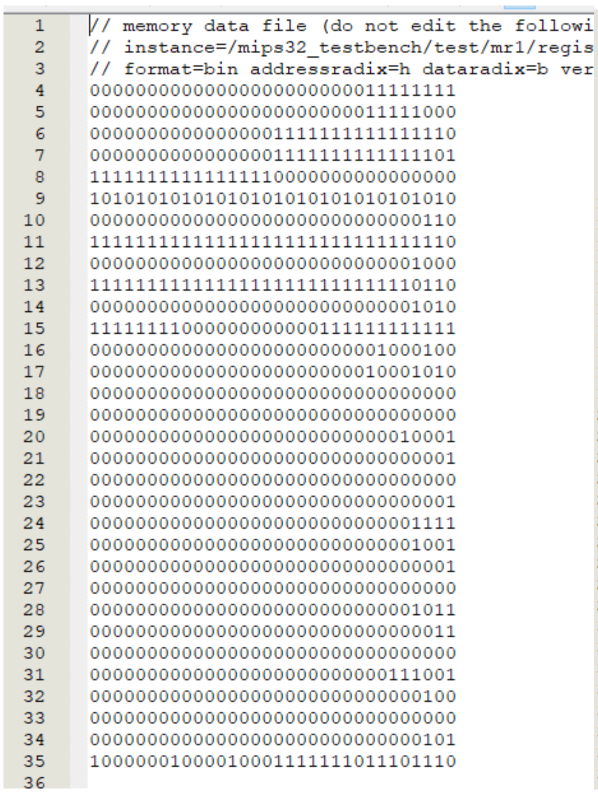
```
10000000000000000000000000000000//lb
10010000001000010000000000000000//lbu
10000100001000010000000000000000//lh
10010100011000110000000000000000//lhu
00111100100001001111111111111111//lui
10001100101001010000000000000000//lw
10100000110001110000000000000000//sb
10100101000010010000000000000000//sh
10101101010010110000000000000000//sw
```

Şekil 8-instruction memory

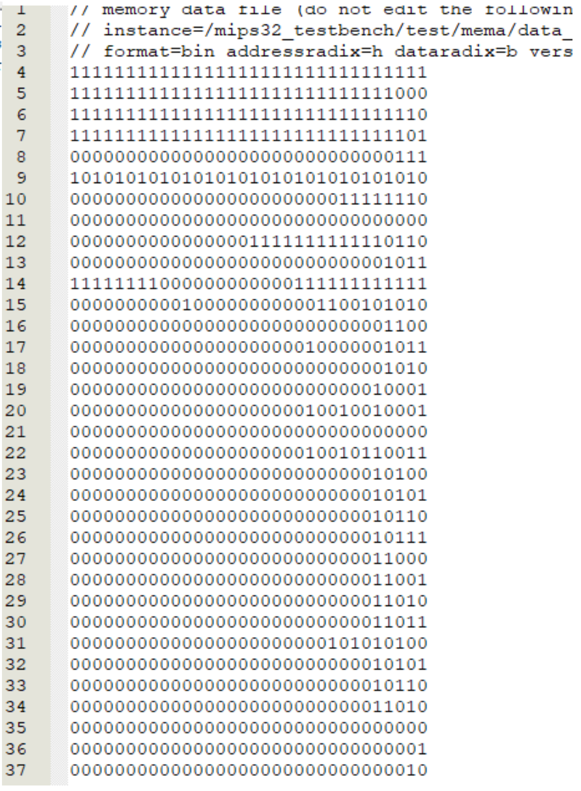


```
VSIM 78> step -current
# ** Warning: (vsim-PLI-3407) Too many data words read on line 36 of file "registers.txt". (Current address [32], address range [0:31])    : C:/altera/13.1/HW3_mips32_processor_restored/hw3_mips_processor/mips_registers.
#    Time: 0 ps  Iteration: 0  Instance: /mips32_testbench/test/mr1
#  PC:  0, instruction: 00001100000000000000000000000101
#  opcode: 000011, rs: 00000, rt: 00000, rd:00000 func:000101 immediate: 0000000000000101
#  sig_reg_write: 0, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
#  Memory: read_data: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address: 00000000000000000000000000000000, write_data: 00000000000000000000000000000000
#  Register: read_data_1: 00000000000000000000000000000000, read_data_2: 00000000000000000000000000000000, write_data: 00000000000000000000000000000000
#
# ** Warning: (vsim-PLI-3407) Too many data words read on line 36 of file "registers.txt". (Current address [32], address range [0:31])    : C:/altera/13.1/HW3_mips32_processor_restored/hw3_mips_processor/mips32.v(78)
#    Time: 50 ps  Iteration: 1  Instance: /mips32_testbench/test
# register 31 is:00000000000000000000000000000001
# readDataMem=11111111111111111111111111111111
# readDataMem=11111111111111111111111111111101
#  PC:  5, instruction: 10010100011000110000000000000000
#  opcode: 100101, rs: 00011, rt: 00011, rd:00000 func:000000 immediate: 0000000000000000
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1, load: 01, clock: 0
#  Memory: read_data: 11111111111111111111111111111101, address: 00000000000000000000000000000011, write_data: 00000000000000000000000000000011
#  Register: read_data_1: 00000000000000000000000000000011, read_data_2: 00000000000000000000000000000011, write_data: 00000000000000001111111111111101
#
#  PC:  5, instruction: 10010100011000110000000000000000
#  opcode: 100101, rs: 00011, rt: 00011, rd:00000 func:000000 immediate: 0000000000000000 |
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 1, load: 01, clock: 1
#  Memory: read_data: 11111111111111111111111111111101, address: 00000000000000000000000000000011, write_data: 00000000000000000000000000000011
#  Register: read_data_1: 00000000000000000000000000000011, read_data_2: 00000000000000000000000000000011, write_data: 00000000000000001111111111111101
#
#  PC:  6, instruction: 00111100100001001111111111111111
#  opcode: 001111, rs: 00100, rt: 00100, rd:11111 func:111111 immediate: 1111111111111111
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 0
#  Memory: read_data: 11111111111111111111111111111101, address: 00000000000000000000000000000000, write_data: 00000000000000000000000000000000
#  Register: read_data_1: 00000000000000000000000000000000, read_data_2: 00000000000000000000000000000000, write_data: 11111111111111110000000000000000
#
#  PC:  6, instruction: 00111100100001001111111111111111
#  opcode: 001111, rs: 00100, rt: 00100, rd:11111 func:111111 immediate: 1111111111111111
#  sig_reg_write: 1, sig_mem_write: 0, sig_mem_read: z, extend: 0, load: 00, clock: 1
#  Memory: read_data: 11111111111111111111111111111101, address: 00000000000000000000000000000000, write_data: 00000000000000000000000000000000
#  Register: read_data_1: 00000000000000000000000000000000, read_data_2: 00000000000000000000000000000000, write_data: 11111111111111110000000000000000
#
# readDataMem=11111111111111111111111111111111
# readDataMem=10101010101010101010101010101010
```

After the run: register and memory



```
// memory data file (do not edit the followi
// instance=/mips32_testbench/test/mr1/regis
// format=bin addressradix=h dataradix=b ver
00000000000000000000000011111111
00000000000000000000000011111000
00000000000000001111111111111110
00000000000000001111111111111101
11111111111111110000000000000000
10101010101010101010101010101010
00000000000000000000000000000110
11111111111111111111111111111110
00000000000000000000000000001000
11111111111111111111111111110110
00000000000000000000000000001010
11111111000000000000111111111111
00000000000000000000000001000100
00000000000000000000000010001010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000010001
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000001111
00000000000000000000000000001001
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000001011
00000000000000000000000000000011
00000000000000000000000000000000
00000000000000000000000000111001
00000000000000000000000000000100
00000000000000000000000000000000
00000000000000000000000000000101
10000001000010001111111011101110
```



```
// memory data file (do not edit the followin
// instance=/mips32_testbench/test/mema/data_
// format=bin addressradix=h dataradix=b vers
11111111111111111111111111111111
11111111111111111111111111111000
11111111111111111111111111111110
11111111111111111111111111111101
00000000000000000000000000000111
10101010101010101010101010101010
00000000000000000000000001111110
00000000000000000000000000000000
00000000000000001111111111110110
00000000000000000000000000001011
11111111000000000000111111111111
00000000010000000000001100101010
00000000000000000000000000001100
00000000000000000000010000001011
00000000000000000000000000001010
00000000000000000000000000010001
00000000000000000000010010010001
00000000000000000000000000000000
00000000000000000000010010110011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000101010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000011010
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
```