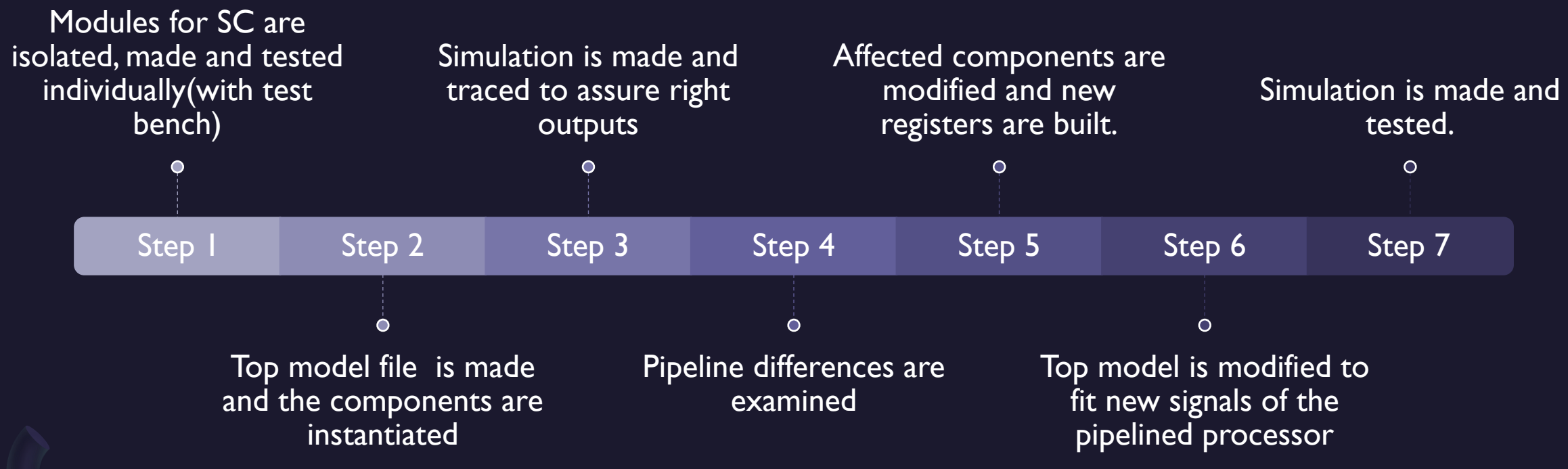
Single cycle & Pipelined MIPS processor

Table of contents

MIPS processor

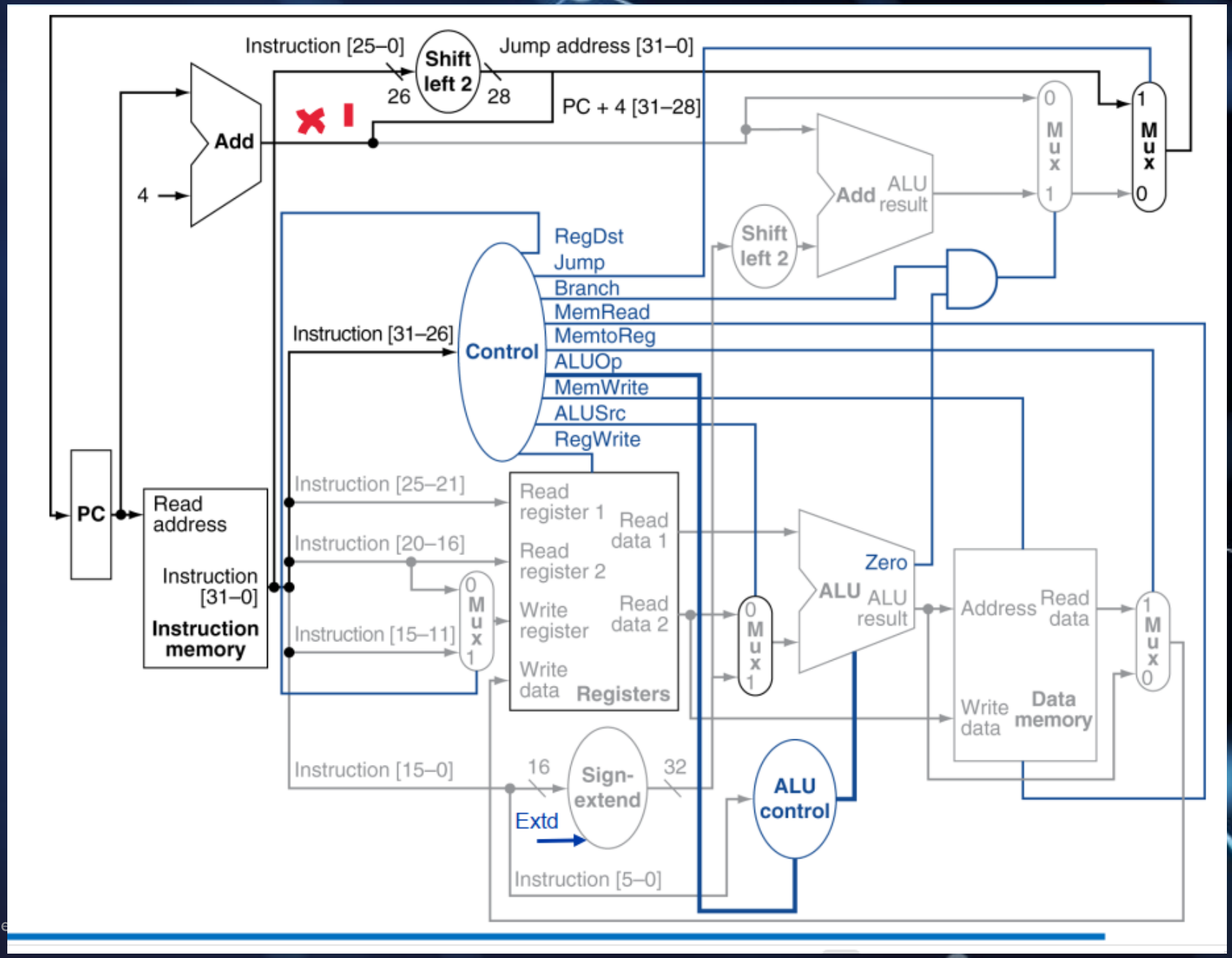
- MIPS processor is built on RISC (Reduced Instruction Set Architecture) and uses a limited amount of fixed length instructions, instructions do a single task, usually in a single cycle, and the downfalls of the slower speed is mitigated using pipelining.

Timeline


Modules for SC are isolated, made and tested individually(with test bench)
Simulation is made and traced to assure right outputs
Affected components are modified and new registers are built.
Simulation is made and tested.
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
Step 7
Top model file is made and the components are instantiated
Pipeline differences are examined
Top model is modified to fit new signals of the pipelined processor

Single cycle

Schematic of SC MIPS



Modules

Instruction memory

Register memory

Data memory

Mux_5 bits , mux_32 bits

Alu

Sign extender

Control unit

Instruction fetch

Top level module

Instruction Memory

```
1  module instruction_mem(readaddr,instruction);
2  input [31:0] readaddr;
3  output reg [31:0] instruction;
4
5  reg [7:0] mem[1023:0];
6
7  always @(*) begin
8      instruction={mem[readaddr],mem[readaddr+1],mem[readaddr+2],mem[readaddr+3]};
9  end
10
11 endmodule
```

Instruction Memory: test bench

```
1  module instruction_mem_tb();
2  reg [31:0] readaddr;
3  wire [31:0] instruction;
4  integer i;
5  instruction_mem m1(readaddr,instruction);
6  initial begin
7      $readmemh("mem.dat",m1.mem);
8      readaddr=0;
9      #10;
10     readaddr=4;
11     #2 $stop;
12 end
13
14 endmodule
```

Register Memory

registerfile.v

```
1  module registerfile(Readreg1,Readreg2,WriteReg,WriteData, RegWrite,ReadData1,ReadData2,clk);
2  input clk,RegWrite;
3  input [4:0] Readreg1,Readreg2,WriteReg;
4  input [31:0] WriteData;
5  output [31:0] ReadData1,ReadData2;
6  //here we make the mem registers
7  reg [31:0] regs_file [31:0];
8
9  assign ReadData1=regs_file[Readreg1];
10 assign ReadData2=regs_file[Readreg2];
11
12 always@(posedge clk) begin
13     if(RegWrite) begin
14         regs_file[WriteReg]<=WriteData;
15     end
16 end
17
18 endmodule
```

Data Memory

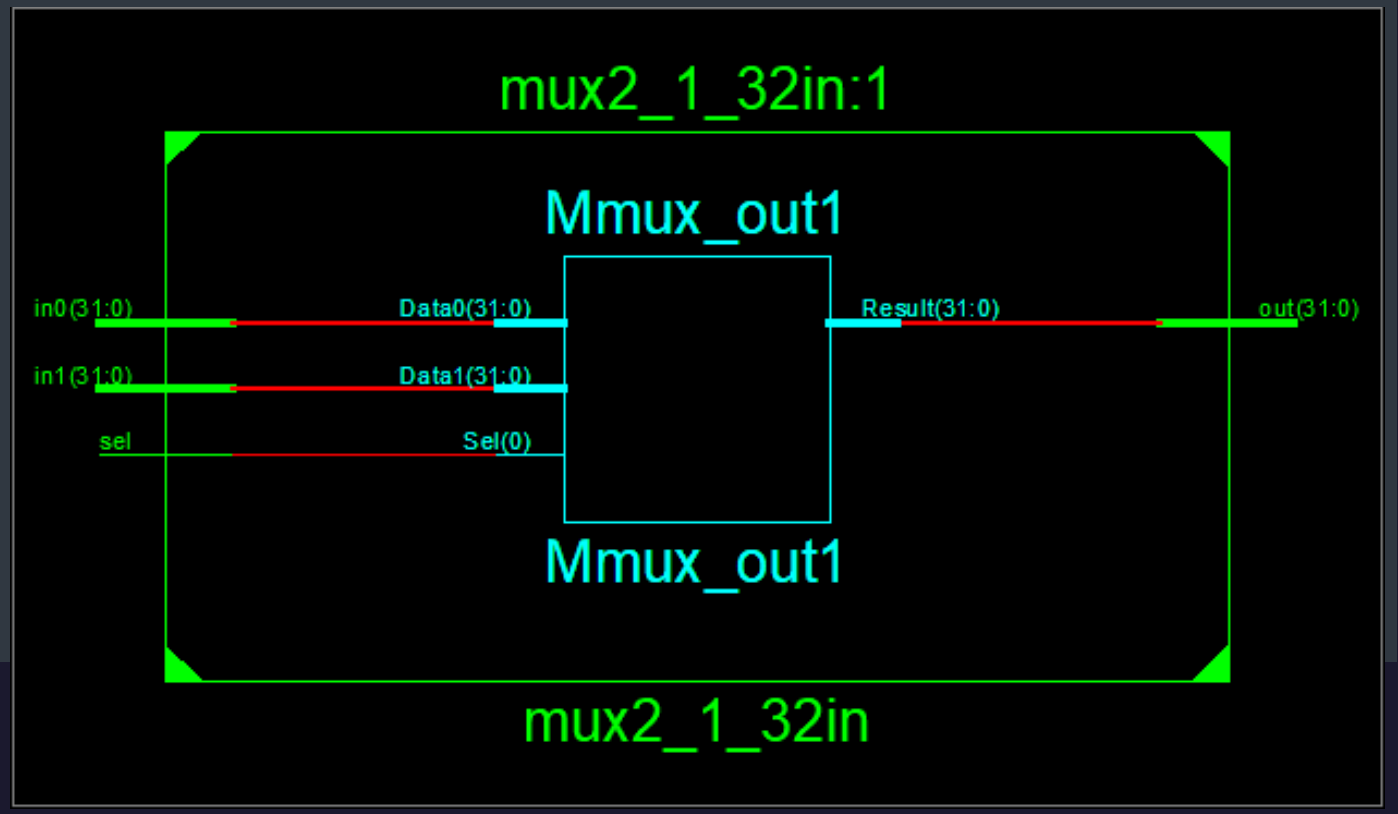
```
1  module data_memory(Addr,write_data,data_out,mem_read,mem_write,clk);
2  input [31:0] Addr,write_data;
3  output reg [31:0] data_out;
4  input mem_read,mem_write;
5  input clk;
6
7  reg [7:0] mem[1023:0];
8
9  always @(posedge clk) begin
10     if(mem_write)
11     {mem[Addr],mem[Addr+1],mem[Addr+2],mem[Addr+3]}<=write_data;
12 end
13 always @(*) begin
14     if(mem_read)
15     data_out<={mem[Addr],mem[Addr+1],mem[Addr+2],mem[Addr+3]};
16 end
17 endmodule
```

Data Memory: test bench

```
1  module data_memory_tb();
2  reg [31:0] Addr,write_data;
3  wire [31:0] data_out;
4  reg mem_read,mem_write;
5  reg clk;
6  //clk generation
7  initial begin
8      clk=0;
9      forever
10     #1 clk=~clk;
11 end
12 data_memory md1(Addr,write_data,data_out,mem_read,mem_write,clk);
13
14
15 initial begin
16     $readmemh("mem2.dat",md1.mem);
17     //test write operation
18     mem_write=1;
19     mem_read=0;
20     Addr=16;
21     write_data=$random;
22     #10;
23     //test read operation
24     mem_write=0;
25     mem_read=1;
26     Addr=4;
27     #10;
28     #2 $stop;
29 end
30
31 endmodule
```

Mux_32 bits

```
1  module mux2_1_32in(in0,in1,sel,out);
2  input [31:0] in1,in0;
3  input sel;
4  output [31:0] out;
5
6  assign out=(sel==1)?in1:in0;
7
8  endmodule
```

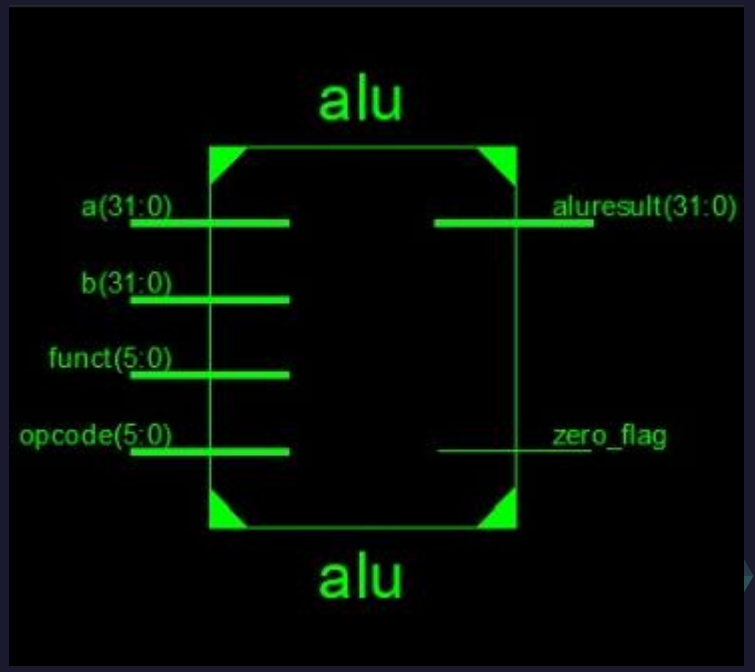


Mux_5 bits

```
1  module mux2_1_6in(in0,in1,sel,out);
2  input [4:0] in1,in0;
3  input sel;
4  output [4:0] out;
5
6  assign out=(sel==1)?in1:in0;
7
8  endmodule
```

Arithmetic Logic Unit(ALU)

```
1  module alu(opcode,funct,a,b,aluresult,zero_flag);
2  input [5:0] opcode,funct;
3  input signed [31:0] a,b;
4  output reg signed [31:0] aluresult;
5  output zero_flag;
6
7  always @(*) begin
8      if(opcode==0 && funct=='h20)//add
9      aluresult=a+b;
10     else if(opcode=='h8)       //addi
11     aluresult=a+b;
12     else if(opcode==0 && funct=='h22) //sub
13     aluresult=a-b;
14     else if(opcode==0 && funct=='h24) //and
15     aluresult=a&b;
16     else if(opcode==0 && funct=='h25) //or
17     aluresult=a|b;
18     else if(opcode==0 && funct=='h2a) //slt
19     aluresult=(a<b)?1:0;
20     else if(opcode=='h23)            //lw
21     aluresult=a+b;
22     else if(opcode=='h2b)            //sw
23     aluresult=a+b;
24     else if(opcode=='h4) //beq
25     aluresult=a-b;
26     else
27     aluresult=10;//dummyvalue
28 end
29
30 assign zero_flag=(aluresult==32'h0000)?1'b1:1'b0;
31
32 endmodule
```

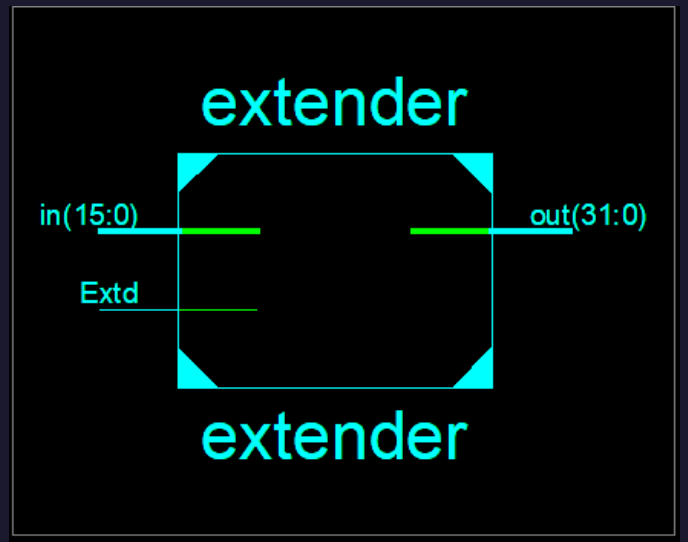


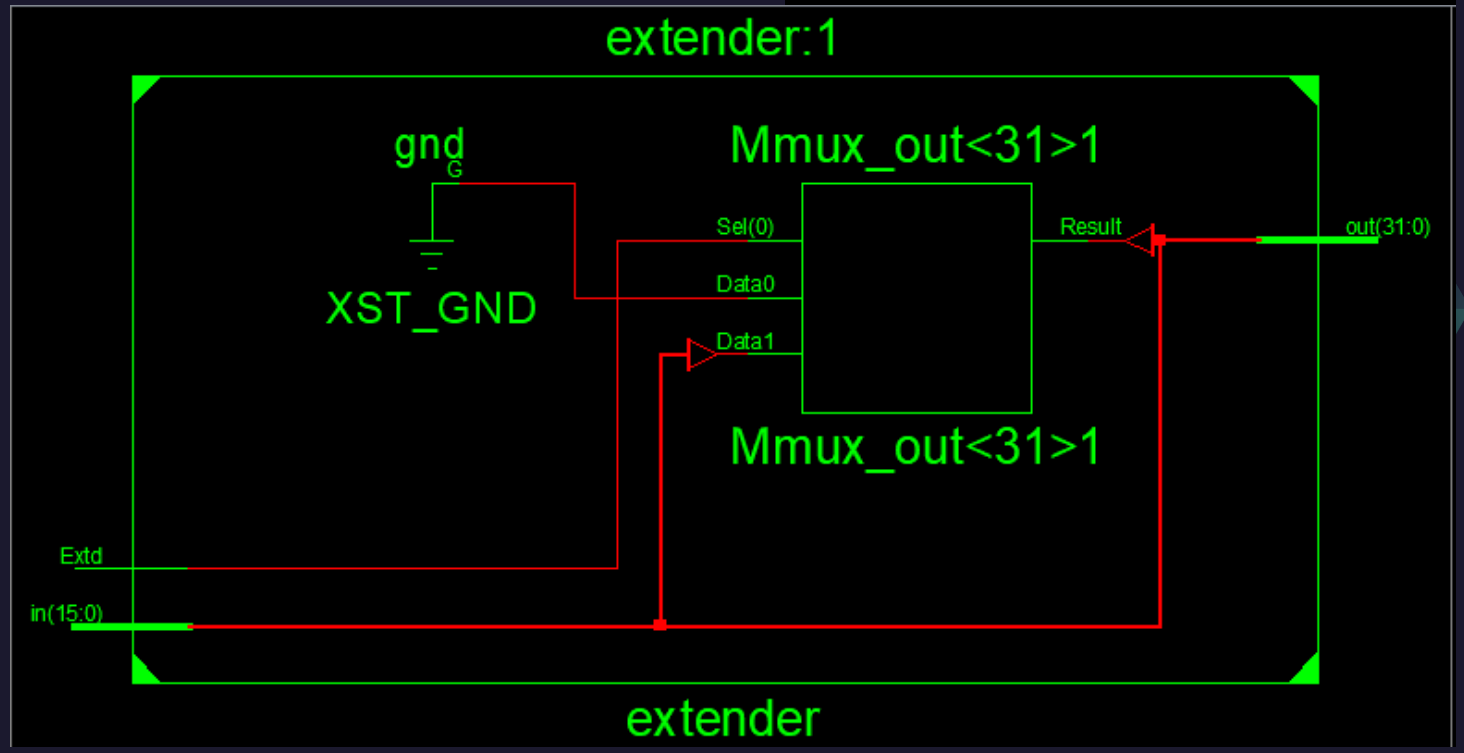
ALU (testbench)

```
module alu_tb();
reg [5:0] opcode,funct;
reg signed [31:0] a,b;
wire signed [31:0] aluresult;
wire zero_flag;

alu a1(opcode,funct,a,b,aluresult,zero_flag);

initial begin
    a=200;
    b=300;
    opcode=0;
    funct='h20;//test add
    #2
    a=100;
    b=-100;
    opcode=0;
    funct='h22;//test sub
    #2
    a=100;
    b=100;
    opcode='h4;
    funct=$random;//test beq
    #2
    a=1000;
    b=50;
    opcode='h8;
    funct=$random;//test addi
    #2
    a=$random;
    b=$random;
    opcode=0;
    funct='h24;//test and
    #2
    a=$random;
    b=$random;
    opcode=0;
    funct='h25;//test or
    #2
    a=50;
    b=100;
    opcode=0;
    funct='h2a;//test slt
    #2
    a=20;
    b=30;
    opcode='h2b;
    funct=$random;//test sw
    #2
    a=80;
    b=70;
    opcode='h23;
    funct=$random;//test lw
    #10
    $stop;
end
initial begin
    $monitor("a=%d b=%d aluresult=%d zero_flag=%h",a,b,aluresult,zero_flag);
end

endmodule
```

Sign Extender

```
1  module extender(in,out,Extd);
2  input [15:0] in;
3  output reg [31:0] out;
4  input Extd;
5
6  always @(*) begin
7      if(Extd)
8      out={{16{in[15]}},in};
9      else
10     out={16'h0000,in};
11 end
12
13 endmodule
```





Sign Extender: testbench

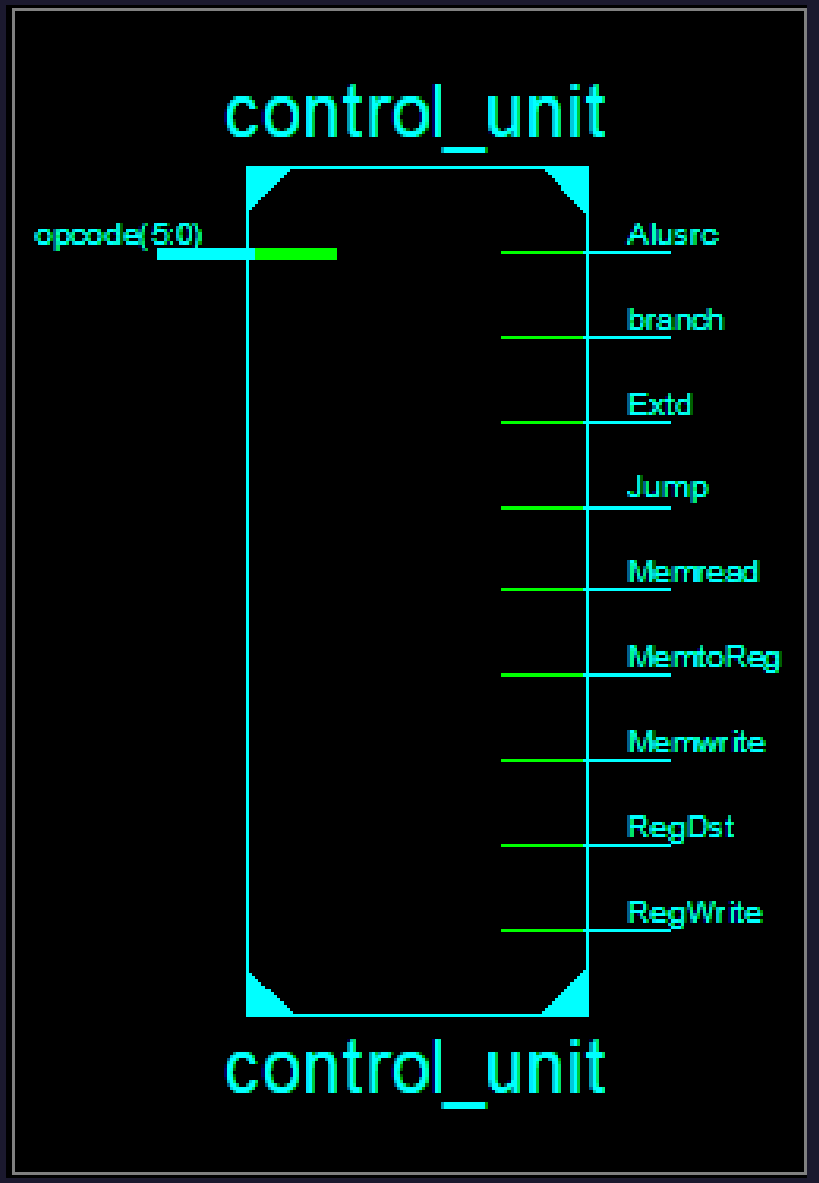
```
module extender_tb();
reg [15:0] in;
wire [31:0] out;
reg Extd;
integer i;

extender ex1(in,out,Extd);

initial begin
    for(i=0;i<99;i=i+1) begin
        in=$random;
        Extd=$random;
        #2;
    end
    #2 $stop;
end
initial begin
    $monitor("in=%b Extd=%b out=%b",in,Extd,out);
end

endmodule
```

Control Unit (CU)

controlunit.v

```v
module control_unit (opcode,RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch);

input [5:0] opcode;
output reg RegDst;
output reg RegWrite;
output reg Extd;
output reg Alusrc;
output reg Memread;
output reg Memwrite;
output reg MemtoReg;
output reg Jump;
output reg branch;

//controal signals decoding (control signal table)
//here we write control signal table but notice that many don't care will replaced by 0
//to ease simulation and avoid red signals which is confusing
//at right is the most correct format

always @(*) begin
    if(opcode==0)//Rtype
    {RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch}=9'b110000000;//9'b11x000000
    else if(opcode=='h8) //addi
    {RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch}=9'b011100000;//9'b011100000
    else if(opcode=='h23)//lw
    {RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch}=9'b011110100;//9'b011110100
    else if(opcode=='h2b)//sw
    {RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch}=9'b001101000;//9'bx01101x00
    else if(opcode=='h4)//beq
    {RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch}=9'b001000001;//9'bx01000x00
    else if(opcode=='h2) //jump
    {RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch}=9'b000000010;//9'bxxxx00010
    else //work R-type
    {RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch}=9'b110000000;//9'b11x000000
end
endmodule
```

Line 1, Column 1



Main Control Signals

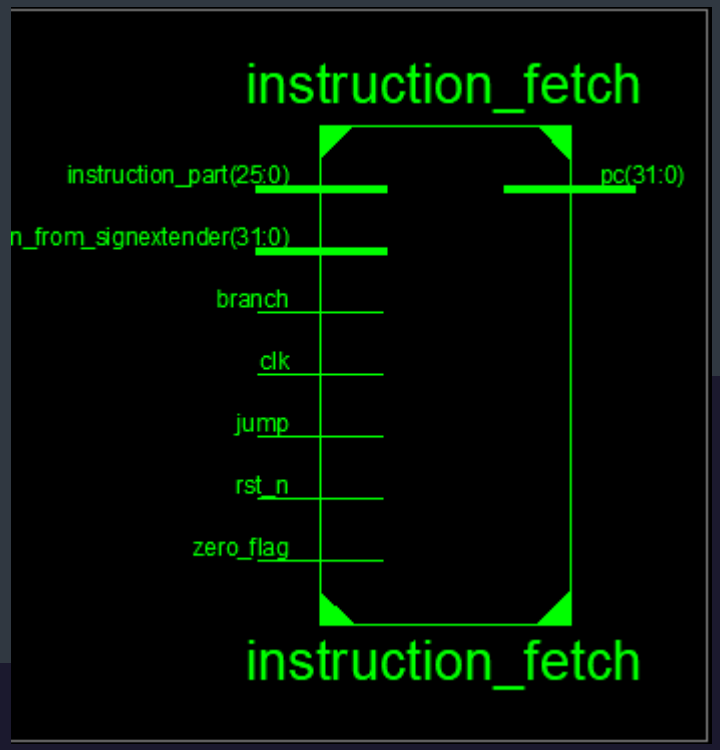
Operation	RegDst	RegWrite	Extd	ALUSrc	MemRead	MemWrite	MemtoReg
R-type	1	1	X	0	0	0	0
ADDI	0	1	1	1	0	0	0
SLTI	0	1	1	1	0	0	0
ANDI	0	1	0	1	0	0	0
ORI	0	1	0	1	0	0	0
XORI	0	1	0	1	0	0	0
LW	0	1	1	1	1	0	1
SW	X	0	1	1	0	1	X
BEQ	X	0	1	0	0	0	X
BNE	X	0	1	0	0	0	X

Two extra bits are added for the branch and jump controls

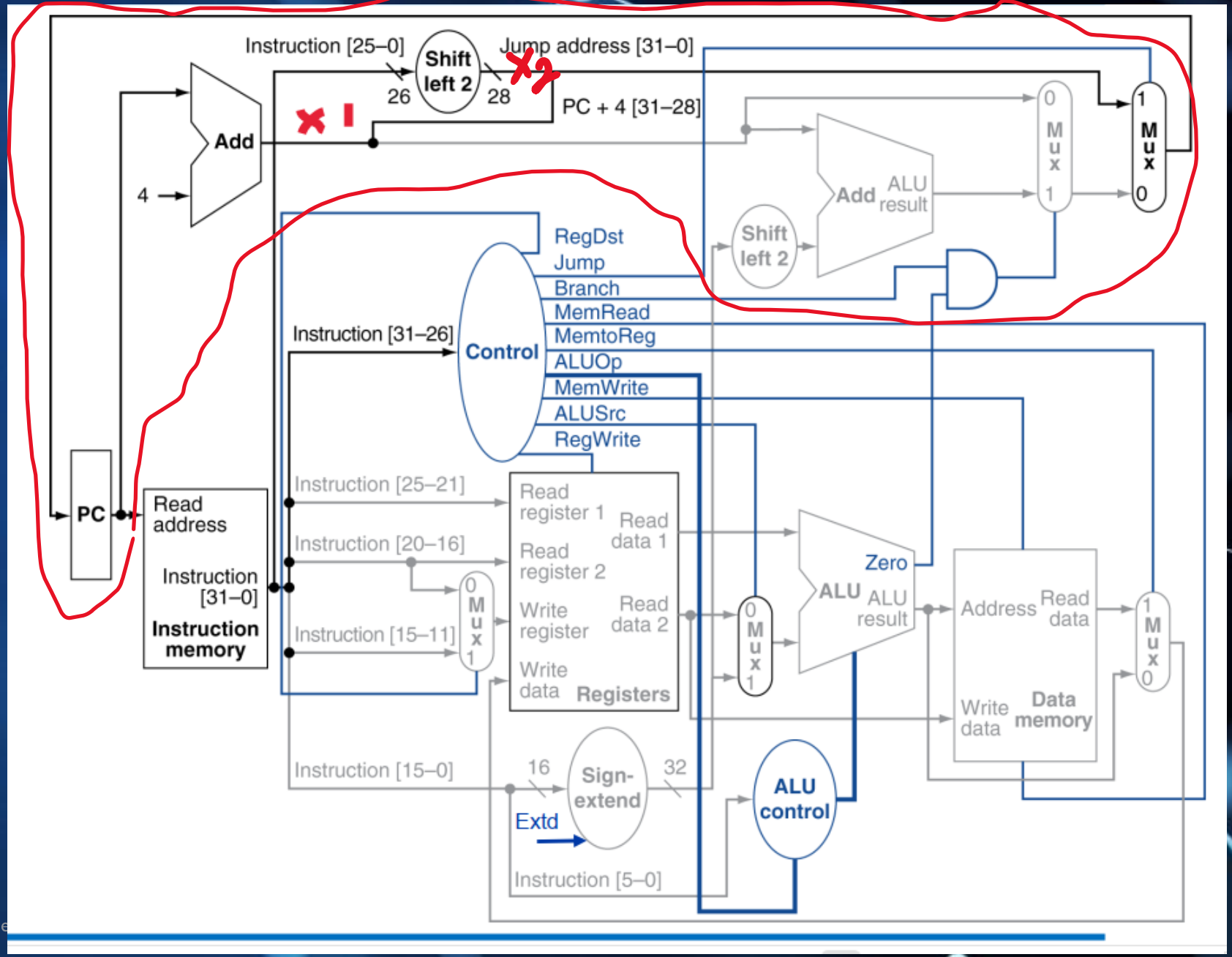
Instruction fetch (IF)

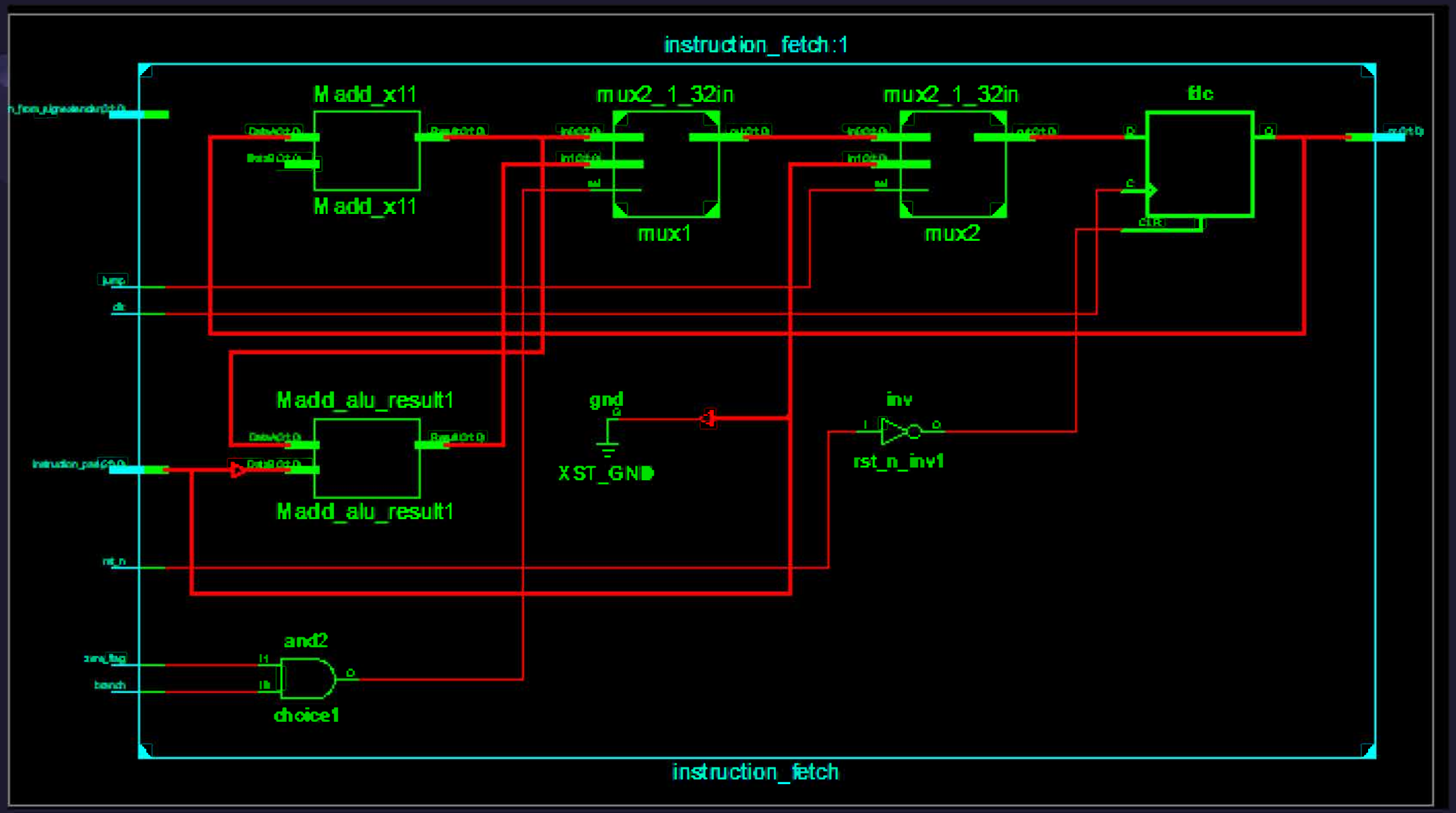
```
1  module instruction_fetch(clk,pc,instruction_part,in_from_signextender,jump,branch,zero_flag,rst_n);
2
3  input clk;
4  input branch,zero_flag,jump;//control signals
5  input [31:0] in_from_signextender;
6  input rst_n;
7  input [25:0] instruction_part;
8  output [31:0] pc;
9
10 //internal wires
11 reg [31:0]x1;//cs+4
12 reg [27:0]x2;//28 bit
13 reg [31:0] jump_address;
14 wire [31:0] jump_address2;
15 reg [31:0] alu_result;
16 wire choice;
17 and (choice,branch,zero_flag);
18 wire [31:0] out_mux1;
19
20 // we will design fetch as moore FSM
21 reg [31:0] cs ;
22 wire [31:0]ns;
23 // next state logic
24 always @(*) begin
25     x1=cs+4;
26     x2=instruction_part[25:0]<<2;
27     jump_address={x1[31:28],x2};
28     alu_result=x1+(in_from_signextender<<2);
29 end
```

```
30 assign jump_address2=jump_address;
31 mux2_1_32in mux1(.in0(x1),.in1(alu_result),.sel(choice),.out(out_mux1));
32 mux2_1_32in mux2(.in0(out_mux1),.in1(jump_address2),.sel(jump),.out(ns));
33
34 //state memory
35 always @(posedge clk or negedge rst_n) begin
36     if(!rst_n)
37     cs<=0;
38     else
39     cs<=ns;
40 end
41 //output logic
42 assign pc=cs;
43 endmodule
```



Schematic of SC MIPS




instruction_fetch:1
Madd_x11
Madd_x11
mux2_1_32in
mux1
mux2_1_32in
mux2
dc
jump
clk
Madd_alu_result1
Madd_alu_result1
gnd
XST_GND
inv
rst_n_inv1
rst_n
and2
branch
choice1
instruction_fetch

Top level

```
module mips1(clk,rst_n);

input clk;
input rst_n;//to reset all

//control signal declaration
wire RegDst;
wire RegWrite;
wire Extd;
wire Alusrc;
wire Memread;
wire Memwrite;
wire MemtoReg;
wire Jump;
wire branch;

//internal wires of fetch and instruction memory
wire [31:0] readaddr;
wire [31:0] instruction;

//internal wire of registerfiles
wire [4:0] Readreg1,Readreg2,WriteReg;
wire [31:0] WriteDataregsf;

//wire RegWrite;
wire [31:0] ReadData1,ReadData2;

//internal wire of alu
wire [5:0]opcode,funct;
wire [31:0] a,b,aluresult;
wire zero_flag;

//internal wires of signextender
wire [15:0] in_signextendr;
wire [31:0] out_signextender;
```

Top level(cont.)

```
35  wire [31:0] out_signextender;
36
37  //internal wire of data memory
38  wire [31:0] ReadData;
39
40
41  //interconnections we could write it direct in instantiations but this is better for reading
42
43  assign opcode=instruction[31:26];
44  assign funct=instruction[5:0];
45  assign Readreg1=instruction[25:21];//rs
46  assign Readreg2=instruction[20:16];//rt
47  assign a=ReadData1;
48  assign in_signextendr=instruction[15:0];
49
50
51  //_____________________________blocks instantiation_________
52
53  control_unit controll(opcode,RegDst,RegWrite,Extd,Alusrc,Memread,Memwrite,MemtoReg,Jump,branch);
54
55  data_memory datamemory(.Addr(aluresult),.write_data(ReadData2),
56      .data_out(ReadData),.mem_read(Memread),.mem_write(Memwrite),.clk(clk));
57
58  mux2_1_32in mux_datamemory(.in0(aluresult),.in1(ReadData),.sel(MemtoReg),.out(WriteDataregsf));
59
60  mux2_1_6in mux_regs(.in0(instruction[20:16]),.in1(instruction[15:11]),.sel(RegDst),.out(WriteReg));
61
62  mux2_1_32in mux_alu(.in0(ReadData2),.in1(out_signextender),.sel(Alusrc),.out(b));
63
64  extender extender1(in_signextendr,out_signextender,Extd);
65
66  instruction_fetch fetch(.clk(clk),.pc(readaddr),.instruction_part(instruction[25:0]),
67      .in_from_signextender(out_signextender),.jump(Jump),.branch(branch)
68      ,.zero_flag(zero_flag),.rst_n(rst_n));
69
68      ,.zero_flag(zero_flag),.rst_n(rst_n));
69
70  instruction_mem isntructionmemory(readaddr,instruction);
71
72  registerfile regfile(Readreg1,Readreg2,WriteReg,WriteDataregsf, RegWrite,ReadData1,ReadData2,clk);
73
74  alu alublock(opcode,funct,a,b,aluresult,zero_flag);
75
76
77  endmodule
```

Top level: testbench

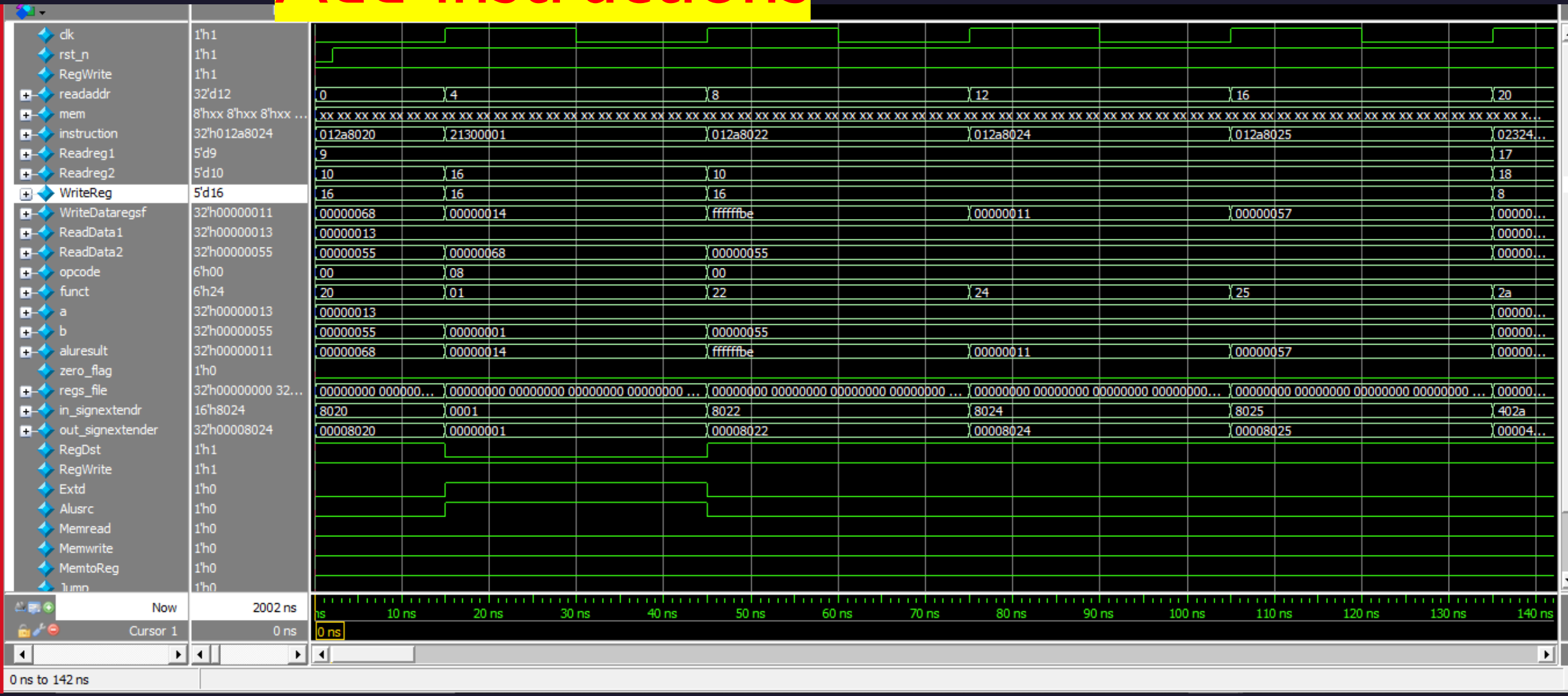
```
1  module mips1_tb();
2  reg clk;
3  reg rst_n;
4  mips1 mipso1(clk,rst_n);
5
6
7  initial begin
8      clk=0;
9      forever
10     #15 clk=~clk;
11 end
12
13 initial begin
14     rst_n=0;
15     $readmemh("mem.dat",mipso1.isntructionmemory.mem);//instructionmemory
16     $readmemh("reg.dat",mipso1.regfile.regs_file); //registerfiles
17     $readmemh("mem2.dat",mipso1.datamemory.mem); //data memory
18     #2 rst_n=1;
19     #2000 $stop;
20 end
21
22 endmodule
```

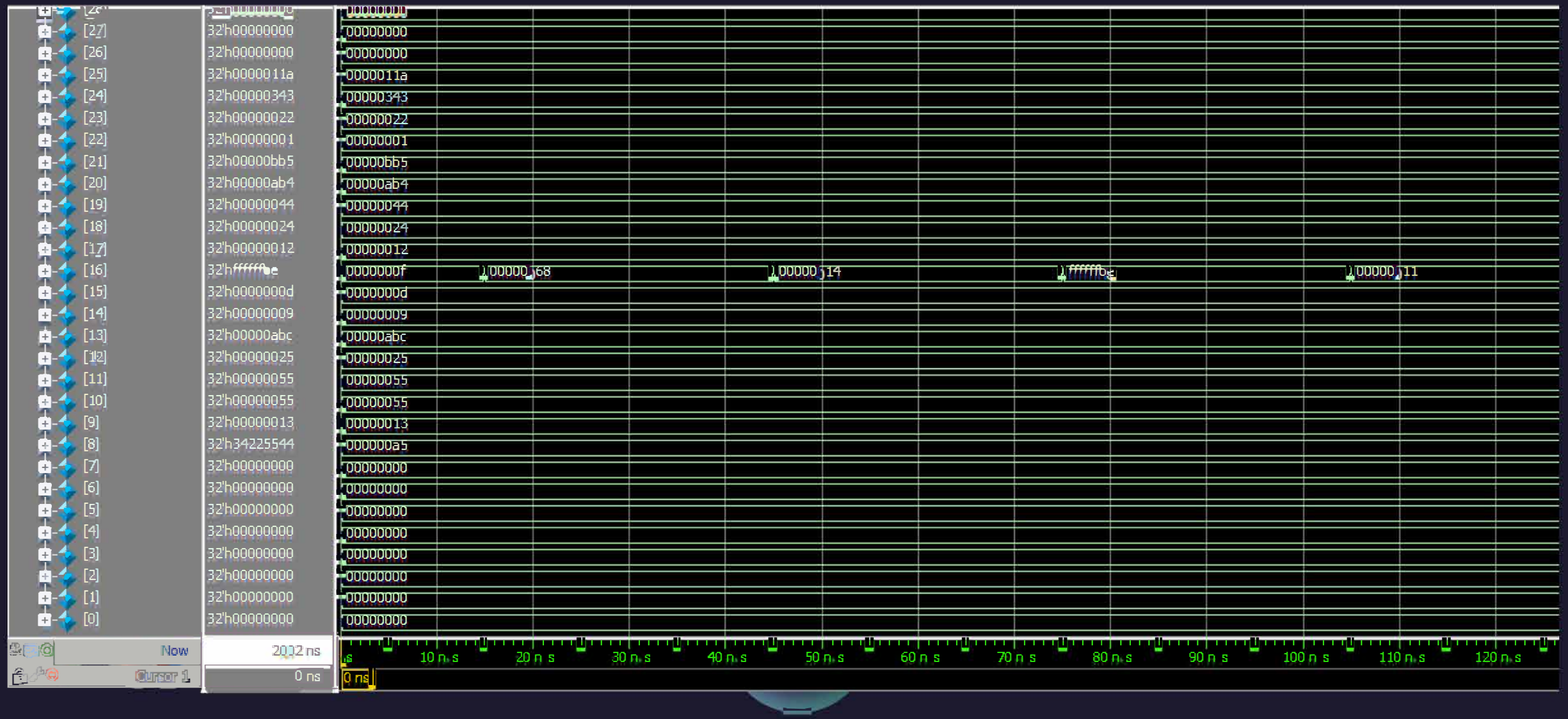
Simulation instructions

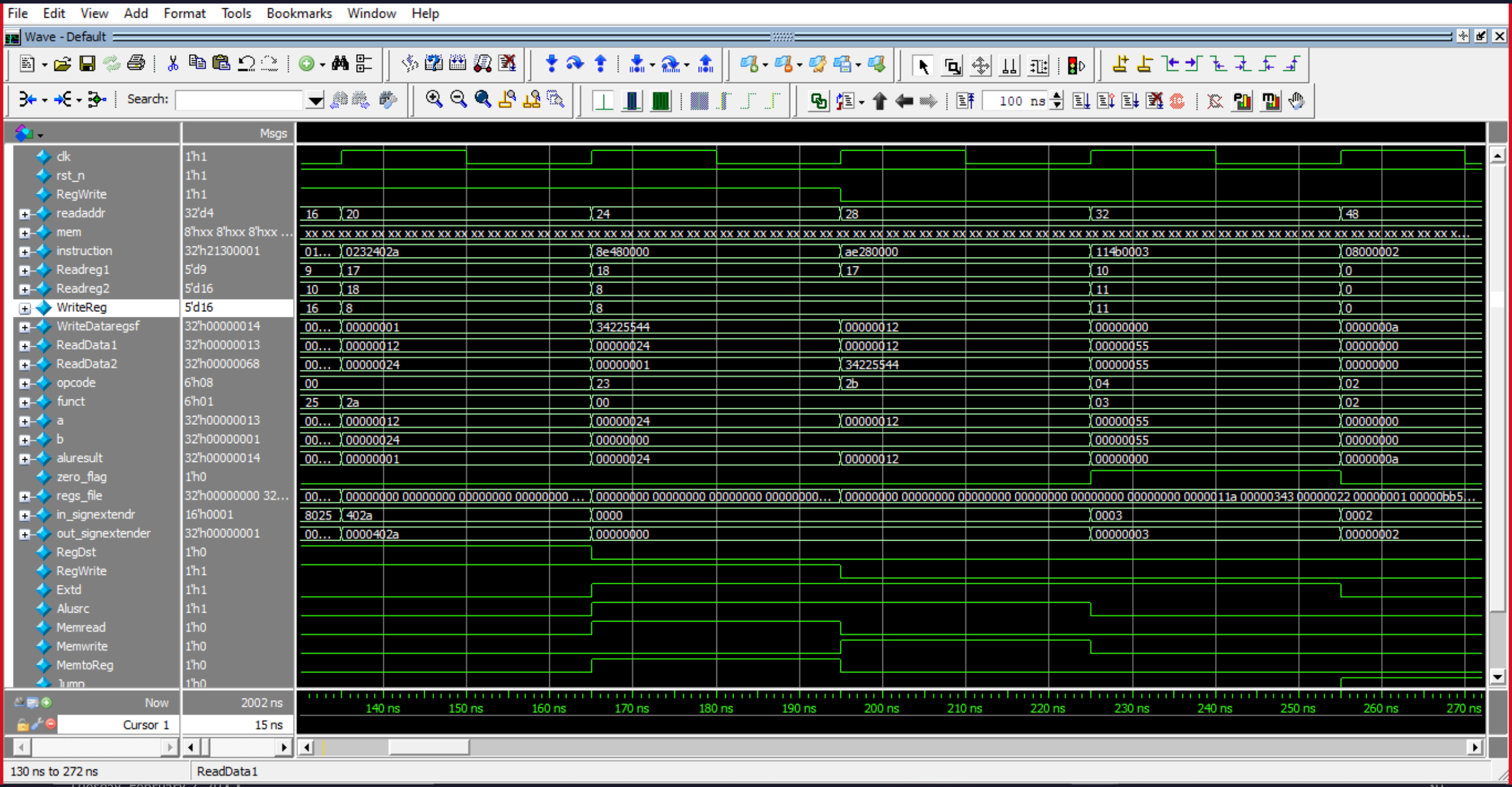
0_ add s0 t1 t2 //ok	**0x012A8020**
4_addi s0 t1 0x00000001 //ok	0x21300001
8_sub s0 t1 t2 //ok	0x012A8022
12_and s0 t1 t2 //ok	0x012A8024
16_or s0 t1 t2 //ok	0x012A8025
20_slt t0 s1 s2 //ok	0x0232402A
24_lw t0 0x00000000(s2) //ok	0x8E480000
28_sw t0 0x00000000(s1) // ok	0xAE280000
32_beq t2 , t3 , 0x00000003 //ok	0x114B0003
36_Add s3,t1,t2	0x012A9820
40_Or s5 ,t0 ,t6	0x010EA825
44_And s4,t0,t6	0x010EA024
48_j 0x00000002 // ok	0x08000002

Simulation

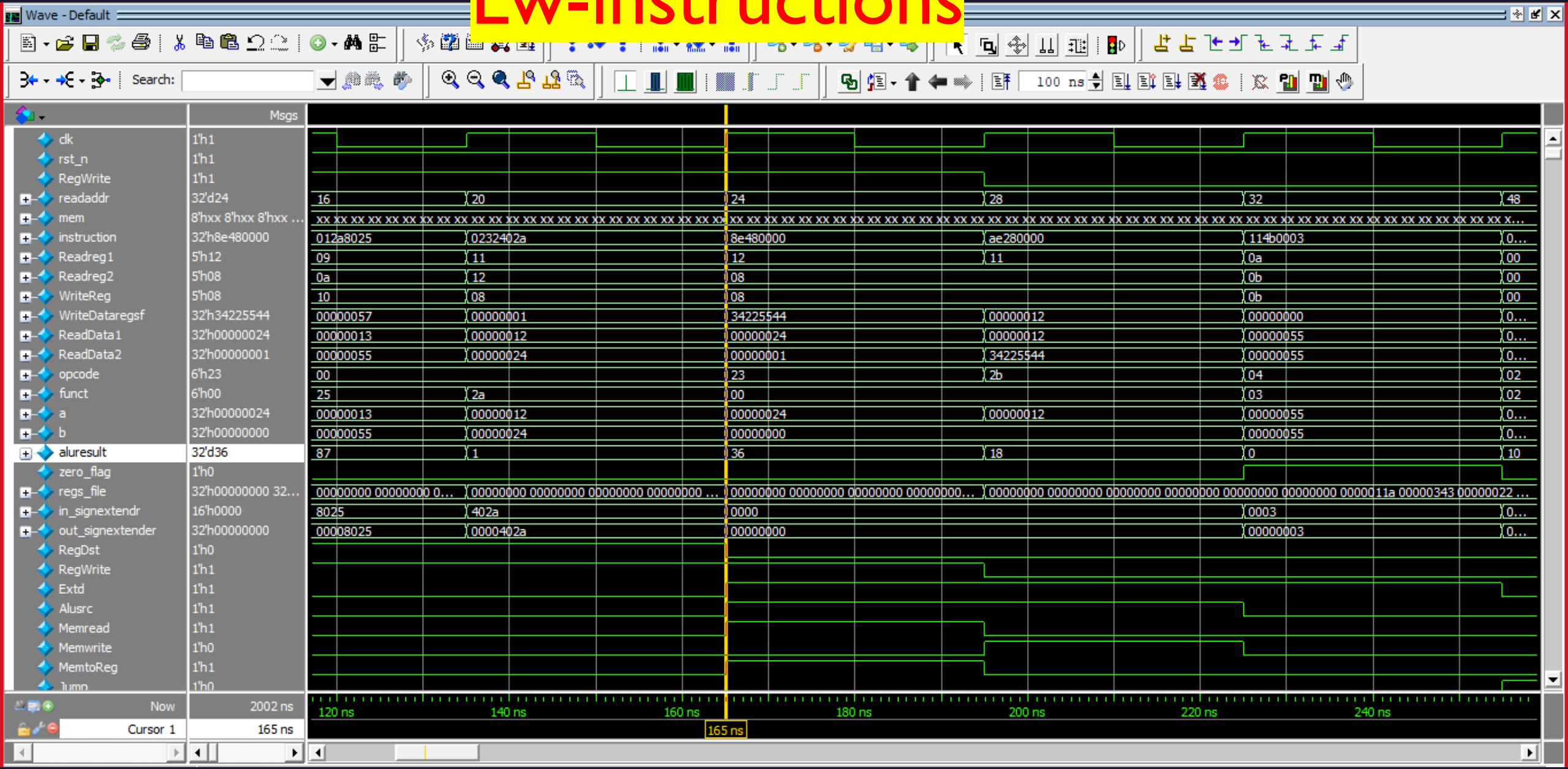
Add-instructions

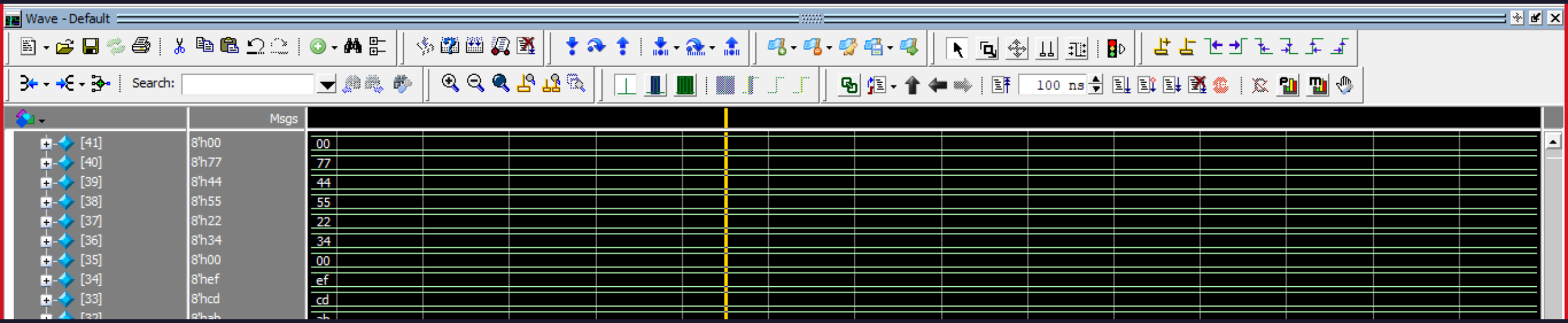


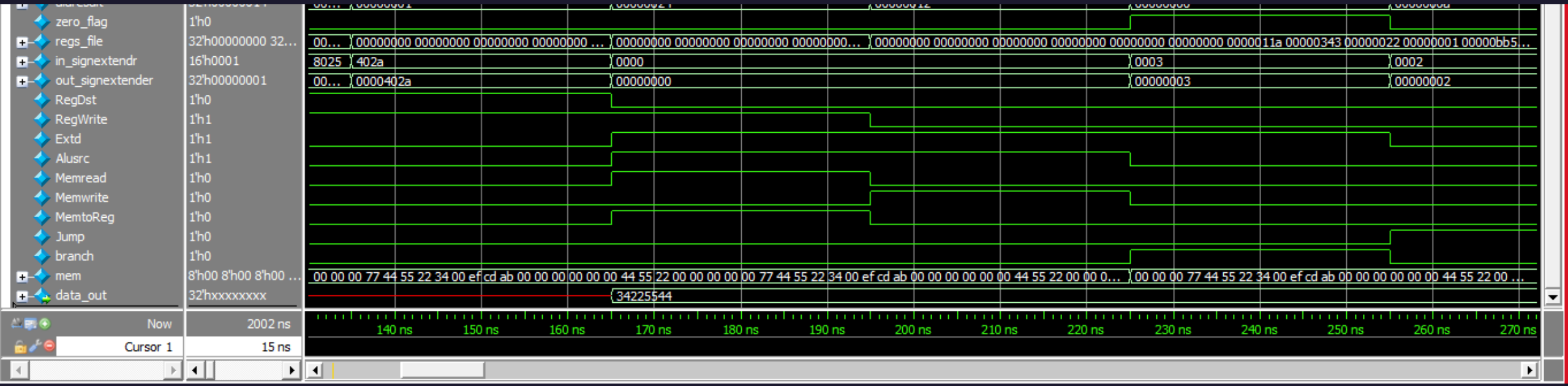

[27] 32'h00000000
[26] 32'h00000000
[25] 32'h0000011a
[24] 32'h00000343
[23] 32'h00000022
[22] 32'h00000001
[21] 32'h00000bb5
[20] 32'h00000ab4
[19] 32'h00000044
[18] 32'h00000024
[17] 32'h00000012
[16] 32'hffffffbe
[15] 32'h0000000d
[14] 32'h00000009
[13] 32'h00000abc
[12] 32'h00000025
[11] 32'h00000055
[10] 32'h00000055
[9] 32'h00000013
[8] 32'h34225544
[7] 32'h00000000
[6] 32'h00000000
[5] 32'h00000000
[4] 32'h00000000
[3] 32'h00000000
[2] 32'h00000000
[1] 32'h00000000
[0] 32'h00000000
00000000
00000000
00000000
0000011a
00000343
00000022
00000001
00000bb5
00000ab4
00000044
00000024
00000012
0000000f
0000000d
00000009
00000abc
00000025
00000055
00000055
00000013
000000a5
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000068
00000014
00000011
Now
2002 ns
Cursor 1
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns

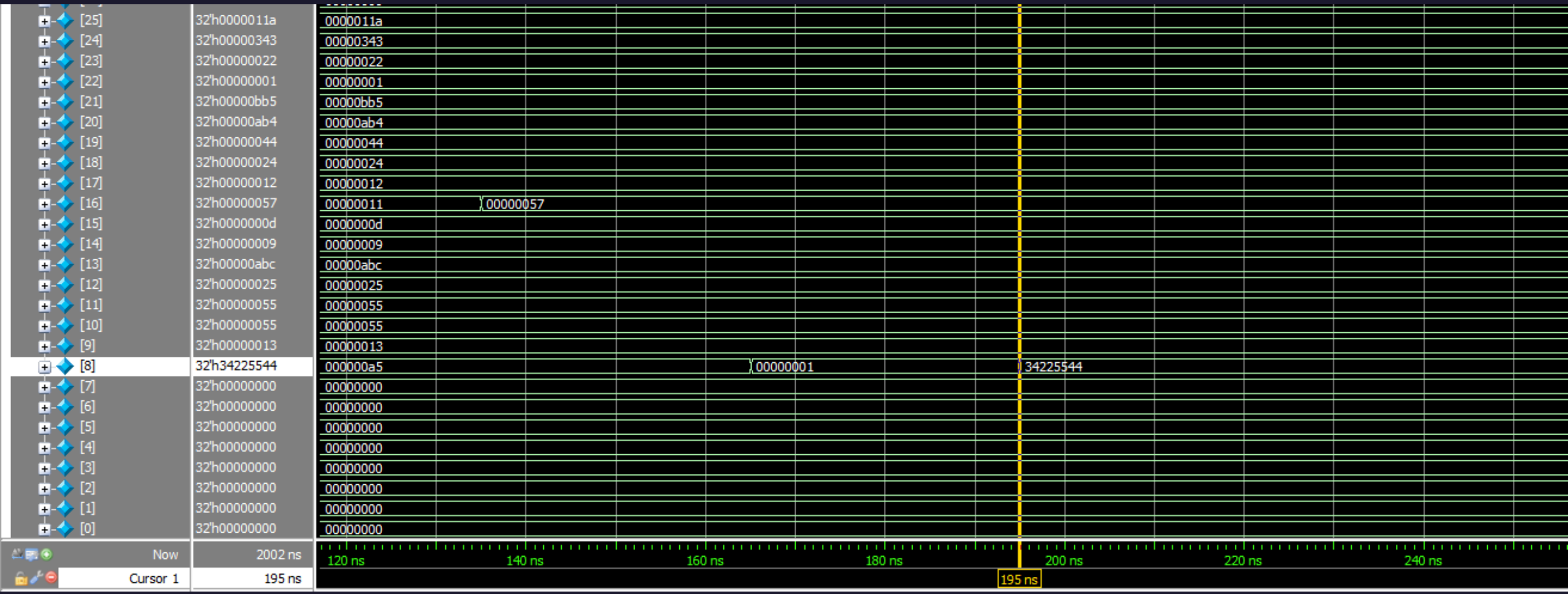

File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
100 ns
Msgs
clk 1'h1
rst_n 1'h1
RegWrite 1'h1
readaddr 32'd4
mem 8'hxx 8'hxx 8'hxx ...
instruction 32'h21300001
Readreg1 5'd9
Readreg2 5'd16
WriteReg 5'd16
WriteDataregsf 32'h00000014
ReadData1 32'h00000013
ReadData2 32'h00000068
opcode 6'h08
funct 6'h01
a 32'h00000013
b 32'h00000001
aluresult 32'h00000014
zero_flag 1'h0
regs_file 32'h00000000 32...
in_signextendr 16'h0001
out_signextender 32'h00000001
RegDst 1'h0
RegWrite 1'h1
Extd 1'h1
Alusrc 1'h1
Memread 1'h0
Memwrite 1'h0
MemtoReg 1'h0
Jump 1'h0
Now 2002 ns
Cursor 1 15 ns
140 ns 150 ns 160 ns 170 ns 180 ns 190 ns 200 ns 210 ns 220 ns 230 ns 240 ns 250 ns 260 ns 270 ns
130 ns to 272 ns
ReadData1

Lw-instructions

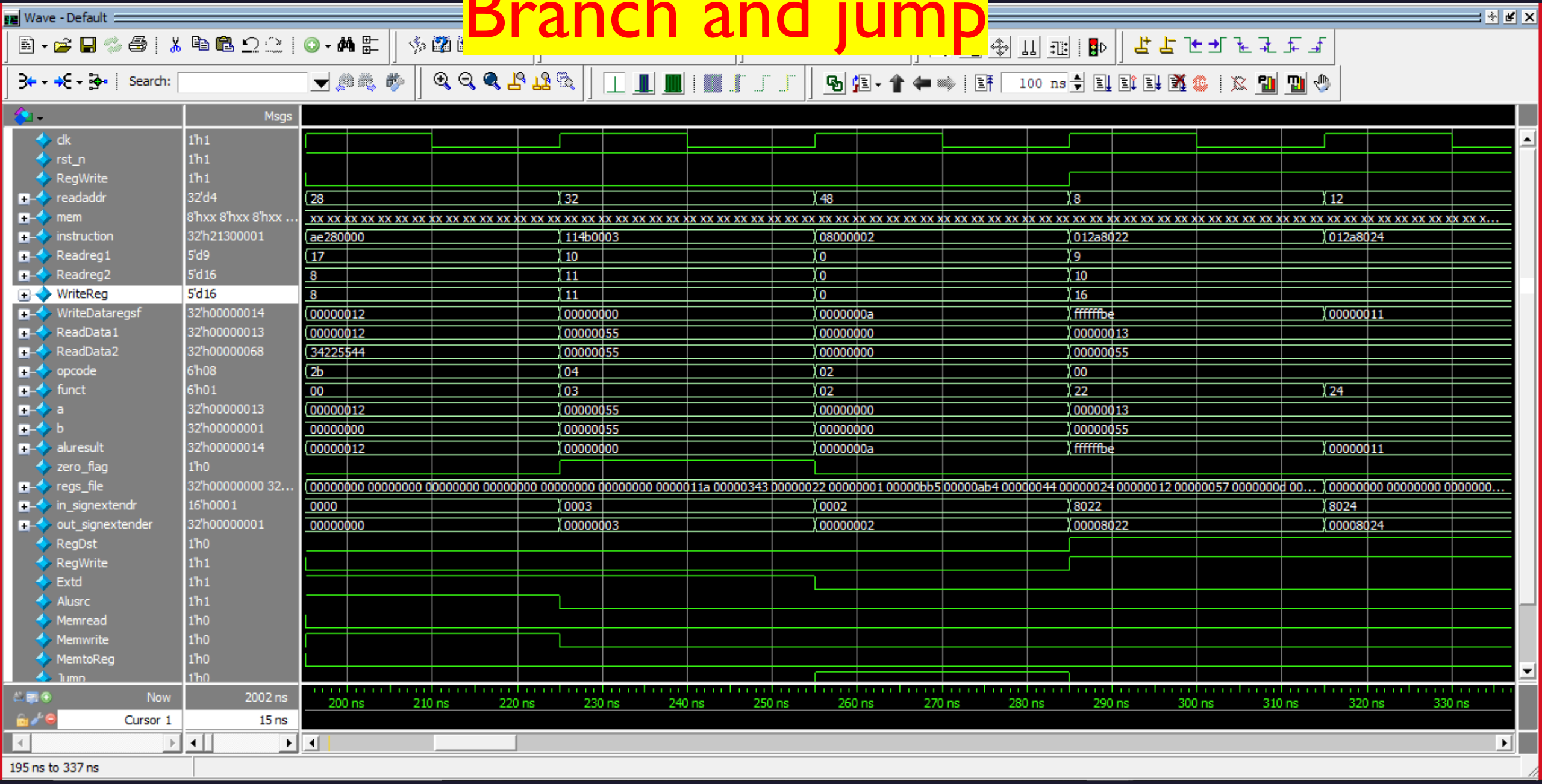


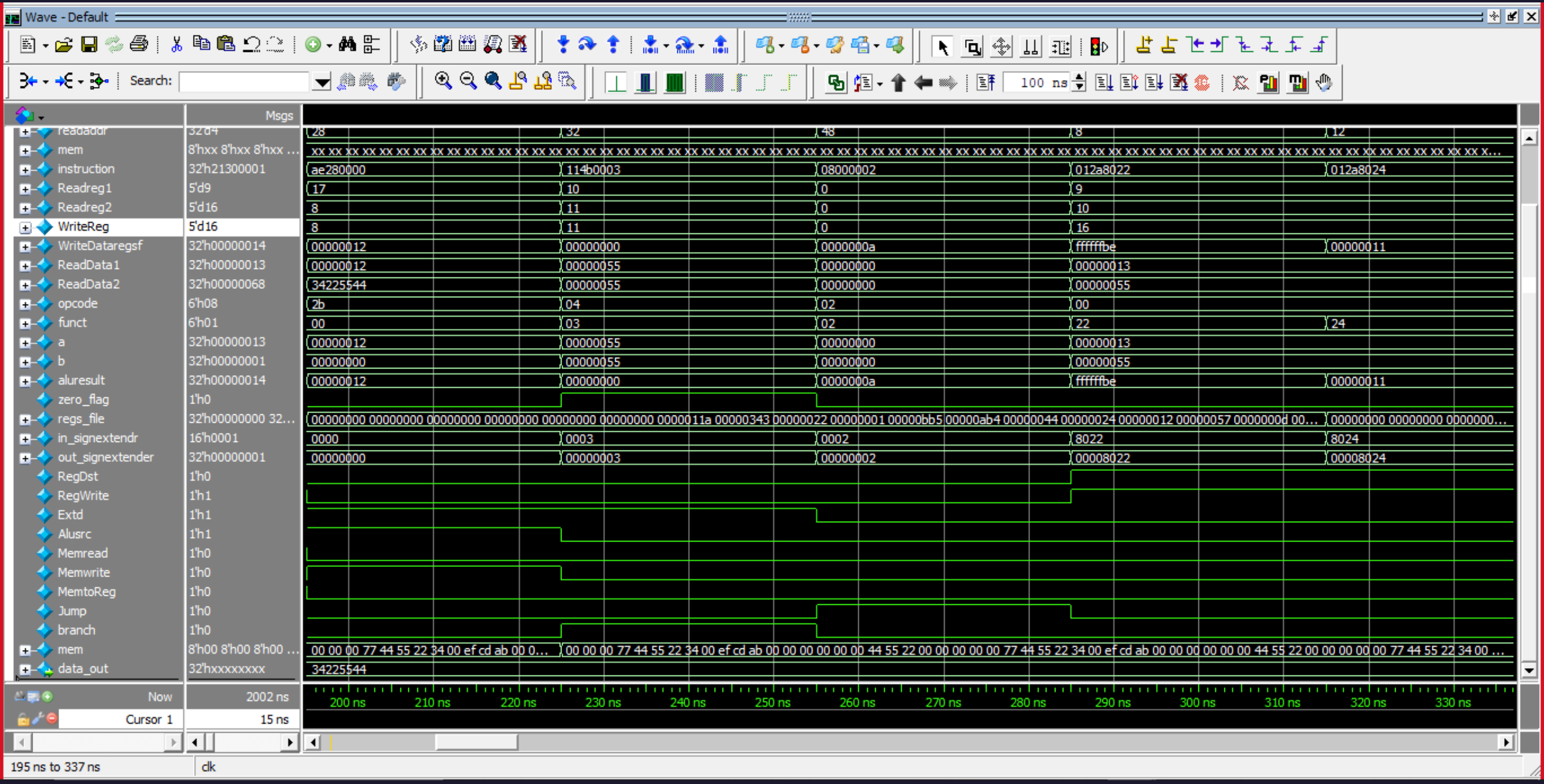

Wave - Default
Search:
100 ns
Msgs
[41] 8'h00 00
[40] 8'h77 77
[39] 8'h44 44
[38] 8'h55 55
[37] 8'h22 22
[36] 8'h34 34
[35] 8'h00 00
[34] 8'hef ef
[33] 8'hcd cd


zero_flag
1'h0
regs_file
32'h00000000 32...
in_signextendr
16'h0001
out_signextender
32'h00000001
RegDst
1'h0
RegWrite
1'h1
Extd
1'h1
Alusrc
1'h1
Memread
1'h0
Memwrite
1'h0
MemtoReg
1'h0
Jump
1'h0
branch
1'h0
mem
8'h00 8'h00 8'h00 ...
data_out
32'hxxxxxxxx
8025
402a
0000
0003
0002
0000402a
00000000
00000003
00000002
34225544
Now
2002 ns
Cursor 1
15 ns
140 ns
150 ns
160 ns
170 ns
180 ns
190 ns
200 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
270 ns


[25] 32'h0000011a
[24] 32'h00000343
[23] 32'h00000022
[22] 32'h00000001
[21] 32'h00000bb5
[20] 32'h00000ab4
[19] 32'h00000044
[18] 32'h00000024
[17] 32'h00000012
[16] 32'h00000057
[15] 32'h0000000d
[14] 32'h00000009
[13] 32'h00000abc
[12] 32'h00000025
[11] 32'h00000055
[10] 32'h00000055
[9] 32'h00000013
[8] 32'h34225544
[7] 32'h00000000
[6] 32'h00000000
[5] 32'h00000000
[4] 32'h00000000
[3] 32'h00000000
[2] 32'h00000000
[1] 32'h00000000
[0] 32'h00000000
0000011a
00000343
00000022
00000001
00000bb5
00000ab4
00000044
00000024
00000012
00000011
00000057
0000000d
00000009
00000abc
00000025
00000055
00000055
00000013
000000a5
00000001
34225544
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
Now
2002 ns
Cursor 1
195 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
195 ns

Branch and jump




Wave - Default
Search:
100 ns
Msgs
mem 8'hxx 8'hxx 8'hxx ...
instruction 32'h21300001
Readreg1 5'd9
Readreg2 5'd16
WriteReg 5'd16
WriteDataregsf 32'h00000014
ReadData1 32'h00000013
ReadData2 32'h00000068
opcode 6'h08
funct 6'h01
a 32'h00000013
b 32'h00000001
aluresult 32'h00000014
zero_flag 1'h0
regs_file 32'h00000000 32...
in_signextendr 16'h0001
out_signextender 32'h00000001
RegDst 1'h0
RegWrite 1'h1
Extd 1'h1
Alusrc 1'h1
Memread 1'h0
Memwrite 1'h0
MemtoReg 1'h0
Jump 1'h0
branch 1'h0
mem 8'h00 8'h00 8'h00 ...
data_out 32'hxxxxxxxx
28
32
48
8
12
ae280000
114b0003
08000002
012a8022
012a8024
17
10
0
9
8
11
0
10
8
11
0
16
00000012
00000000
0000000a
fffffffbe
00000011
00000012
00000055
00000000
00000013
34225544
00000055
00000000
00000055
2b
04
02
00
00
03
02
22
24
00000012
00000055
00000000
00000013
00000000
00000055
00000000
00000055
00000012
00000000
0000000a
fffffffbe
00000011
0000
0003
0002
8022
8024
00000000
00000003
00000002
00008022
00008024
Now
2002 ns
Cursor 1
15 ns
200 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
270 ns
280 ns
290 ns
300 ns
310 ns
320 ns
330 ns
195 ns to 337 ns
clk

Simulation instructions

0_ADD $s0 $t1 $t2	**0x012A8020**
4_ADDI $s0 $t1 0x0001	0x21300001
8_SUB $s0 $t1 $t2	0x012A8022
12_AND $s0 $t1 $t2	0x012A8024
16_OR $s0 $t1 $t2	0x012A8025
20_SLT $t0 $s1 $s2	0x0232402A
24_LW $t0 0x0000 $s2	0x8E480000
28_SW $t0 0x0000 $s1	0xAE280000
32_BEQ $t0 $t1 0x0001	0x11090001
36_j 0x0000002	0x08000002
40_SUB $s0 $t1 $t2	0x012A8022

DO-File (single cycle)

```
vlib work
vlog alu.v datamemory.v instruction_fetch.v instruction_memory.v mips_tb.v mips1.v mux2_1_6in.v mux2_1_32in.v registerfile.v signextender.v controlunit.v
vsim -voptargs=+acc work.mips1_tb
add wave *
add wave -position insertpoint  \
sim:/mips1_tb/mipso1/RegWrite \
sim:/mips1_tb/mipso1/readaddr \
sim:/mips1_tb/mipso1/isntructionmemory/mem \
sim:/mips1_tb/mipso1/instruction \
sim:/mips1_tb/mipso1/Readreg1 \
sim:/mips1_tb/mipso1/Readreg2 \
sim:/mips1_tb/mipso1/WriteReg \
sim:/mips1_tb/mipso1/WriteDataregsf \
sim:/mips1_tb/mipso1/ReadData1 \
sim:/mips1_tb/mipso1/ReadData2 \
sim:/mips1_tb/mipso1/opcode \
sim:/mips1_tb/mipso1/funct \
sim:/mips1_tb/mipso1/a \
sim:/mips1_tb/mipso1/b \
sim:/mips1_tb/mipso1/aluresult \
sim:/mips1_tb/mipso1/zero_flag \
sim:/mips1_tb/mipso1/regfile/regs_file \
sim:/mips1_tb/mipso1/in_signextendr \
sim:/mips1_tb/mipso1/out_signextender \
sim:/mips1_tb/mipso1/RegDst \
sim:/mips1_tb/mipso1/RegWrite \
sim:/mips1_tb/mipso1/Extd \
sim:/mips1_tb/mipso1/Alusrc \
sim:/mips1_tb/mipso1/Memread \
sim:/mips1_tb/mipso1/Memwrite \
sim:/mips1_tb/mipso1/MemtoReg \
sim:/mips1_tb/mipso1/Jump \
sim:/mips1_tb/mipso1/branch \
sim:/mips1_tb/mipso1/datamemory/mem \
sim:/mips1_tb/mipso1/datamemory/data_out
run -all
#quit -sim
```

File Edit Format View Help

```
vlib work
vlog -coveropt 3 +cover +acc alu.v datamemory.v instruction_fetch.v instruction_memory.v mips_tb.v mips1.v mux2_1_6in.v mux2_1_32in.v registerfile.v signextender.v cont
vsim -coverage -vopt work.mips1_tb -c -do "coverage save -onexit -directive -codeAll ReportCoverge.ucdb; run -all"
vcover report -html ReportCoverge.ucdb
#quit -sim
```

Instances | Design Units

Search...

mips1_tb (83.44%)

Questa Coverage Report Summary

Instance Coverage Summary (83.44%)

Coverage Type	Bins	Hits
Branches	39	37
Conditions	21	18
Expressions	1	1
Statements	58	57
Toggles	1662	638

Design Units Coverage Summary (84.72%)

Coverage Type	Bins	Hits
Branches	33	31
Conditions	21	18
Expressions	1	1
Statements	55	54
Toggles	2574	1179

Report generated by Questa (ver: 2021.1) on (Friday 30 December 2022 21:52:44) with command line:
vcover report -html ReportCoverge.ucdb

references

Mips code converter : https://www.eg.bucknell.edu/~csci320/mips_web/

Mips green sheet.

Ahmed Saeed, PhD, SMIEEE lectures

Thank You