

ITI DIGITAL VERIFICATION ENGINEERING PROGRAM

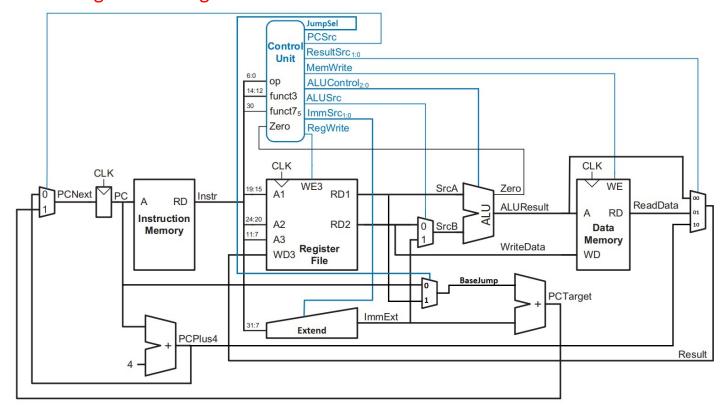
ASSIGNMENT 3 VERILOG RISC-V SINGLE CYCLE CPU

SUBMITTED TO ENG MOHAMED ZAYTOUN

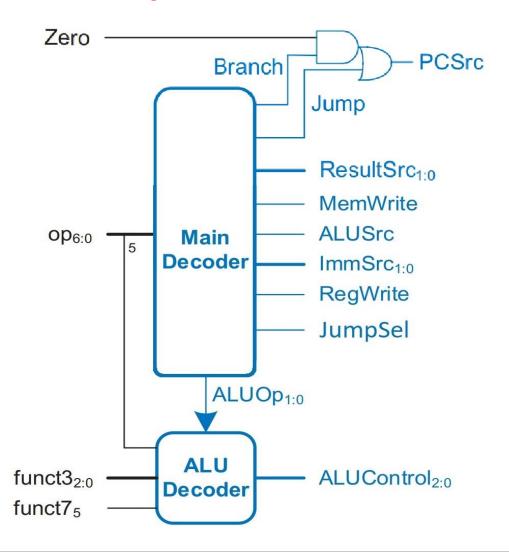
BY
ABDURRAHMAN SHERIF ABDULHAMID

I. Block diagram:

Design Block Diagram:



• Control Unit Block Diagram



II. Design codes:

- Universal blocks
 - Adder

```
1
2 module adder (
3     input [31:0] a, b,
4     output [31:0] y
5 );
6     assign y = a + b;
7 endmodule
```

0 2-1 MUX

```
1 module mux2 #(parameter WIDTH = 8) (
2    input [WIDTH-1:0] d0, d1,
3    input s,
4    output [WIDTH-1:0] y
5 );
6    assign y = s ? d1 : d0;
7 endmodule
```

0 3-1 MUX

```
1 module mux3 #(parameter WIDTH = 8) (
2    input [WIDTH-1:0] d0, d1, d2,
3    input [1:0] s,
4    output [WIDTH-1:0] y
5 );
6    assign y = s[1] ? d2 : (s[0] ? d1 : d0);
7 endmodule
```

O FLOP REGISTER

```
module flopr #(parameter WIDTH = 8) (
input clk, reset,
input [WIDTH-1:0] d,
output reg [WIDTH-1:0] q
);
always @(posedge clk or posedge reset) begin
if(reset) begin
q <= 0;
end else begin
q <= d;
end
end
end
end
end
end</pre>
```

Control Unit:

Main Decoder

```
module maindec (
    input [6:0] op,
    output [1:0] ResultSrc,
    output MemWrite,
    output Branch, ALUSrc,
    output RegWrite, Jump,
    output [1:0] ImmSrc,
    output [1:0] ALUOp,
    output JumpSel
    reg [11:0] controls;
    assign {RegWrite, ImmSrc, ALUSrc, MemWrite,
         ResultSrc, Branch, ALUOp, Jump, JumpSel} = controls;
    always@(*)begin case(op)
// RegWrite ImmSrc ALUSrc MemWrite ResultSrc Branch ALUOp Jump JumpSel
              7'b0000011: controls = 12'b1_00_1_0_01_0_00_0; // lw
             7'b0100011: controls = 12'b0 01 1 1 00 0 00
                                                                0 0; // sw
             7'b0110011: controls = 12'b1_xx_0_0_00_0_10_0; // R-type
             7'b1100011: controls = 12'b0 10 0 0 00 1 01
                                                                0 0; // beq/bne*
             7'b0010011: controls = 12'b1_00_1_0_00_0_10_00; // I-type ALU
7'b1101111: controls = 12'b1_11_0_0_10_00_1_0; // jal
7'b1100111: controls = 12'b1_00_0_0_10_00_0_1; // jalr *
             default: controls = 11'bx_xx_x_x_xx_xx_x; // ???
         endcase
    end
endmodule
```

ALU Decoder

```
module aludec (
     input opb5,
input [2:0] funct3,
input funct7b5,
     input [1:0] ALUOp,
output reg [2:0] ALUControl
     wire RtypeSub;
     assign RtypeSub = funct7b5 & opb5; // TRUE for R-type subtract
     always@(*)
case(ALUOp)
                 2'b00: ALUControl = 3'b000; // addition
                 2'b01: ALUControl = 3'b001; // subtraction
                 default: case(funct3) // R-type or I-type ALU
                      3'b000: if (RtypeSub)
                            ALUControl = 3'b001; // sub
                            ALUControl = 3'b000; // add, addi
                      3'b010: ALUControl = 3'b101; // slt, slti
3'b110: ALUControl = 3'b011; // or, ori
3'b111: ALUControl = 3'b010; // and, andi
default: ALUControl = 3'bxxx; // ???
                 endcase
           endcase
     endmodule
```

Control Unit Wrapper

```
module controller (
input [6:0] op,
input [2:0] funct3,
input funct7b5,
input Zero,
output [1:0] ResultSrc,
output PCSrc, ALUSrc,
output RegWrite, Jump,
output [1:0] ImmSrc,
output [2:0] ALUControl,
output JumpSel

in wire [1:0] ALUOp;
wire Branch;
maindec md(op, ResultSrc, MemWrite, Branch,
ALUSrc, RegWrite, Jump, ImmSrc, ALUOp, JumpSel);
aludec ad(op[5], funct3, funct7b5, ALUOp, ALUControl);
assign PCSrc = (Branch & Zero & funct3[0]) | Jump | (Branch & !Zero & funct3[0]) | JumpSel;
endmodule
```

Data Path:

```
module datapath (
         input clk, reset,
         input [1:0] ResultSrc,
               PCSrc, ALUSrc,
                RegWrite,
         input
                [1:0] ImmSrc,
                [2:0] ALUControl,
         output Zero,
         output [31:0] PC,
                [31:0] Instr,
11
         output [31:0] ALUResult, WriteData,
12
         input [31:0] ReadData,
13
         input JumpSel
     );
15
         wire [31:0] PCNext, PCPlus4, PCTarget;
         wire [31:0] ImmExt;
         wire [31:0] SrcA, SrcB;
17
         wire [31:0] Result;
         wire [31:0] BaseJump;
     // next PC logic
         flopr #(32) pcreg(clk, reset, PCNext, PC);
         adder pcadd4(PC, 32'd4, PCPlus4);
22
         mux2 #(32) jumpbranch (PC,SrcA , JumpSel, BaseJump);
23
         adder pcaddbranch(BaseJump, ImmExt, PCTarget);
         mux2 #(32) pcmux(PCPlus4, PCTarget, PCSrc, PCNext);
     // register file logic
         regfile rf(clk, RegWrite, Instr[19:15], Instr[24:20],
             Instr[11:7], Result, SrcA, WriteData);
         extend ext(Instr[31:7], ImmSrc, ImmExt);
29
     // ALU logic
         mux2 #(32) srcbmux(WriteData, ImmExt, ALUSrc, SrcB);
         alu alu(SrcA, SrcB, ALUControl, Zero, ALUResult);
         mux3 #(32) resultmux(ALUResult, ReadData, PCPlus4,
             ResultSrc, Result);
     endmodule
```

• ALU:

```
module alu (
    input [31:0] SrcA,
    input [31:0] SrcB,
    input [2:0] ALUcontrol,
    output zero,
   output reg [31:0] ALUResult
);
    always @(*) begin
        case (ALUcontrol)
           3'b000: ALUResult = SrcA + SrcB;
                                                    // Addition
            3'b001: ALUResult = SrcA - SrcB ;
                                                    // Subtraction
            3'b010: ALUResult = SrcA & SrcB;
                                                   // Bitwise AND
            3'b011: ALUResult = SrcA | SrcB;
            3'b101: ALUResult = (SrcA < SrcB) ? 1 : 0; // Set if less than (SLT)
            default: ALUResult = 32'bx;
                                              // Default: Zero output
        endcase
    assign zero = (ALUResult == 32'b0) ? 1'b1 : 1'b0;
endmodule
```

Extend block:

```
module extend (
         input [31:7] instr,
         input [1:0] immsrc,
        output reg [31:0] immext
     );
     always @(*) begin
         case(immsrc)
             // I-type
             2'b00: immext = {{20{instr[31]}}, instr[31:20]};
             // S-type (stores)
             2'b01: immext = {{20{instr[31]}}, instr[31:25], instr[11:7]};
             // B-type (branches)
             2'b10: immext = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
             // J-type (jal)
             2'b11: immext = {{12{instr[31]}}}, instr[19:12], instr[20], instr[30:21], 1'b0};
             default: immext = 32'bx; // undefined
         endcase
     end
    endmodule
22
```

Data Memory (RAM):

```
1  module dmem (
2    input clk, we,
3    input [31:0] a, wd,
4    output [31:0] rd
5  );
6    reg [31:0] RAM [255:0];
7    assign rd = RAM[a[7:0]]; // word aligned
9    always @(posedge clk) begin
11    if (we) RAM[a[7:0]] <= wd;
12    end
13    endmodule</pre>
```

Instruction Memory (ROM):

```
1  module imem (
2    input [31:0] a,
3    output [31:0] rd
4  );
5    reg [31:0] ROM[255:0]; // 1KB memory (256 words)
6    initial
8    $readmemh("riscvtest.txt", ROM);
9    assign rd = ROM[a[9:2]]; // word accessible
11  endmodule
12
```

Register File:

```
module regfile (
         input clk,
         input we3,
         input [4:0] a1, a2, a3,
         input [31:0] wd3,
         output [31:0] rd1, rd2
     );
         reg [31:0] rf [0:31]; // 32 registers
11
         // Three-ported register file
12
         // Read two ports combinationaly (A1/RD1, A2/RD2)
         // Write third port on rising edge of clock (A3/WD3/WE3)
         // Register 0 hardwired to 0
15
         always @(posedge clk) begin
             if (we3)
                 rf[a3] \leftarrow wd3;
         end
         assign rd1 = (a1 != 0) ? rf[a1] : 32'h0;
         assign rd2 = (a2 != 0) ? rf[a2] : 32'h0;
21
     endmodule
```

Single cycle RISC-V Processor:

```
module riscvsingle (
         input clk, reset,
        output [31:0] PC,
        input [31:0] Instr,
        output MemWrite,
        output [31:0] ALUResult, WriteData,
        input [31:0] ReadData
    );
        wire ALUSrc, RegWrite, Jump, Zero;
        wire [1:0] ResultSrc, ImmSrc;
        wire [2:0] ALUControl;
11
12
        wire JumpSel;
13
        controller c(Instr[6:0], Instr[14:12], Instr[30], Zero,
             ResultSrc, MemWrite, PCSrc,
15
             ALUSrc, RegWrite, Jump,
             ImmSrc, ALUControl, JumpSel);
17
         datapath dp(clk, reset, ResultSrc, PCSrc,
             ALUSrc, RegWrite,
20
             ImmSrc, ALUControl,
21
             Zero, PC, Instr,
             ALUResult, WriteData, ReadData, JumpSel);
    endmodule
```

Top Module Including Memories (Data, Instruction):

```
module top (
         input clk, reset,
         output [31:0] WriteData, DataAdr,
         output MemWrite,
        output [31:0] PC
     );
        wire [31:0] Instr, ReadData;
     // instantiate processor and memories
         riscvsingle rvsingle(clk, reset, PC, Instr, MemWrite,
             DataAdr, WriteData, ReadData);
11
         imem imem(PC, Instr);
12
         dmem dmem(clk, MemWrite, DataAdr, WriteData, ReadData);
    endmodule
14
```

III. Test Bench:

• Code:

```
module testbench();
    reg clk;
    reg reset;
    wire [31:0] WriteData, DataAdr, PC;
    wire MemWrite;
// instantiate device to be tested
    top dut(clk, reset, WriteData, DataAdr, MemWrite, PC);
   initial
        begin
            reset <= 1; # 22; reset <= 0;
// generate clock to sequence tests
    always
        begin
            clk <= 1; # 5; clk <= 0; # 5;
        end
    always @(negedge clk)
        begin
            if(MemWrite) begin
                 if(DataAdr === 32'h61 & WriteData === 32'h25) begin
                     $display("Simulation succeeded");
                else begin
                     $display("Simulation failed");
            if(PC === 32'h60) $display("PROGRAM COMPLETE");
endmodule
```

Assembly Program:

o Program:

```
main:
        addi x2, x0, 5
                              # x2 = 5
                                                                   00500113
        addi x3, x0, 12
                               \# x3 = 12
                                                         4
                                                                   00C00193
        addi x7, x3, -9
                               # x7 = (12 - 9) = 3
                                                         8
                                                                   FF718393
        or
            x4, x7, x2
                               # x4 = (3 OR 5) = 7
                                                        C
                                                                   0023E233
                              # x5 = (12 AND 7) = 4
        and
            x5, x3, x4
                                                        10
                                                                   0041F2B3
                               # x5 = (4 + 7) = 11
        add
            x5, x5, x4
                                                        14
                                                                   004282B3
       beq
            x5, x7, end
                               # shouldn't be taken
                                                        18
                                                                   02728863
        bne
            x4, x0, around
                              # should be taken
                                                         1C
                                                                   00021463
                                                         20
        addi x5, x0, 0
                               # shouldn't happen
                                                                   00000293
                                                        24
around: add x7, x4, x5
                              # x7 = (7 + 11) = 18
                                                                   005203B3
        sub
                              # x7 = (18 - 5) = 13
                                                        28
            x7, x7, x2
                                                                   402383B3
            x7, 84(x3)
                               # [96] = 13
                                                         2C
                                                                   0471AA23
        SW
            x2, 96(x0)
                               # x2 = [96] = 13
        lw
                                                         30
                                                                   06002103
        add x9, x2, x5
                               # x9 = (13 + 11) = 24
                                                         34
                                                                   005104B3
        addi x4, x4, 0x7CC
                               # x4 = 7D3
                                                         38
                                                                   7CC20213
        jal x3, end
                              # jump to end, x3 = 0x40 3C
                                                                   008001EF
        addi x2, x0, 1
                              # shouldn't happen
                                                         40
                                                                   00100113
        add x2, x2, x9
                               # \times 2 = (13 + 24) = 37
                                                         44
end:
                                                                   00910133
            x2, 0x21(x3)
                               \# mem[61] = 0x25 = 37
                                                         48
        SW
                                                                   0221A0A3
                              # infinite loop
                                                         4C
done:
       beq x2, x2, done
                                                                   00210063
                                                         50
        addi x3, x0, 12
                              # x3 = 0
                                                                   00000193
        jalr x2, x3, 60
                              # jump to 0+8
                                                         54
                                                                   06018167
        add x9, x2, x5
                              # x9 = (13 + 11) = 24
                                                         58
                                                                   005104B3
        addi x7, x3, -9
                               # x7 = (12 - 9) = 3
                                                         5C
                                                                   FF718393
                                                         60
```

o Code in HEX:

00500113 00C00193 FF718393 0023E233 0041F2B3 004282B3 02728863 00021463 00000293 005203B3 402383B3 0471AA23 06002103 005104B3 7CC20213 008001EF 00100113 00910133 0221A0A3 00210063 00000193 06018167 005104B3 FF718393

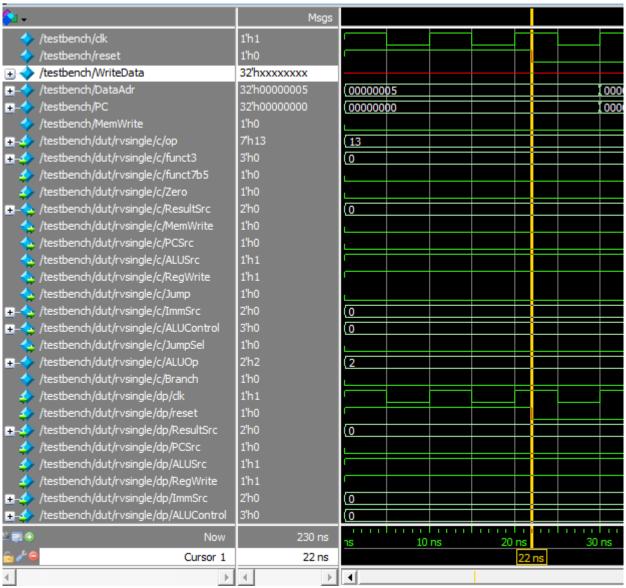
o Description :

- Program manipulates some registers performing different types of instructions.
- Next, on the last "store instruction" program checks on particular registers after manipulating their values making values manipulation among register serially, which means that the register will never reach this value unless instructions were correctly processed.
- Finally, test bench checks on the program counter to make sure that we've reached the ending point, meaning that all instructions, were correctly performed.

NOTE: Program will never reach the last program count unless all instructions we're well handled with no errors.

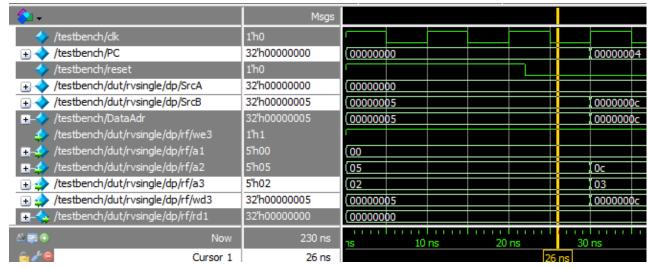
• Snapshots:

1- Reset was set high for 22ns and then toggled.

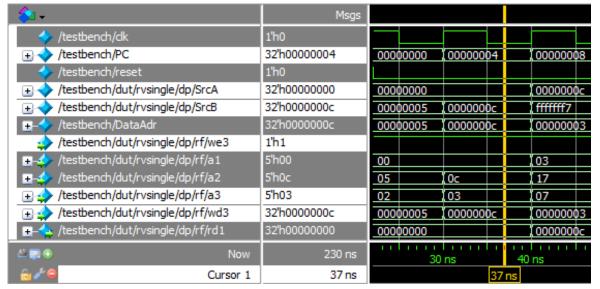


2- addi x2, x0, 5

At PC= 0, write 5 to address 2 in the reg file (x2).

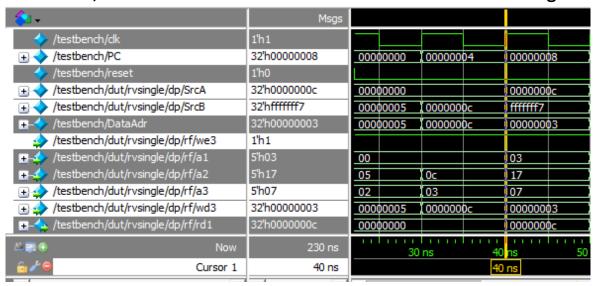


3- addi x3, x0, 12 # x3 = 12 At PC= 4, write 12 (0xC) to address 3 in the reg file (x3).



4- addi x7, x3, -9 # x7 = (12 - 9) = 3

At PC= 8, subtract and write 3 to address 7 in the reg file (x3).



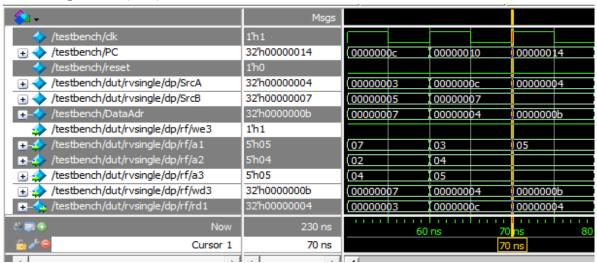
5- or x4, x7, x2 # x4 = (3 OR 5) = 7 At PC= C, perform bitwise 3 OR 5, and write 7 to address 4 in the reg file (x4).

€ 1+	Msgs				
/testbench/dk	1'h1				
	32'h0000000c	00000000	(00000004	00000008	0000000c
/testbench/reset	1'h0				
→ /testbench/dut/rvsingle/dp/SrcA	32'h00000003	00000000		(000000¢c	00000003
	32'h00000005	00000005	(0000000c	fffffff7	00000005
≖ - / /testbench/DataAdr	32'h00000007	00000005	(0000000c	000000003	00000007
/testbench/dut/rvsingle/dp/rf/we3	1'h1				
+ / /testbench/dut/rvsingle/dp/rf/a1	5'h07	00		03	07
-/-// /testbench/dut/rvsingle/dp/rf/a2	5'h02	05	Oc	17	02
→ /testbench/dut/rvsingle/dp/rf/a3	5'h04	02	03	07	04
<u>+</u>	32'h00000007	00000005	(0000000c	000000003	00000007
/testbench/dut/rvsingle/dp/rf/rd1	32'h00000003	00000000		↓000000φc	00000003
A ■ ■ Now	230 ns	31	liiiiliiii Ons 4	liiiiliiii Ons 50	ns 60
Cursor 1	50 ns			50	ns

6- and x5, x3, x4 # x5 = (12 AND 7) = 4
At PC= 10, perform bitwise 12 AND 7 and write 4 to address 5 in the reg file (x5).

\$ 1 →	Msgs			
/testbench/dk	1'h1			
	32'h00000010	(0000000c	00000010	00000014
/testbench/reset	1'h0			
	32'h0000000c	00000003	000000c	00000004
	32'h00000007	00000005	00000007	
∓ – ♦ /testbench/DataAdr	32'h00000004	00000007	00000004	0000000b
/testbench/dut/rvsingle/dp/rf/we3	1'h1			
	5'h03	07	03	05
	5'h04	02	04	
<u>→</u> /testbench/dut/rvsingle/dp/rf/a3	5'h05	04	05	
<u>→</u> /testbench/dut/rvsingle/dp/rf/wd3	32'h00000004	00000007	00000004	0000000b
	32'h0000000c	00000003	0000000c	00000004
△ 🛒 💿 Now	230 ns	60	ns 70	ns .
© ✓ © Cursor 1	60 ns	60	ns	

7- add x5, x5, x4 # x5 = (4 + 7) = 11 At PC= 14, perform addition and write 11 (0xB) to address 5 in the reg file (x5).



8- beq x5, x7, end # shouldn't be taken At PC= 18, check if x7(3) == x5(11) then the next PC = 44

\$ 1+	Msgs			
/testbench/dk	1'h1			
	32'h00000018	00000014	00000018	(0000001c
/testbench/reset	1'h0			
	32'h0000000b	00000004	000000фЬ	00000007
+ /testbench/dut/rvsingle/dp/SrcB	32'h00000003	0000007	00000003	(00000000
IIIIIIIIIIIII	32'h00000008	, 0000000р	00000008	00000007
/testbench/dut/rvsingle/dp/rf/we3	1'h0			
∓ - 4 /testbench/dut/rvsingle/dp/rf/a1	5'h05	05		(04
II -	5'h07	04	07	(00
	5'h10	05	10	(08
	32'h00000008	(0000000р	00000008	(00000007
IIIIIIIIIIIII	32'h0000000b	00000004	000000фЬ	00000007
!=-4 /testbench/dut/rvsingle/dp/rf/rd2	32'h00000003	00000007	00000003	(00000000)
/testbench/dut/rvsingle/c/Zero	1'h0			
→ /testbench/dut/rvsingle/c/Branch	1'h1			
/toothoods/MomMilito	150			
△ 🕶 🙃 Now	230 ns	ʻ0 ns 80	ns 9	Ons 100
G ✓ 9 Cursor 1	80 ns	80	ns	

9- bne x4, x0, around # should be taken At PC= 1C, if x4 \neq 0 then next PC= 24

\$ 1 ₹ 1	Msgs				
/testbench/clk	1'h1				
	32'h0000001c	00000014	00000018	0000001c	(00000024
/testbench/reset	1'h0				
<u>+</u> → /testbench/dut/rvsingle/dp/SrcA	32'h00000007	00000004	(000000фЬ	00000007	
→ /testbench/dut/rvsingle/dp/SrcB	32'h00000000	00000007	(0000000	00000000	(0000000Ь
∓ - ∜ /testbench/DataAdr	32'h00000007	(000000фЬ	(0000000	00000007	(00000012
/testbench/dut/rvsingle/dp/rf/we3	1'h0				
// /testbench/dut/rvsingle/dp/rf/a1	5'h04	05		04	
/testbench/dut/rvsingle/dp/rf/a2	5'h00	04	07	00	05
∓ -	5'h08	05	10	08	07
∓ -	32'h00000007	(0000000Ь	00000008	00000007	00000012
/testbench/dut/rvsingle/dp/rf/rd1	32'h00000007	00000004	(000000фЬ	00000007	
/testbench/dut/rvsingle/dp/rf/rd2	32'h00000000	00000007	00000003	00000000	(0000000Ь
∴ /testhench/dut/rvsinale/c/7ero	1'h0				
△ Now	230 ns	0 ns	80 ns 90	ns 10	0 ns 110
© Lursor 1	90 ns) ns	

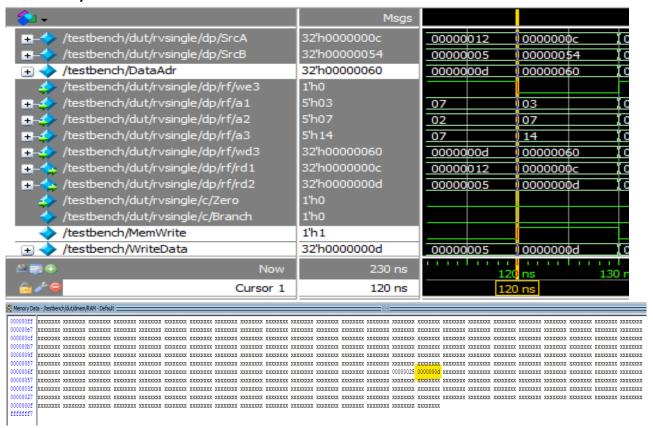
10- around: add x7, x4, x5 # x7 = (7 + 11) = 18 At PC= 24 perform addition and write 18 to address 7 in the reg file (x7).

≨ 1+	Msgs			
/testbench/dk	1'h1			
	32'h00000024	0000001c	00000024	00000028
/testbench/reset	1'h0			
	32'h00000007	00000007		00000012
	32'h0000000b	00000000	000000ФЬ	00000005
+	32'h00000012	00000007	00000012	0000000d
/testbench/dut/rvsingle/dp/rf/we3	1'h1			
+ // /testbench/dut/rvsingle/dp/rf/a1	5'h04	04		07
+ // /testbench/dut/rvsingle/dp/rf/a2	5'h05	00	05	02
<u>+</u>	5'h07	08	07	,
<u>+</u> → /testbench/dut/rvsingle/dp/rf/wd3	32'h00000012	00000007	00000012	, poooooo (
	32'h00000007	00000007		00000012
/testbench/dut/rvsingle/dp/rf/rd2	32'h0000000b	00000000	000000фЬ	000000005
/testbench/dut/rvsingle/c/Zero	1'h0			
△ Now	230 ns	100	ns 11	Dis 120
6 ∕ 6 Cursor 1	100 ns) ns	

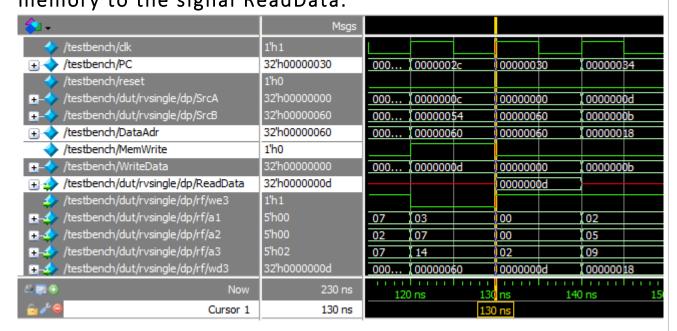
11- sub x7, x7, x2 # x7 = (18 - 5) = 13At PC= 28 perform subtraction and write 13 (0xD) to address 7 in the reg file (x7).

\$ 1 ₹	Msgs			
/testbench/clk	1'h1			
	32'h00000028	0000001c	00000024	00000028
/testbench/reset	1'h0			
	32'h00000012	00000007		00000012
	32'h00000005	00000000	(000000фь	000000005
- /testbench/DataAdr	32'h0000000d	00000007	00000012	0000000d
/testbench/dut/rvsingle/dp/rf/we3	1h1			
∓	5'h07	04		07
 	5'h02	00	05	02
	5'h07	08	07	<u> </u>
	32'h0000000d	00000007	00000012	0000000d (
 /testbench/dut/rvsingle/dp/rf/rd1	32'h00000012	00000007		00000012
 /testbench/dut/rvsingle/dp/rf/rd2	32'h00000005	00000000	(000000фь	00000005
/testbench/dut/rvsingle/c/Zero	1'h0			
A p ⊕ Now	230 ns	10	0 ns 11	0 ns 120

12- sw x7, 84(x3) # [96] = 13
At PC= 2C, store value 0xD in address 96 (0x60) in the data memory



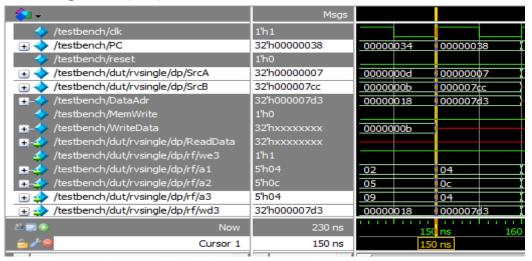
13- lw x2, 96(x0) # x2 = [96] = 13
At PC= 30, load value 0xD from address 96 (0x60) in the data memory to the signal ReadData.



14- add x9, x2, x5 # x9 = (13 + 11) = 24 At PC= 34, perform addition and write 24 to address 9 in the reg file (x9).

≨ 1 +	Msgs							
/testbench/dk	1'h1							
	32'h00000034	000	0000002c		000000	30	000000	34
/testbench/reset	1'h0							
	32'h0000000d	000	0000000c		0000000	00	000000	0d
→ /testbench/dut/rvsingle/dp/SrcB	32'h0000000b	000	00000054	10	0000006	0	000000	0b
+	32'h00000018	000	00000060		0000006	0	000000	18
/testbench/MemWrite	1'h0							
∓ - / /testbench/WriteData	32'h0000000b	000	D0000000	10	0000000	00	000000	0b
- - / testbench/dut/rvsingle/dp/ReadData	32'hxxxxxxxxx				0000000)d		
/testbench/dut/rvsingle/dp/rf/we3	1h1							
-/	5'h02	07	03		00		02	
-/-// /testbench/dut/rvsingle/dp/rf/a2	5'h05	02	07	(00		05	
<u>→</u> /testbench/dut/rvsingle/dp/rf/a3	5'h09	07	14	10	02		09	
<u>→</u>	32'h00000018	000	00000060	į (0000000)d	000000	18
△ 👺 🏵 Now	230 ns	120	liiiili Ons	130 :	iiiii ns	140	ns	150
© ✓ € Cursor 1	140 ns					140		

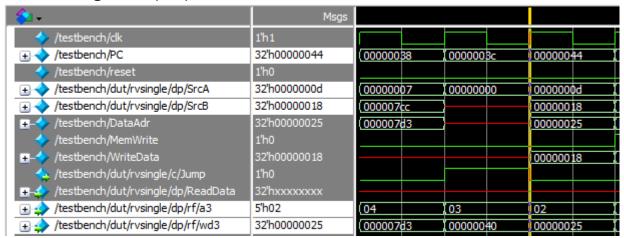
15- addi x4, x4, 0x7CC # x4 = 7D3
At PC= 38, add (0x7CC+ 7) and write 0x7D3 to address 4 in the reg file (x4).



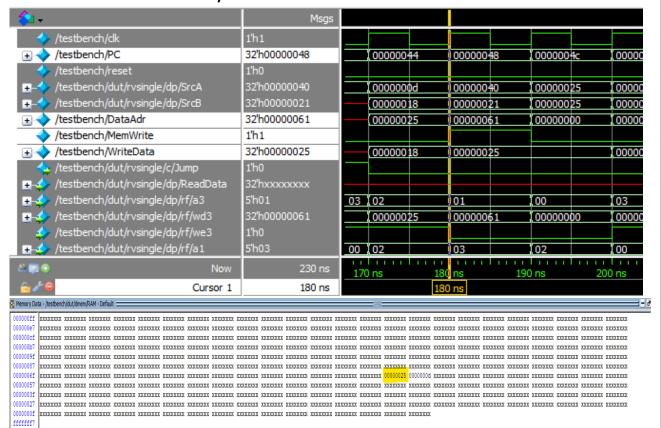
16- jal x3, end # jump to end, x3 = 0x40
At PC= 3C, make next PC= 44 and write the original next PC to address 3 in the reg file (x3).

\$1 →	Msgs			
/testbench/clk	1'h1			
	32'h0000003c	(00000038	0000003c	00000044
/testbench/reset	1'h0			
≖ - ∜ /testbench/dut/rvsingle/dp/SrcA	32'h00000000	00000007	00000000	P0000000
/testbench/dut/rvsingle/dp/SrcB	32'hxxxxxxxxx	(000007cc	<u> </u>	00000018
 → /testbench/DataAdr	32'hxxxxxxxxx	(000007d3		00000025
/testbench/MemWrite	1'h0			
+	32'hxxxxxxxxx			00000018
/testbench/dut/rvsingle/c/Jump	1'h1			
1 1 1 1 1 1 1 1 1 1	32'hxxxxxxxxx			
<u>→</u> /testbench/dut/rvsingle/dp/rf/a3	5'h03	04	03	02
→ /testbench/dut/rvsingle/dp/rf/wd3	32'h00000040	(000007d3	00000040	00000025
/testbench/dut/rvsingle/dp/rf/we3	1'h1			
	5'h00	04	00	02
[™] 🐺 🏵 Now	230 ns	1000	50 <mark>ns 17</mark>	70 ns

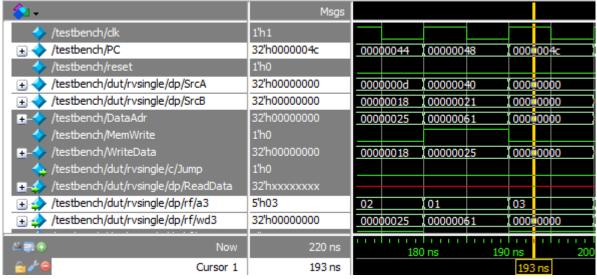
17- end: add x2, x2, x9 # x2 = (13 + 24) = 37 At PC= 44, add 13+24 and write 37 (0x25) to address 2 in the reg rile (x2).



18- sw x2, 0x21(x3) # mem[97] = 0x25 = 37
At PC= 48, store the value 37 (0x25) from (x2) to address
0x61 in data memory.

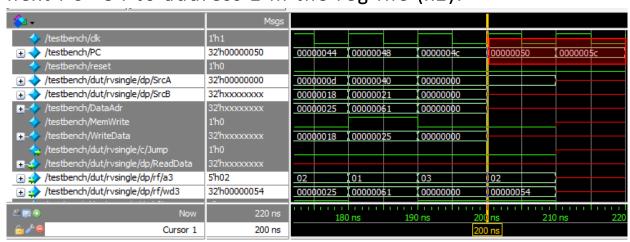


19- addi x3, x0, 0 # x3 = 0 At PC= 4C, write 0 to address 3 in the reg file (x3).



20- jalr x2, x3, 5C # jump to 0+8

At PC=50, set next PC= x3 + 5C = 5C, and write the original next PC= 54 to address 2 in the reg file (x2).



SUCCESSFUL Jump, and store...

21- END OF SIMULATION!

```
VOIM 192 restart
  ** Note: (vsim-8009) Loading existing optimized design opt2
  Loading work.testbench(fast)
 Loading work.top(fast)
 Loading work.riscvsingle(fast)
 Loading work.controller(fast)
 Loading work.maindec(fast)
 Loading work.aludec(fast)
 Loading work.datapath(fast)
# Loading work.flopr(fast)
# Loading work.adder(fast)
# Loading work.mux2(fast)
# Loading work.regfile(fast)
# Loading work.extend(fast)
# Loading work.alu(fast)
# Loading work.mux3(fast)
# Loading work.imem(fast)
# Loading work.dmem(fast)
VSIM 20> run
 Simulation succeeded
 PROGRAM COMPLETE
VSIM 21>
```