**Homework 1: RISC-V**

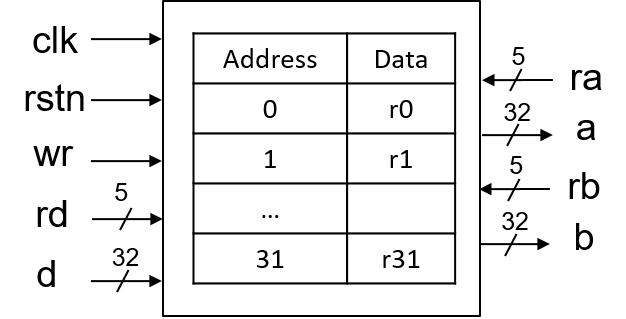
**Issued: December 27 (Tue.), 2021**  **Due: Jan 02 (Mon.), 2023**

**What to turn in**: Copy the text from your **MODIFIED** codes and paste it into a document. If a question asks you to plot or display something on the screen, include the plot and screen output your code generates. Submit either a \*.doc or \*.pdf file.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Problem 1 (15p): Register File**

Design a RISC-V register file (RF) in Verilog. Please see the description in the lecture note



What you have to do:

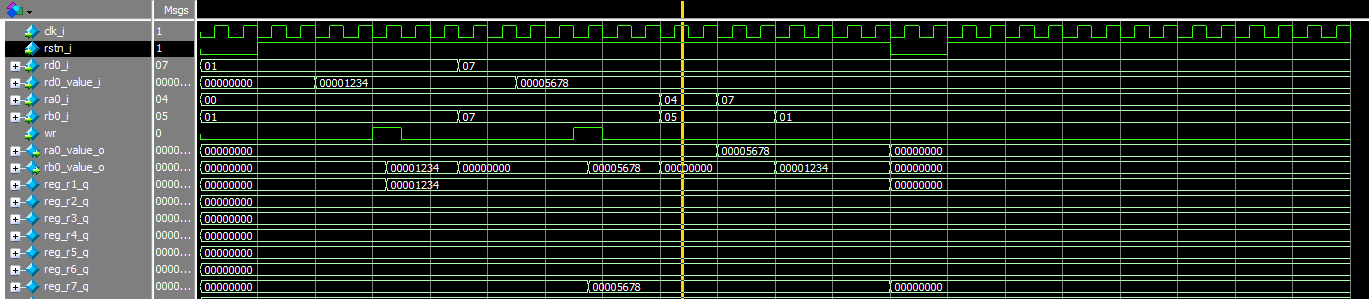
* Design an RF and its test bench based on the baseline codes.
* Submit your RTL files (riscv\_regfile.v and riscv\_regfile\_tb.v)

riscv\_regfile.v

1. /\*Insert your code \*/
2. reg\_r4\_q       <= 32'h00000000;
3. reg\_r5\_q       <= 32'h00000000;
4. reg\_r6\_q       <= 32'h00000000;
5. reg\_r7\_q       <= 32'h00000000;
6. /\*Insert your code \*/
7. **if**      (rd0\_i == 5'd16) reg\_r16\_q <= rd0\_value\_i;
8. **if**      (rd0\_i == 5'd17) reg\_r17\_q <= rd0\_value\_i;
9. **if**      (rd0\_i == 5'd18) reg\_r18\_q <= rd0\_value\_i;
10. **if**      (rd0\_i == 5'd19) reg\_r18\_q <= rd0\_value\_i;
11. /\*Insert your code \*/
12. 5'd10: ra0\_value\_r = reg\_r10\_q;
13. 5'd11: ra0\_value\_r = reg\_r11\_q;
14. 5'd12: ra0\_value\_r = reg\_r12\_q;
15. 5'd13: ra0\_value\_r = reg\_r13\_q;
16. /\*Insert your code \*/
17. 5'd7: rb0\_value\_r = reg\_r7\_q;
18. 5'd8: rb0\_value\_r = reg\_r8\_q;
19. 5'd9: rb0\_value\_r = reg\_r9\_q;
20. 5'd10: rb0\_value\_r = reg\_r10\_q;

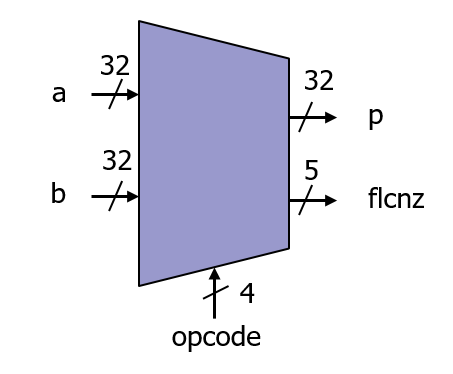
riscv\_regfile\_tb.v

* `timescale 1ns / 100ps
* **module** riscv\_regfile\_tb;
* wire [31:0] ra0\_value\_o, rb0\_value\_o;
* reg [4:0] ra0\_i, rb0\_i;
* reg [4:0] rd0\_i;
* reg [31:0] rd0\_value\_i;
* reg wr, clk, rstn;
* parameter CLOCK\_PERIOD = 10;
* // Components
* riscv\_regfile u\_riscv\_regfile
* (  .clk\_i(clk),
* .rstn\_i(rstn),
* .ra0\_i(ra0\_i),
* .rb0\_i(rb0\_i),
* .rd0\_i(rd0\_i),
* .rd0\_value\_i(rd0\_value\_i),
* .wr(wr),
* .ra0\_value\_o(ra0\_value\_o),
* .rb0\_value\_o(rb0\_value\_o)
* );
* // CLOCK
* initial **begin**
* clk = 0;
* forever #5 clk = ~clk;
* **end**
* // Testcase
* initial
* **begin**
* rstn = 0;
* ra0\_i = 5'b0000;        // Select R0
* rb0\_i = 5'b0001;        // Select R1
* wr    = 0;
* rd0\_i = 5'b0001;
* rd0\_value\_i = 32'h00000000;
* //#2        rstn = 0;
* #20     rstn = 1;       // Reset
* ////////// WRITE /////////
* ra0\_i = 5'b0000;        // Select R0
* rb0\_i = 5'b0001;        // Select R1
* #20     rd0\_value\_i = 32'h1234;
* #20     wr = 1;         // write to R1
* rd0\_i =  5'b0001;
* #10     wr = 0;
* #20     rb0\_i = 5'b0111;// Select R7
* rd0\_i =  5'b0111;
* #20     rd0\_value\_i = 32'h5678;
* #20     wr = 1;         // write to R7
* #10     wr = 0;
* ///////////////////////////
* /////////// READ //////////
* #20     ra0\_i = 5'b0100;        // Read R4
* rb0\_i = 5'b0101;        // Read R5
* #20     ra0\_i = 5'b0111;        // Read R7
* #20     rb0\_i = 5'b0001;        // Read R1
* ////////////////////////////
* #20
* #20     rstn = 0;
* #20     rstn = 1;
* **end**
* endmodule
* Capture the waveform.



**Problem 2 (15p): Arithmetic Logic Unit (ALU)**

Design a RISC-V ALU in Verilog. Please see the description in the lecture note.



What you have to do:

* Design an ALU with FLCNZ and its test bench based on the baseline codes.
* Submit your RTL files (riscv\_alu.v and riscv\_alu\_tb.v).

riscv\_alu.v

1. //-----------------------------------------------------------------
2. // Flag for exception cases: Homework
3. //-----------------------------------------------------------------
4. // Insert your code here
5. shift\_right\_fill\_r = 16'b0;
6. shift\_right\_1\_r = 32'b0;
7. shift\_right\_2\_r = 32'b0;
8. shift\_right\_4\_r = 32'b0;
9. shift\_right\_8\_r = 32'b0;
11. shift\_left\_1\_r = 32'b0;
12. shift\_left\_2\_r = 32'b0;
13. shift\_left\_4\_r = 32'b0;
14. shift\_left\_8\_r = 32'b0;
15. // Insert your code here
16. **if** (alu\_b\_i[2] == 1'b1)                                     //\*\*\* shift left 4(x8) to 7(x128)
17. shift\_left\_4\_r = {shift\_left\_1\_r[27:0],4'b0000};        //    :(alu\_b\_i[2:0]== 3b100 to 3b111)
18. **else**
19. shift\_left\_4\_r = shift\_left\_2\_r;
21. **if** (alu\_b\_i[3] == 1'b1)
22. shift\_left\_8\_r = {shift\_left\_1\_r[23:0],8'b00000000};        //\*\*\* shift left 8 to 15
23. **else**                                                        //    :(alu\_b\_i[3:0]== 4'b1000 to 4'b1111 )
24. shift\_left\_8\_r = shift\_left\_4\_r;
25. // Insert your code here
26. **if** (alu\_b\_i[0] == 1'b1)
27. shift\_right\_1\_r = {shift\_right\_fill\_r[31:31], alu\_a\_i[31:1]};
28. **else**
29. shift\_right\_1\_r = alu\_a\_i;
30. **if** (alu\_b\_i[1] == 1'b1)
31. shift\_right\_2\_r = {shift\_right\_fill\_r[31:30], shift\_right\_1\_r[31:2]};
32. **else**
33. shift\_right\_2\_r = shift\_right\_1\_r;
35. result\_r      = (alu\_a\_i | alu\_b\_i); //Insert your code //
36. result\_r      = (alu\_a\_i ^ alu\_b\_i);//Insert your code //
37. //Insert your code
38. **if** (alu\_a\_i[31] != alu\_b\_i[31])
39. result\_r  = (alu\_a\_i[31] < alu\_b\_i[31]) ? 32'h1 : 32'h0;
40. **else**
41. result\_r  = sub\_res\_w[31] ? 32'h1 : 32'h0;

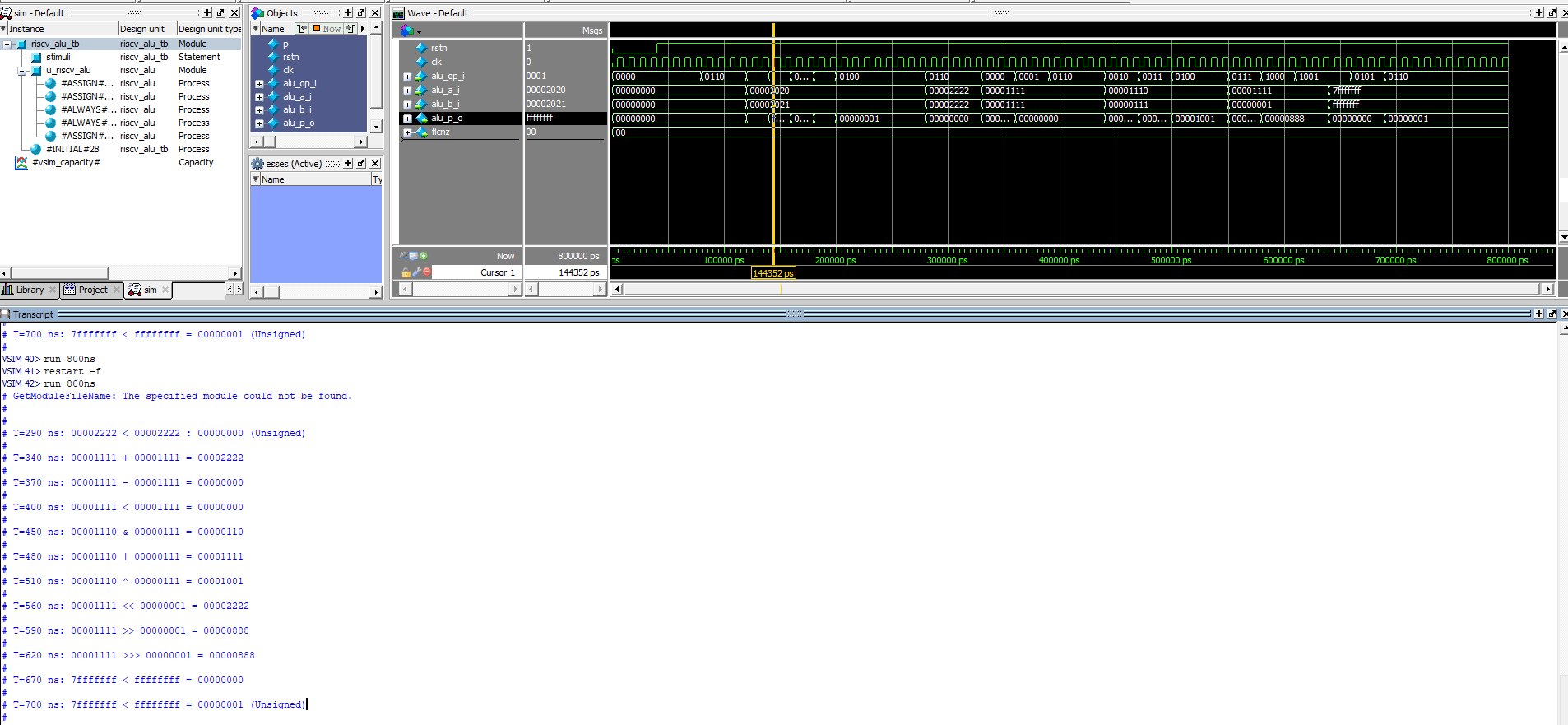
riscv\_alu\_tb.v

1. `timescale 1ns/1ps
3. `include"riscv\_defines.v"
5. module riscv\_alu\_tb ();
7. // Signals
8. reg rstn;
9. reg clk;
10. reg [3:0] alu\_op\_i;
11. reg [31:0] alu\_a\_i, alu\_b\_i;
12. wire [31:0] alu\_p\_o;
13. wire [4:0] flcnz;
15. // ALU module
16. riscv\_alu
17. u\_riscv\_alu  (
18. .alu\_a\_i(alu\_a\_i),
19. .alu\_b\_i(alu\_b\_i),
20. .alu\_op\_i(alu\_op\_i),
21. .alu\_p\_o(alu\_p\_o),
22. .flcnz(flcnz)
23. );
24. // Clock and Reset
25. parameter p=10;
26. initial
27. begin
28. clk = 1'b0;
29. forever #(p/2) clk = !clk;
30. end
32. initial
33. begin
34. rstn = 1'b0;        // negedge reset on
35. #(4\*p) rstn = 1'b1; // negedge reset off
36. end
38. // Test cases
39. initial
40. begin:stimuli
41. // Initialization
42. alu\_a\_i = 32'h0;
43. alu\_b\_i = 32'h0;
44. alu\_op\_i = `ALU\_ADD;
46. #(8\*p)  alu\_a\_i = 32'h0;
47. alu\_b\_i = 32'h0;
48. alu\_op\_i = `ALU\_SLTU;
50. #(4\*p)  alu\_a\_i = 32'h2020;
51. alu\_b\_i = 32'h2021;
52. alu\_op\_i = `ALU\_ADD;
53. #(2\*p)  alu\_op\_i = `ALU\_SUB;
55. #(2\*p)  alu\_op\_i = `ALU\_AND;
56. #(2\*p)  alu\_op\_i = `ALU\_OR;
57. #(2\*p)  alu\_op\_i = `ALU\_XOR;

60. //----------------------------------------------------------------------------------------
61. // ALU operations
62. //----------------------------------------------------------------------------------------
63. #(8\*p)
64. alu\_a\_i = 32'h0000\_2222;
65. alu\_b\_i = 32'h0000\_2222;
66. alu\_op\_i = `ALU\_SLTU;
67. #(p)   $display("T=%03t ns: %h < %h : %h (Unsigned)\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
69. // Addition
70. #(4\*p)
71. alu\_a\_i = 32'h0000\_1111;
72. alu\_b\_i = 32'h0000\_1111;
73. alu\_op\_i = `ALU\_ADD;
74. #(p)   $display("T=%03t ns: %h + %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
76. // Subtraction
77. #(2\*p) alu\_op\_i = `ALU\_SUB;
78. #(p)   $display("T=%03t ns: %h - %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
80. // Comparison
81. #(2\*p) alu\_op\_i = `ALU\_SLTU;
82. #(p)   $display("T=%03t ns: %h < %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
84. //----------------------------------------------------------------------------------------
85. // Logic operations
86. //----------------------------------------------------------------------------------------
87. // AND
88. #(4\*p)
89. alu\_a\_i = 32'h0000\_1110;
90. alu\_b\_i = 32'h0000\_0111;
91. alu\_op\_i = `ALU\_AND;
92. #(p)   $display("T=%03t ns: %h & %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
94. // OR
95. #(2\*p) alu\_op\_i = `ALU\_OR;
96. #(p)   $display("T=%03t ns: %h | %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
98. // XOR
99. #(2\*p) alu\_op\_i = `ALU\_XOR;
100. #(p)   $display("T=%03t ns: %h ^ %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);

103. //----------------------------------------------------------------------------------------
104. // Shift operations
105. //----------------------------------------------------------------------------------------
106. #(4\*p)
107. alu\_a\_i = 32'h0000\_1111;
108. alu\_b\_i = 32'h0000\_0001;
109. alu\_op\_i = `ALU\_SLL;
110. #(p)   $display("T=%03t ns: %h << %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
112. #(2\*p) alu\_op\_i = `ALU\_SRL;
113. #(p)   $display("T=%03t ns: %h >> %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
115. #(2\*p) alu\_op\_i = `ALU\_SRA;
116. #(p)   $display("T=%03t ns: %h >>> %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
118. #(2\*p)
119. alu\_a\_i = 32'h7FFF\_FFFF;
120. alu\_b\_i = 32'hFFFF\_FFFF;
122. #(2\*p) alu\_op\_i = `ALU\_SLT;
123. #(p)   $display("T=%03t ns: %h < %h = %h\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
125. #(2\*p) alu\_op\_i = `ALU\_SLTU;
126. #(p)   $display("T=%03t ns: %h < %h = %h (Unsigned)\n",$realtime/1000, alu\_a\_i, alu\_b\_i,alu\_p\_o);
127. end
128. endmodule

* Capture the waveform.

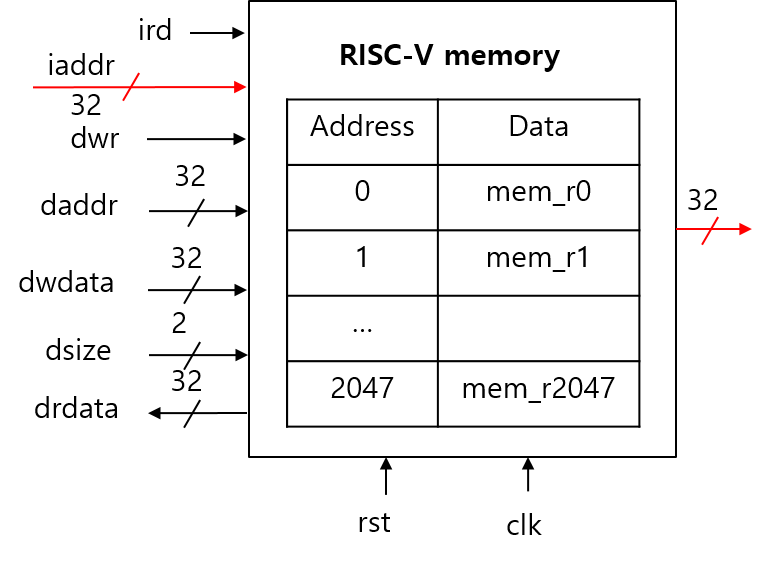


The FLCNZ flags are defined as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Flag | Opcode | Description | Remarks |
| C | ADD, SUB | Set C to HIGH if a carry/borrow from the most significant bit (MSB) position occurs when the operands are treated as unsigned numbers. | Arithmetic |
| F | ADD, SUB | Set F to HIGH if an overflow occurs when the operands are treated as two’s complement numbers (signed numbers). | Arithmetic |
| Z | SLT, SLTU | Perform a subtraction between Rsrc1 (a) and Rsrc2 (b) without writing back to Rdest (p) and set Z to HIGH if the result is zero | Comparison |
| L | SLT, SLTU | Perform a subtraction between Rsrc1 (a) and Rsrc2 (b) without writing back to Rdest (p) and set L to HIGH if a > b when the operands are treated as unsigned numbers. | Comparison |
| N | SLT, SLTU | Perform a subtraction between Rsrc1 (a) and Rsrc2 (b) without writing back to Rdest (p) and set the N flag if a > b when the operands are treated as signed numbers. | Comparison |

**Problem 3 (20p): Memory model**

Design a memory model in Verilog. Please see the description in the lecture note.



What you have to do:

1. Memory model (15p)

* Design a memory and its test bench based on the baseline codes.
* Submit your RTL files (riscv\_memory.v and riscv\_memory\_tb.v)

riscv\_memory.v

1. // Insert your code
2. **if** (ird\_i)
3. irdata\_r <= mem\_r[iaddr\_i[DEPTH:2]];/\*Insert your code \*/
4. end
5. // Insert your code
6. drdata\_r <= iaddr\_i;
7. // Insert your code
8. **if** (dbe\_w[2] && dwr\_i)              //    For HALFWORD mode, select 1 of 2 slots among 32bits
9. /\*Insert your code \*/  mem\_r[daddr\_i[DEPTH:2]][23:16]<= dwdata\_w[23:16];     //    4'b0011            : 4'b1100
10. //    lower 16 bits overwrite  higher 16 bits overwrite
11. **if** (dbe\_w[3] && dwr\_i)              //    For WORD mode
12. /\*Insert your code \*/  mem\_r[daddr\_i[DEPTH:2]][31:24]<= dwdata\_w[31:24];     //    full 32bits overwrite

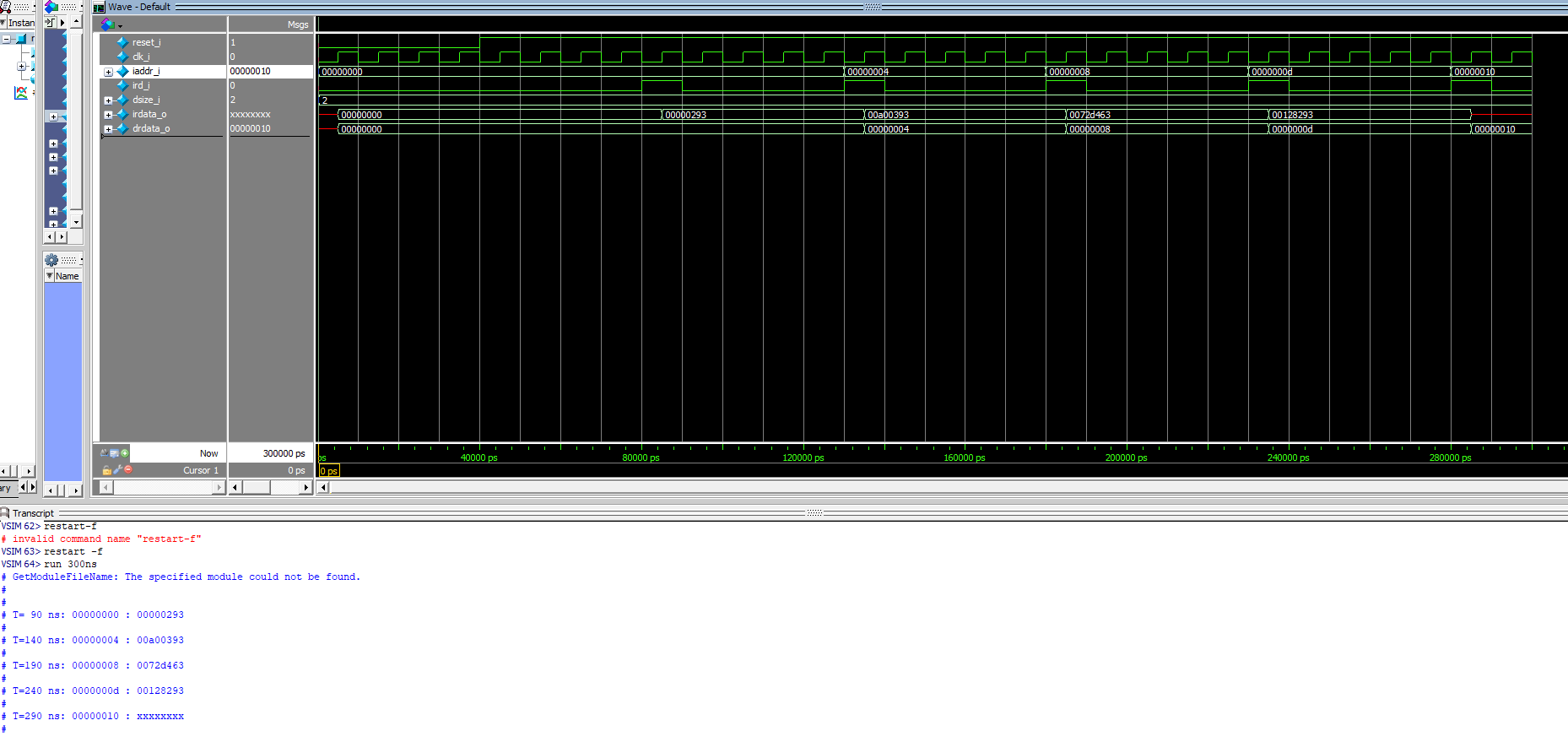
riscv\_memory\_tb.v

1. `timescale 1ns/1ps

4. module riscv\_memory\_tb ();
6. reg reset\_i;
7. reg clk\_i;
9. // Input instruction
10. reg  [31:0] iaddr\_i;
11. reg            ird\_i;
12. reg [31:0] daddr\_i;
13. reg [31:0] dwdata\_i;
14. reg [1:0]   dsize\_i;
15. reg       drd\_i;
16. reg           dwr\_i;
17. // Outputs
18. wire [31:0] irdata\_o;
19. wire [31:0] drdata\_o;
21. localparam
22. SIZE\_BYTE = 2'd0,
23. SIZE\_HALF = 2'd1,
24. SIZE\_WORD = 2'd2;
26. // Memory module
27. riscv\_memory
28. u\_memory (
29. .clk\_i(clk\_i),
30. .reset\_i(reset\_i),
31. .iaddr\_i(iaddr\_i),
32. .ird\_i(ird\_i),
33. .daddr\_i(daddr\_i),
34. .dwdata\_i(dwdata\_i),
35. .dsize\_i(dsize\_i),
36. .drd\_i(drd\_i),
37. .dwr\_i(dwr\_i),
38. .irdata\_o(irdata\_o),
39. .drdata\_o(drdata\_o)
40. );

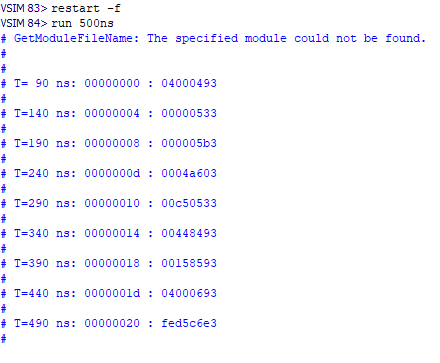

44. // Clock and Reset
45. parameter p=10;
46. initial begin
47. clk\_i = 1'b0;
48. forever #(p/2) clk\_i = !clk\_i;
49. end
51. // Test cases
52. initial
53. begin:stimuli
54. reset\_i = 1'b0;
55. iaddr\_i = 0;
56. ird\_i = 0;
57. dsize\_i=SIZE\_WORD;
58. #(4\*p) reset\_i = 1'b1;
60. #(4\*p)  iaddr\_i = 32'h0;
61. ird\_i = 1'b1;
62. #(p)    ird\_i = 1'b0;
63. $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr\_i, irdata\_o);
64. #(4\*p)  iaddr\_i = 32'h4;
65. ird\_i = 1'b1;
66. #(p)    ird\_i = 1'b0;
67. $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr\_i, irdata\_o);
68. #(4\*p)  iaddr\_i = 32'h8;
69. ird\_i = 1'b1;
70. #(p)    ird\_i = 1'b0;
71. $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr\_i, irdata\_o);
72. #(4\*p)  iaddr\_i = 32'hd;
73. ird\_i = 1'b1;
74. #(p)    ird\_i = 1'b0;
75. $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr\_i, irdata\_o);
76. #(4\*p)  iaddr\_i = 32'h10;
77. ird\_i = 1'b1;
78. #(p)    ird\_i = 1'b0;
79. $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr\_i, irdata\_o);
80. end
81. endmodule

* Capture the waveform.



1. Program memory file (5p)

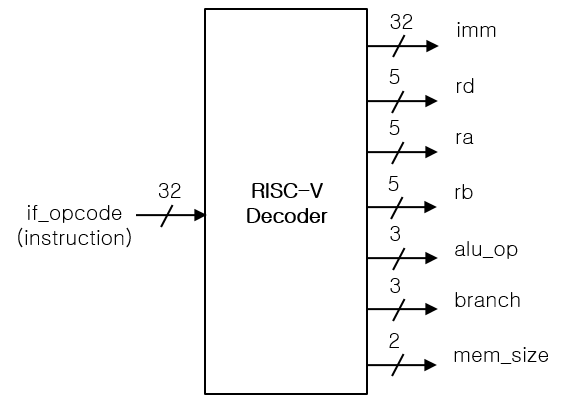
* Use the RISC-V simulator Venus (<https://www.kvakil.me/venus/>) to generate a program file.
* Modify the test bench (riscv\_memory\_tb.v) to simulate the memory model with the newly generated file.
* Capture the waveform and the display result that is printed out in Transcript Window.



|  |  |  |
| --- | --- | --- |
| **C code** | **Assembly Code** | |
| int A[64];  int sum = 0;  for (int i=0; i<64; i++)  sum += A[i]; | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:  lw x12, 0(x9)  add x10, x10, x12  addi x9, x9, 4  addi x11, x11, 1  addi x13, x0, 64  blt x11, x13, Loop | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |

**Problem 4 (15p): RISC-V Decoder**

Design a RISC-V decoder in Verilog. Please see the description in the lecture note.



What you have to do:

* Design a RISC-V decoder and its test bench based on the baseline codes.
* Submit your RTL files (riscv\_decoder.v and riscv\_decoder\_tb.v).

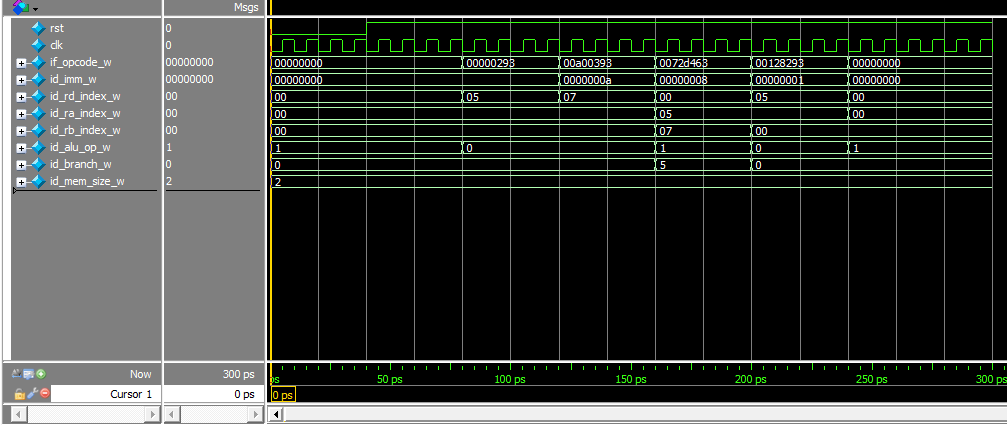
riscv\_decoder.v

1. assign ra\_w = if\_opcode\_w[19:15];/\*your code\*/
2. assign op\_alu\_reg\_w = (7'b0110011 == op\_w);/\*your code\*/
3. assign op\_f7\_alt\_w  = (7'b0100000 == f7\_w);/\*your code\*/
4. assign auipc\_w  = (7'b0010111  == op\_w);/\*your code\*/
5. assign bne\_w    = op\_branch\_w  && (3'b001 == f3\_w);/\*your code\*/
6. assign lw\_w     = op\_load\_w    && (3'b010 == f3\_w);/\*your code\*/
7. assign sb\_w     = op\_store\_w   && (3'b000 == f3\_w);/\*your code\*/
8. assign xori\_w   = op\_alu\_imm\_w && (3'b100 == f3\_w);/\*your code\*/
9. assign slli\_w   = op\_alu\_imm\_w && (3'b001 == f3\_w) && op\_f7\_main\_w;/\*your code\*/
10. assign slt\_w    = op\_alu\_reg\_w && (3'b010 == f3\_w) && op\_f7\_main\_w;/\*your code\*/
11. assign srl\_w    = op\_alu\_reg\_w && (3'b101 == f3\_w) && op\_f7\_main\_w;/\*your code\*/
12. assign mulh\_w   = op\_alu\_reg\_w && (3'b001 == f3\_w) && op\_f7\_mul\_w;/\*your code\*/
13. assign remu\_w   = op\_alu\_reg\_w && (3'b111 == f3\_w) && op\_f7\_mul\_w;/\*your code\*/

riscv\_decoder\_tb.v

1. module riscv\_decoder\_tb;
2. reg rst;
3. reg clk;
5. // Input instruction
6. reg  [31:0] if\_opcode\_w;
7. // Outputs
8. wire [31:0] id\_imm\_w;
9. wire [4:0] id\_rd\_index\_w;
10. wire [4:0] id\_ra\_index\_w;
11. wire [4:0] id\_rb\_index\_w;
12. wire [3:0] id\_alu\_op\_w;
13. wire [2:0] id\_branch\_w;
14. wire [1:0] id\_mem\_size\_w;
15. //Flags
16. wire mulh\_w;
17. wire mulhsu\_w;
18. wire div\_w;
19. wire rem\_w;
20. wire sra\_w;
21. wire srai\_w;
22. wire alu\_imm\_w;
23. wire jal\_w;
24. wire load\_w;
25. wire store\_w;
26. wire lbu\_w;
27. wire lhu\_w;
28. wire jalr\_w;
29. wire id\_illegal\_w;
31. riscv\_decoder
32. u\_decoder
33. (
34. ./\*input  [31:0]\*/ if\_opcode\_w(if\_opcode\_w),
35. ./\*output [31:0]\*/ id\_imm\_w(id\_imm\_w),
36. ./\*output [4:0] \*/id\_rd\_index\_w(id\_rd\_index\_w),
37. ./\*output [4:0] \*/id\_ra\_index\_w(id\_ra\_index\_w),
38. ./\*output [4:0] \*/id\_rb\_index\_w(id\_rb\_index\_w),
39. ./\*output [3:0] \*/id\_alu\_op\_w(id\_alu\_op\_w),
40. ./\*output [2:0] \*/id\_branch\_w(id\_branch\_w),
41. ./\*output [1:0] \*/id\_mem\_size\_w(id\_mem\_size\_w),
42. //Flags
43. ./\*output \*/mulh\_w(mulh\_w),
44. ./\*output \*/mulhsu\_w(mulhsu\_w),
45. ./\*output \*/div\_w(div\_w),
46. ./\*output \*/rem\_w(rem\_w),
47. ./\*output \*/sra\_w(sra\_w),
48. ./\*output \*/srai\_w(srai\_w),
49. ./\*output \*/alu\_imm\_w(alu\_imm\_w),
50. ./\*output \*/jal\_w(jal\_w),
51. ./\*output \*/load\_w(load\_w),
52. ./\*output \*/store\_w(store\_w),
53. ./\*output \*/lbu\_w(lbu\_w),
54. ./\*output \*/lhu\_w(lhu\_w),
55. ./\*output \*/jalr\_w(jalr\_w),
56. ./\*output \*/id\_illegal\_w(id\_illegal\_w)
57. );
59. parameter p=10;
61. initial
62. begin
63. clk = 1'b0;
64. forever #(p/2) clk = !clk;
65. end
67. initial
68. begin
69. rst = 1'b0;     // negedge reset on
70. if\_opcode\_w = 0;
71. #(4\*p) rst = 1'b1;  // negedge reset off
72. /\*
73. C:
74. for (int i = 0;i < 10;i++) {
75. // Repeated code goes here.
76. }
78. Assembly:
79. # t0 = 0
80. li      t0, 0
81. li      t2, 10
82. loop\_head:
83. bge     t0, t2, loop\_end
84. # Repeated code goes here
85. addi    t0, t0, 1
86. loop\_end:
88. // Venus
89. 0x00000293  addi x5 x0 0    li t0, 0
90. 0x00a00393  addi x7 x0 10   li t2, 10
91. 0x0072d463  bge x5 x7 8 bge t0, t2, loop\_end
92. 0x00128293  addi x5 x5 1    addi t0, t0, 1
93. \*/
94. #(4\*p) if\_opcode\_w = 32'h00000293;
95. #(4\*p) if\_opcode\_w = 32'h00a00393;
96. #(4\*p) if\_opcode\_w = 32'h0072d463;
97. #(4\*p) if\_opcode\_w = 32'h00128293;
98. #(4\*p) if\_opcode\_w = 32'h0;
99. end
100. endmodule

* Capture the waveform.



**Problem 5 (15p): Program Counter**

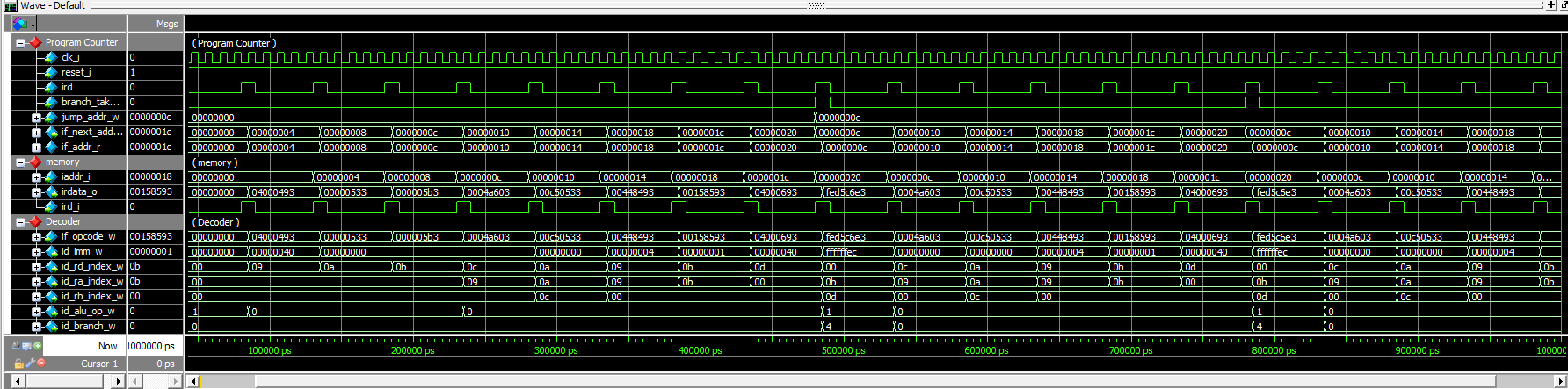
Design a program counter in Verilog. Please see the description in the lecture note for details.

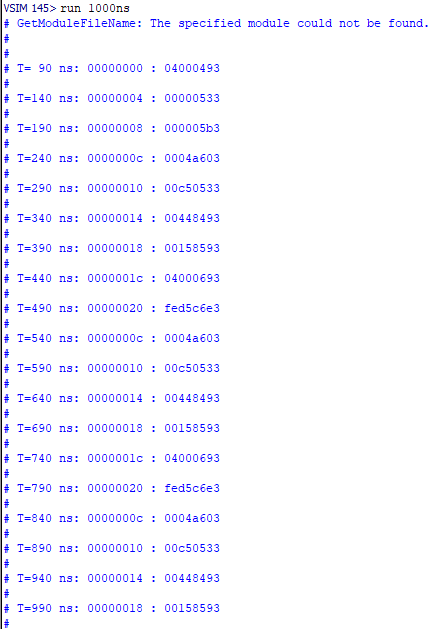
C:\Users\User\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HW04_Progam_counter.tif

What you have to do:

1. Program Counter (10p)

* Complete a program counter and test bench based on the baseline codes.
* Do simulation and capture its result, i.e., waveform and transcript window.





1. Extended program counter (5p)

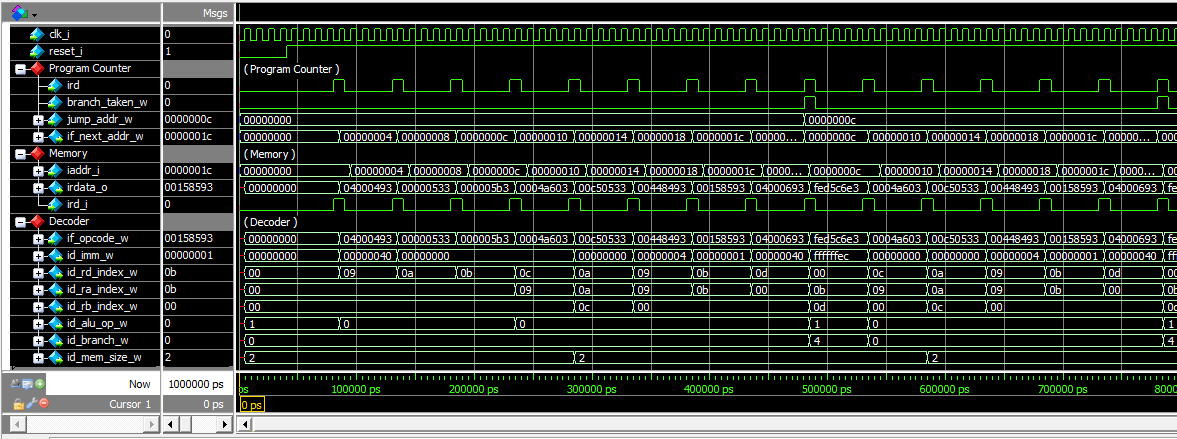
Compared to riscv\_pc\_tb1, riscv\_pc\_tb2 connects iaddr\_i to the output of the Program Counter.

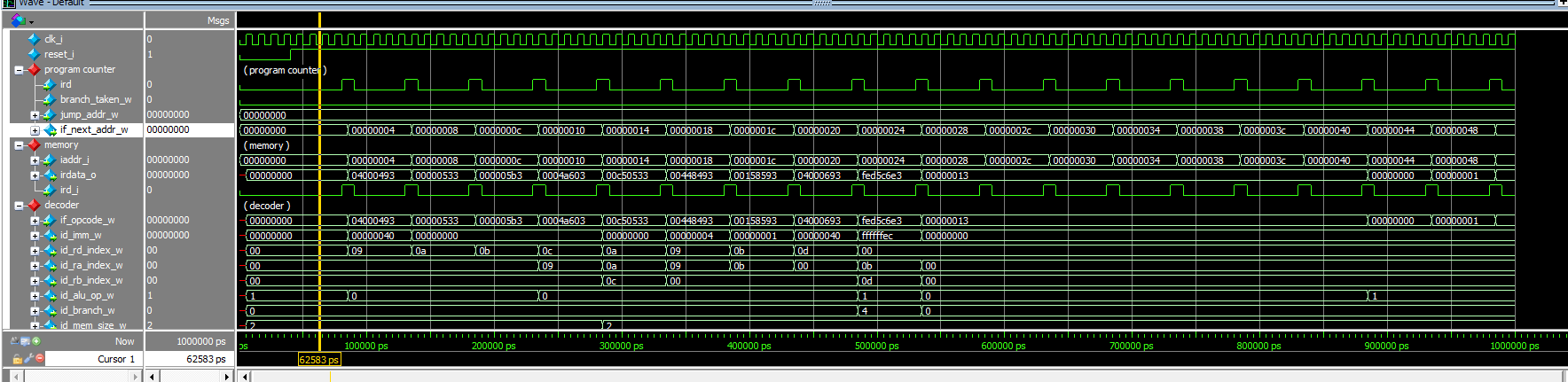
* Complete the test bench (riscv\_pc\_tb2.v).

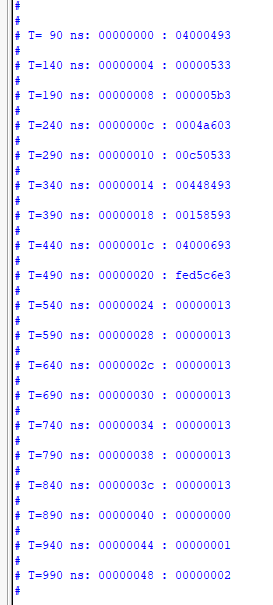
riscv\_pc\_tb2.v

1. //Insert Your code
2. //{{{
3. ./\*input                \*/ird(ird\_i),               // Instruction Read request
4. ./\*input                \*/branch\_taken\_w(branch\_taken\_w),   // Jump instruction
5. ./\*input  [PC\_SIZE-1:0] \*/jump\_addr\_w(jump\_addr\_w),     // Jump address
6. ./\*output [PC\_SIZE-1:0] \*/if\_next\_addr\_w(if\_next\_addr\_w)        // Next instruction
7. //}}}
8. //Insert Your code
9. //{{{
10. .iaddr\_i(if\_next\_addr\_w),
11. .ird\_i(ird\_i),
12. //}}}

* Do simulation and capture its result, i.e., waveform and transcript window.





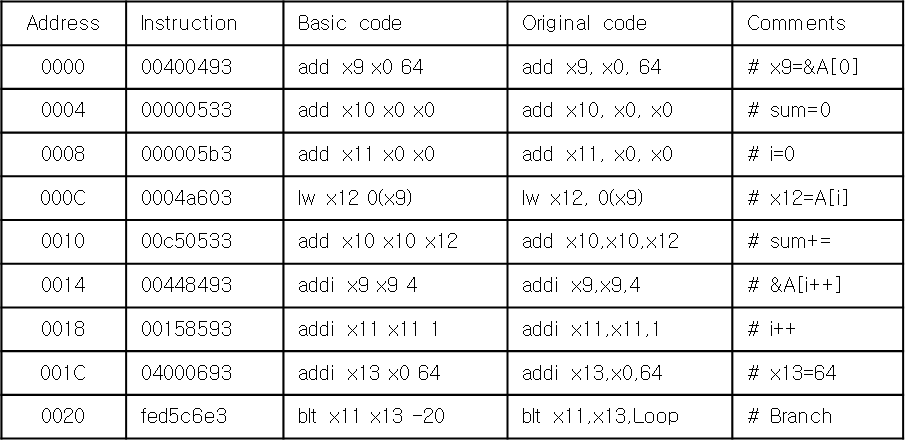


**Problem 6 (20p): Program**

Design a simplified RISC-V core in Verilog. Please see the description in the lecture note.



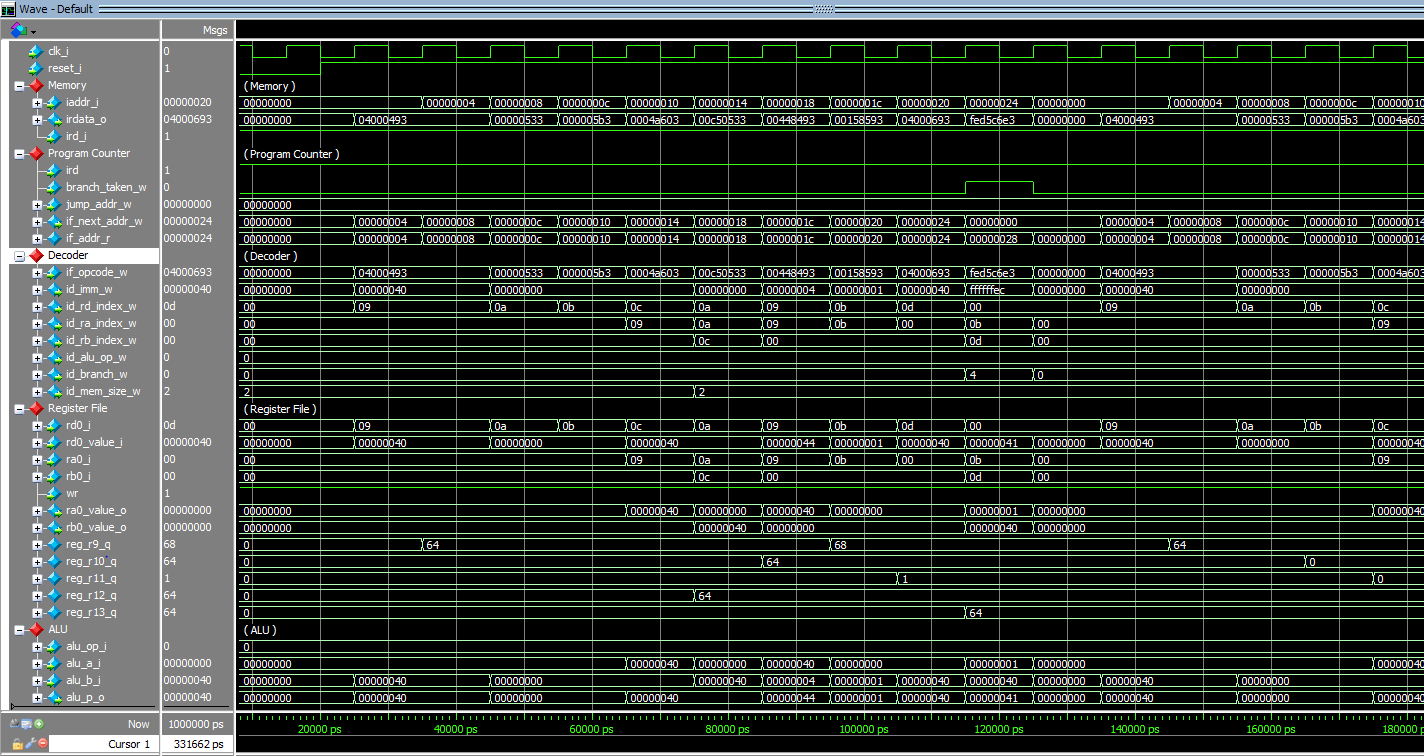
The assembly code (Problem 2b):

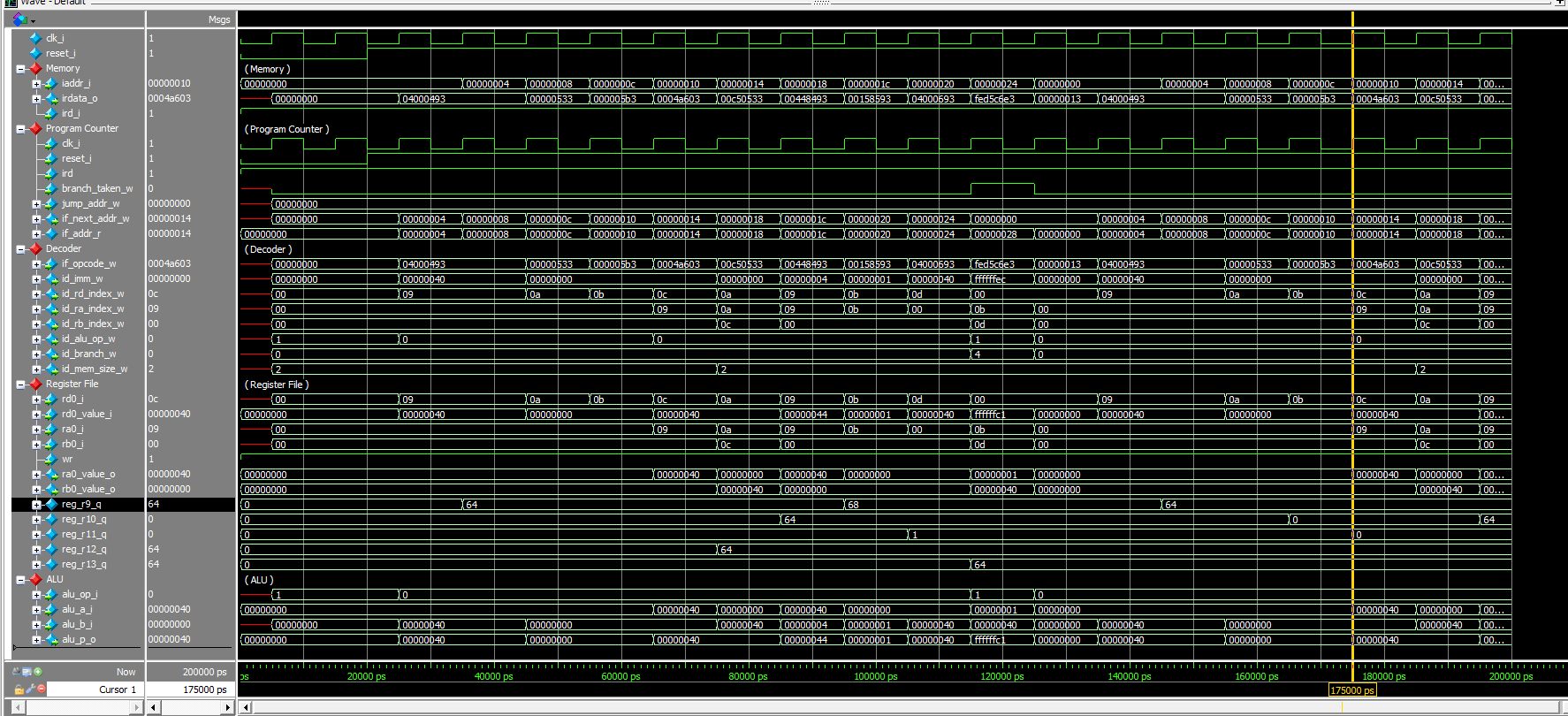


What you have to do:

1. Baseline core (10p)

* Use Program Counter in Problem and complete all connections for ALU, Decoder, and Register File modules.
* Do a simulation and capture its result, i.e., waveform.





1. Default target address and branch enable signal for Brach-Less-Than BLT (2p)

In the baseline mode for (BLT), the target address for Brach-Less-Than (BLT) is set to 0, while the branch “enable” signal is **always set to 1**. Explain why r11 in Register File only has two values, 0 and 1.

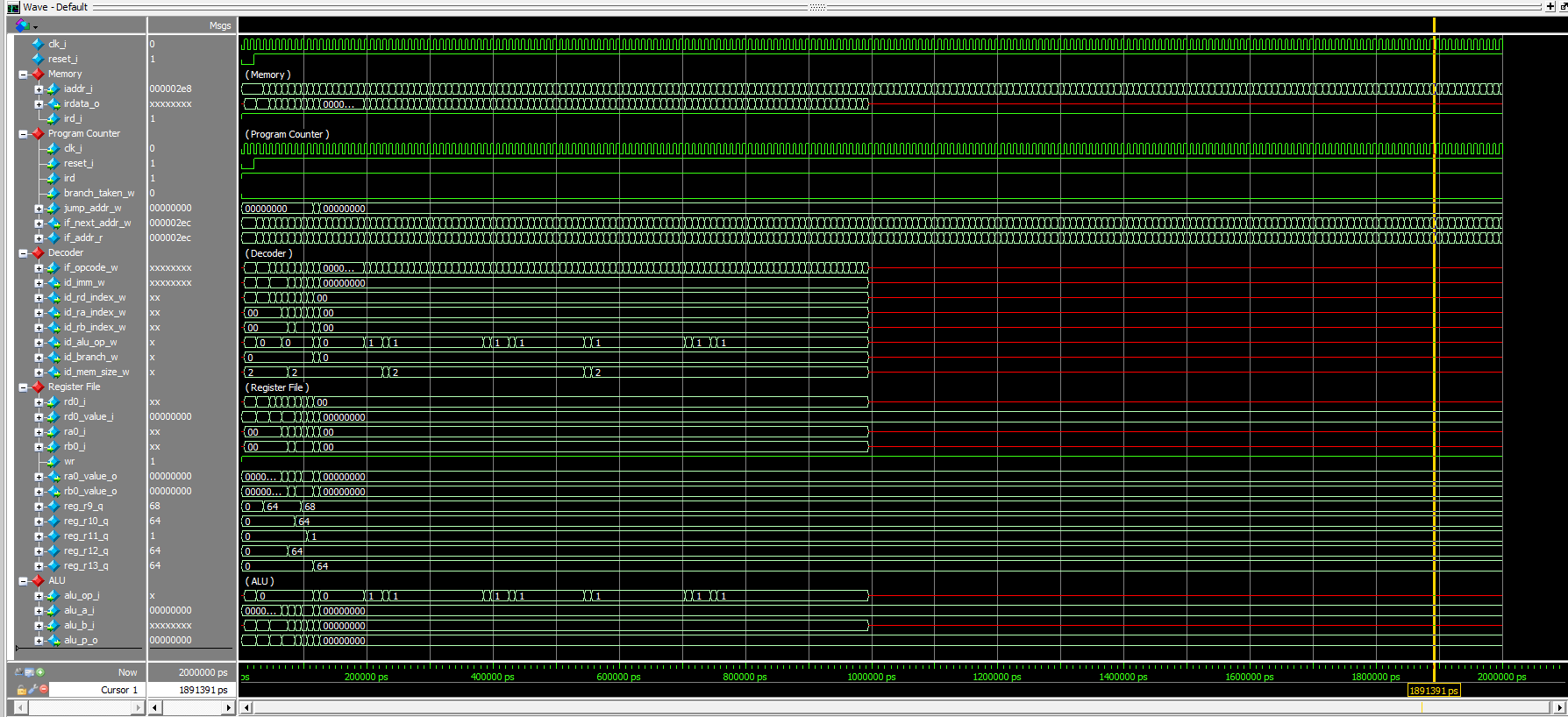
Answer:

The instruction `add x11 x0 x0` will make the r11 in Register File to be 0, then in the loop `addi x11 x11 1` will make the r11 in Register File to be 1. At the loop condition judgment `blt x11, x13, Loop` will not jump to the target address which make the instruction will begin initially. Thus, After the one loop, the r11 will be initialized with 0 instead of be added 1.

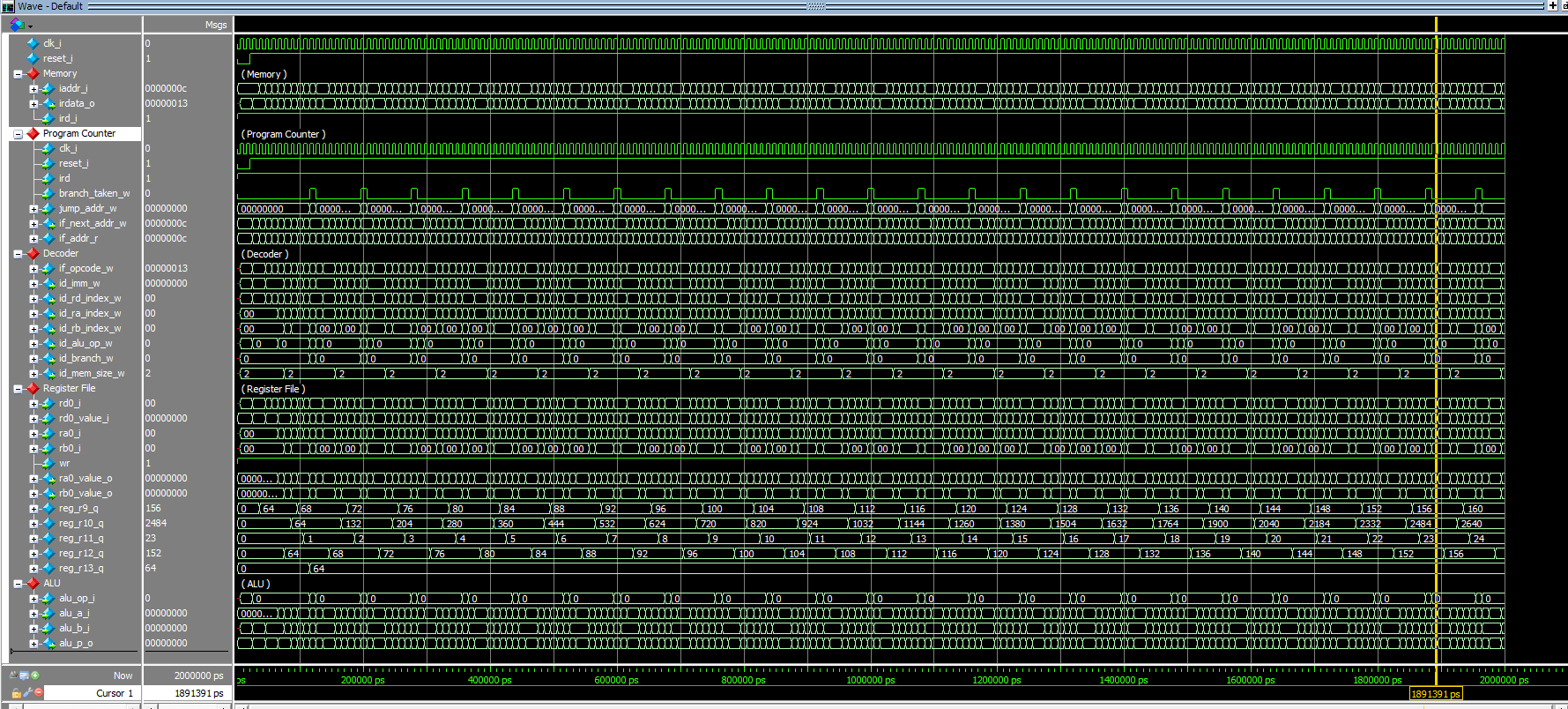
1. Branch instruction Brach-Less-Than (BLT) (8p)
   1. Target Address for BLT (4p)

* Make code to calculate a target address for BLT.
* Do a simulation and capture its result, i.e., waveform.





* 1. Target enable for BLT (4p)
* Make code to generate a branch enable signal for BLT
* Do a simulation and capture its result, i.e., waveform.



**Problem 7 (8p) (Optional): Bonus**

1. (2p) In Problem 2, a shift module can be implemented in two ways: 1) using a “case” expression; or 2) using a barrel shift approach. How many shifters with a constant shifting amount are used in both cases? Generalize the problem for given n-bit numbers a and b.

**Answer:** I think the barrel shift approach is more convenient and efficiency than the “case” expression. For the given n-bit numbers to shift, the “case” expression needs n-shifters and the barrel shift approach needs clog2(n)-bit numbers.

1. (1p) Assume that each operation in ALU is described in a block. Draw a block diagram of ALU.

**Answer:**

Register

Register

ALU

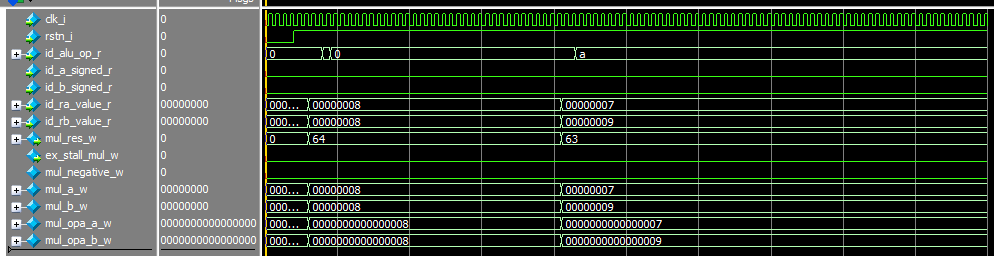
1. (2p) In problem 3, a slow sequential multiplier completes a 32-bit multiplication in 32 cycles. Is a computation result correct if the input is changed during computation? Modify the code to make the computation result correct even though input is changed during computation. Show the test cases and the waveform corresponding to both the baseline code and a modified code.

**Answer:**

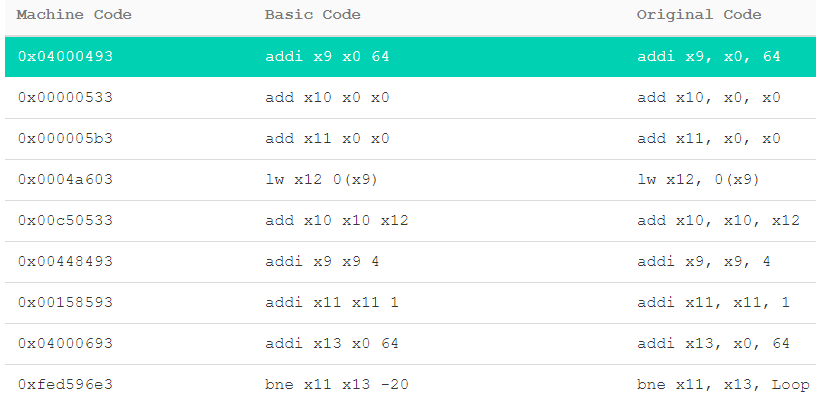
Modified code:

1. //Insert your code
2. mul\_count\_r  <= mul\_count\_r - 5'd1;
3. /\*Insert your code \*/                               //\*\*\* counter check for state-end
4. mul\_res\_r   <= { mul\_sum\_w, mul\_res\_r[31:1] };    //\*\*\* 64bit width register composed by previous adder result and multiplier b
5. //\*\*\* uppper with adder result (or initial multiplicand a), lower with multiplier b
6. //\*\*\* 1bit shift right per state change
7. // Insert your code
8. mul\_ready\_r <= 1'b1;
9. // Insert your code
10. mul\_res\_r   <= { 32'h0, id\_ra\_value\_r};  //\*\*\* intial value with upper 32bits zero and lower 32bit multiplicand a

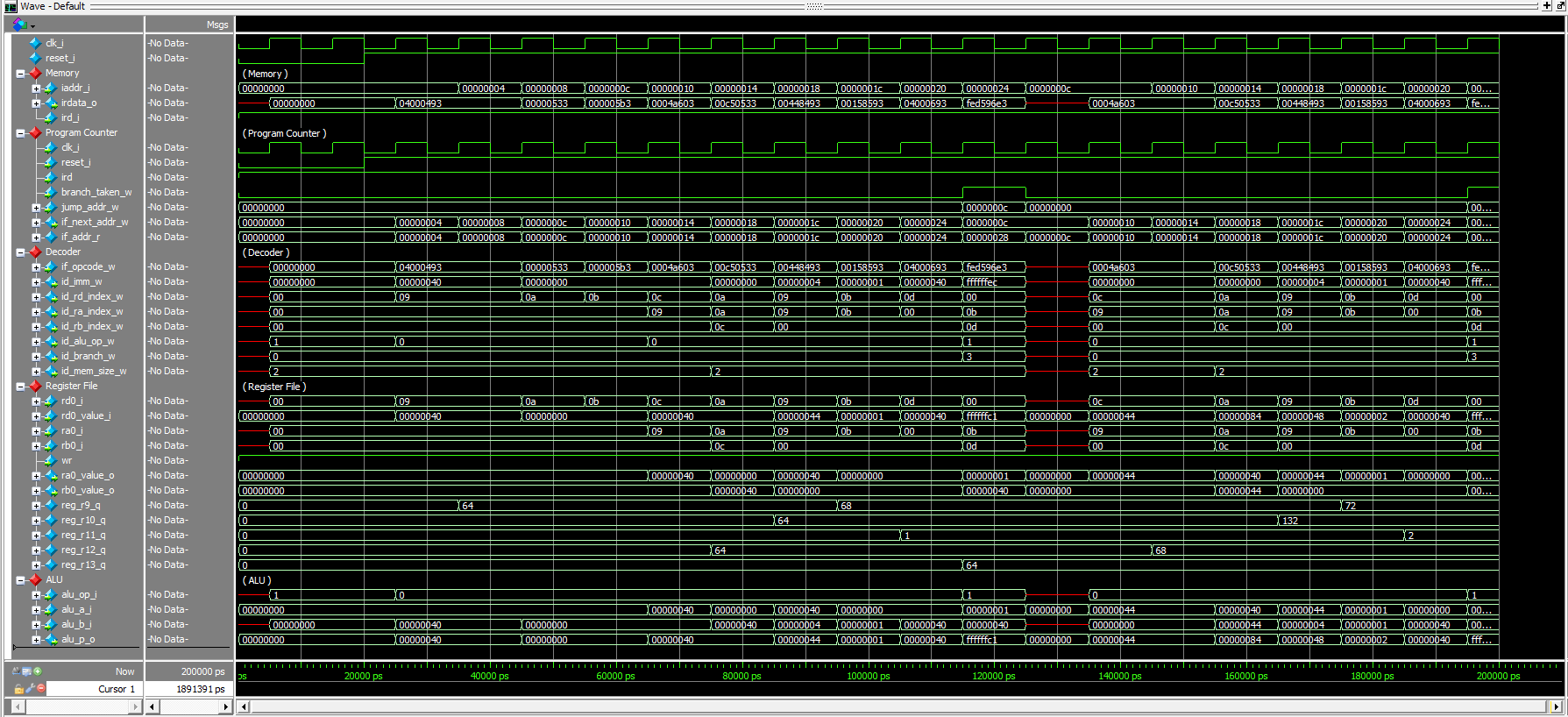
WaveForm:



1. (3p) The following table shows C codes and their assembly codes for a program that calculates a sum of all elements of an array. Two code versions use LESS-THAN and NOT-EQUAL operations for a conditional branch in a loop. In this problem, you have to do:
2. Generate a memory file for the modified assembly code using a NOT-EQUAL operation, called mem\_bne.hex.



1. Modify a test bench to use a new memory file (riscv\_core\_sim\_bne\_tb.v).
2. `timescale 1ns / 100ps
3. module riscv\_core\_sim\_bne\_tb;
4. reg clk\_i;
5. reg reset\_i;
6. wire        lock;
7. wire [31:0] iaddr;
8. wire [31:0] irdata;
9. wire        ird;
10. wire [31:0] daddr;
11. wire [31:0] dwdata;
12. wire [31:0] drdata;
13. wire  [1:0] dsize;
14. wire        drd;
15. wire        dwr;
17. riscv\_memory #(.FIRMWARE("mem\_bne.hex"))
18. u\_riscv\_memory
19. (
20. ./\*input         \*/clk\_i(clk\_i),
21. ./\*input         \*/reset\_i(reset\_i),
22. ./\*input  [31:0] \*/iaddr\_i(iaddr),
23. ./\*output [31:0] \*/irdata\_o(irdata),
24. ./\*input         \*/ird\_i(ird),
25. ./\*input  [31:0] \*/daddr\_i(daddr),
26. ./\*input  [31:0] \*/dwdata\_i(dwdata),
27. ./\*output [31:0] \*/drdata\_o(drdata),
28. ./\*input   [1:0] \*/dsize\_i(dsize),
29. ./\*input         \*/drd\_i(drd),
30. ./\*input         \*/dwr\_i(dwr)
31. );
33. riscv\_core\_sim
34. u\_riscv\_core\_sim
35. (
36. ./\*input         \*/clk\_i(clk\_i),
37. ./\*input         \*/reset\_i(reset\_i),
38. ./\*output        \*/lock\_o(lock),
39. ./\*output [31:0] \*/iaddr\_o(iaddr),
40. ./\*input  [31:0] \*/irdata\_i(irdata),
41. ./\*output        \*/ird\_o(ird),
42. ./\*output [31:0] \*/daddr\_o(daddr),
43. ./\*output [31:0] \*/dwdata\_o(dwdata),
44. ./\*input  [31:0] \*/drdata\_i(drdata),
45. ./\*output  [1:0] \*/dsize\_o(dsize),
46. ./\*output        \*/drd\_o(drd),
47. ./\*output        \*/dwr\_o(dwr)
48. );
49. // CLOCK
50. initial begin
51. clk\_i = 0;
52. forever #5 clk\_i = ~clk\_i;
53. end
55. // Testcase
56. initial
57. begin
58. reset\_i = 1'b0;
60. #20 reset\_i = 1'b1; // Reset
62. end
64. endmodule
65. Make code to compute a target address and a branch enable signal for Branch-Not-Equal (BEQ).
66. // Insert your code
67. //{{{
68. branch\_taken\_w = 1'b1;
69. jump\_addr\_w = if\_pc\_d + id\_imm\_w;
70. //}}}
71. Do a simulation and capture its waveform.



|  |  |  |
| --- | --- | --- |
| **C code** | **Assembly Code** | |
| int A[64];  int sum = 0;  for (int i=0; i<64; i++)  sum += A[i]; | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:  lw x12, 0(x9)  add x10, x10, x12  addi x9, x9, 4  addi x11, x11, 1  addi x13, x0, 64  blt x11, x13, Loop | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |
| int A[64];  int sum = 0;  for (int i=0; i≠64; i++)  sum += A[i]; | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:  lw x12, 0(x9)  add x10, x10, x12  addi x9, x9, 4  addi x11, x11, 1  addi x13, x0, 64  bne x11, x13, Loop | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |