

---

UM-SJTU JOINT INSTITUTE  
Introduction to Computer Organization  
(VE370)

---

PROJECT REPORT  
PROJECT 2

Name: Wu Yuhao	ID:517370910119	Group: 17
Name: Zhong Xiaoxue	ID:517370910076	
Name: Jin Zhejian	ID:517370910167	

Date: 7 November 2019

# 1 Project Description

Model both single cycle and pipelined implementation of MIPS computer in Verilog that support a subset of MIPS instruction set including:

- The memory-reference instructions load word (lw) and store word (sw)
- The arithmetic-logical instructions add, addi, sub, and, andi, or, and slt
- The jumping instructions branch equal (beq), branch not equal (bne), and jump (j)

Use Figure 1 as a top-level block diagram of your single cycle implementation and Figure 2 for the pipelined structure. Note: there may be some components and control signals omitted from the figures that you will have to add to support all instructions listed above. Forwarding should be implemented in the pipelined structure to handle data and control hazards. (source: Computer Organization and Design, by Patterson and Hennessy, Morgan Kaufmann Publishers).

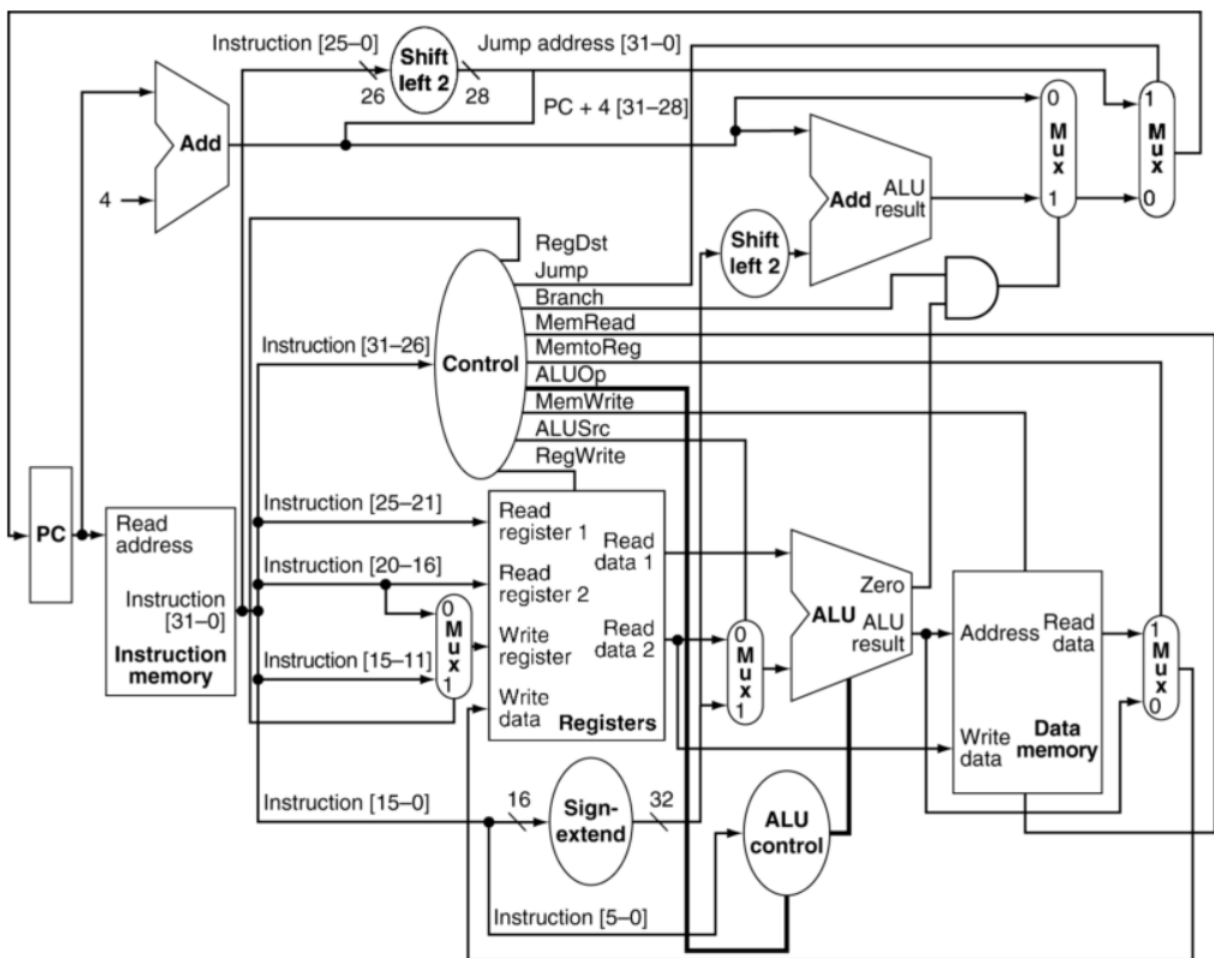


Figure 1: Single cycle implementation of MIPS architecture

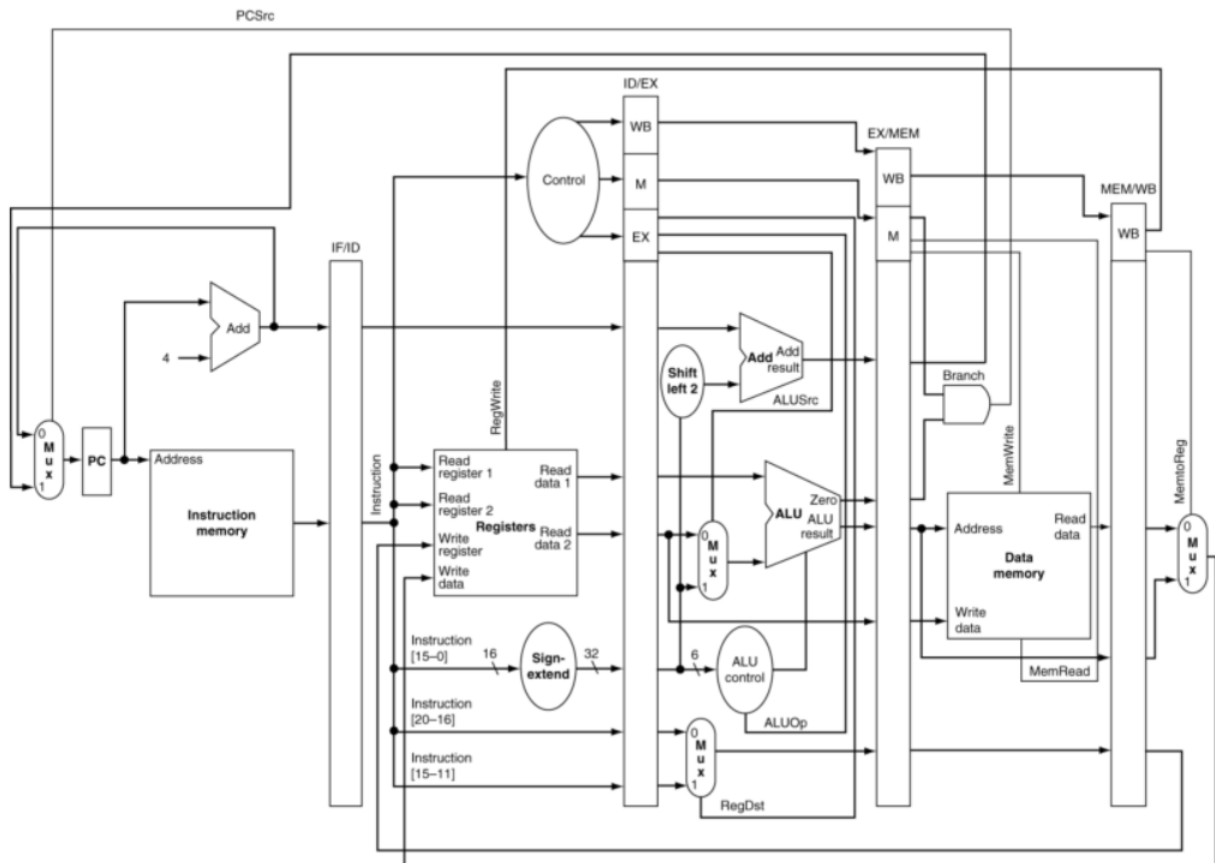


Figure 2: Pipelined implementation of MIPS architecture

## 2 Single-cycle

### 2.1 Verilog Code Realisation

The Verilog code with comments is given in the appendix. We divided the CPU into components of:

1. **Adder:** to add two numbers A and B. And we use adder in single-cycle processor to get next address (PC+4).
2. **Sign extension:** to extend a 16-bit input to a 32-bit output so that we can calculate the target address or use it as another input for ALU.
3. **Register:** to have an access to Register File so that content can be read from registers or written into registers.
4. **ALU control:** to output the 4-bit ALU control signal so that ALU can do the corresponding operation. With given ALUOp and function code, ALU control can give the 4-bit ALU control signal.
5. **ALU:** to do the corresponding operation with two 32-bit inputs and a 4-bit ALU control signal.
6. **Data Memory:** to have an access to Data Memory so that the processor can store data into data memory or read data from data memory.

7. **Instruction Memory:** to store instructions and with an input PC address, it can output the corresponding instruction.
8. **Control:** with the input instruction code, it can give the corresponding values of control signals that are needed in future stages (like MemRead, MemWrite, etc).
9. **Program Counter:** to have an access to PC register so that it can output the correct PC address in IF stage.
10. **N bit MUX:** to help choose the correct data depending on the control signal (used for branch instructions or choose rt/rd).

The main function “single\_cycle” wired all components according to Figure 1 and realized the whole single cycle process.

## 2.2 Simulation Result and Discussion

For simulation, we use Vivado to write a simulation file “test\_single” which is shown below:

```

1  `timescale 1ns / 1ps
2
3  `include "single_cycle.v"
4
5  module single_cycle_tb;
6
7      integer i = 0;
8
9      // Inputs
10     reg clk;
11
12     // Instantiate the Unit Under Test (UUT)
13     single_cycle uut (
14         .clk(clk)
15     );
16
17     initial begin
18         // Initialize Inputs
19         clk = 0;
20         $dumpfile("single_cycle.vcd");
21         $dumpvars(1, uut);
22         $display("Textual result of single cycle:");
23         $display("=====");
24         #750;
25         $stop;
26     end
27
28     always #10 begin
29         $display("Time: %d, CLK = %d, PC = 0x%H", i, clk, uut.PC_out);
30         $display("[ $s0 ] = 0x%H, [ $s1 ] = 0x%H, [ $s2 ] = 0x%H", uut.get_reg.
31             register_memory[16],
32             uut.get_reg.register_memory[17],
33             uut.get_reg.register_memory[18]);
34         $display("[ $s3 ] = 0x%H, [ $s4 ] = 0x%H, [ $s5 ] = 0x%H", uut.get_reg.
35             register_memory[19], uut.get_reg.register_memory[20], uut.get_reg.
36             register_memory[21]);

```

```

34    $display("[ $s6] = 0x%H, [ $s7] = 0x%H, [ $t0] = 0x%H", uut.get_reg.
    register_memory[22], uut.get_reg.register_memory[23], uut.get_reg.
    register_memory[8]);
35    $display("[ $t1] = 0x%H, [ $t2] = 0x%H, [ $t3] = 0x%H", uut.get_reg.
    register_memory[9], uut.get_reg.register_memory[10], uut.get_reg.
    register_memory[11]);
36    $display("[ $t4] = 0x%H, [ $t5] = 0x%H, [ $t6] = 0x%H", uut.get_reg.
    register_memory[12], uut.get_reg.register_memory[13], uut.get_reg.
    register_memory[14]);
37    $display("[ $t7] = 0x%H, [ $t8] = 0x%H, [ $t9] = 0x%H", uut.get_reg.
    register_memory[15], uut.get_reg.register_memory[24], uut.get_reg.
    register_memory[25]);
38    clk = ~clk;
39    if (~clk) begin
40        i = i + 1;
41        $display("=====");
42    end
43 end
44 endmodule

```

The following is our simulation result for the single-cycle processor in Vivado. And from simulation, we can see that the single cycle processor successfully completed the instructions.

#### Simulation Result:

```

1 Textual result of single cycle:
2 =====
3 Time:          0, CLK = 0, PC = 0xffffffffc
4 [ $s0] = 0x00000000, [ $s1] = 0x00000000, [ $s2] = 0x00000000
5 [ $s3] = 0x00000000, [ $s4] = 0x00000000, [ $s5] = 0x00000000
6 [ $s6] = 0x00000000, [ $s7] = 0x00000000, [ $t0] = 0x00000000
7 [ $t1] = 0x00000000, [ $t2] = 0x00000000, [ $t3] = 0x00000000
8 [ $t4] = 0x00000000, [ $t5] = 0x00000000, [ $t6] = 0x00000000
9 [ $t7] = 0x00000000, [ $t8] = 0x00000000, [ $t9] = 0x00000000
10 Time:         0, CLK = 1, PC = 0x00000000
11 [ $s0] = 0x00000000, [ $s1] = 0x00000000, [ $s2] = 0x00000000
12 [ $s3] = 0x00000000, [ $s4] = 0x00000000, [ $s5] = 0x00000000
13 [ $s6] = 0x00000000, [ $s7] = 0x00000000, [ $t0] = 0x00000000
14 [ $t1] = 0x00000000, [ $t2] = 0x00000000, [ $t3] = 0x00000000
15 [ $t4] = 0x00000000, [ $t5] = 0x00000000, [ $t6] = 0x00000000
16 [ $t7] = 0x00000000, [ $t8] = 0x00000000, [ $t9] = 0x00000000
17 =====
18 Time:         1, CLK = 0, PC = 0x00000000
19 [ $s0] = 0x00000000, [ $s1] = 0x00000000, [ $s2] = 0x00000000
20 [ $s3] = 0x00000000, [ $s4] = 0x00000000, [ $s5] = 0x00000000
21 [ $s6] = 0x00000000, [ $s7] = 0x00000000, [ $t0] = 0x00000020
22 [ $t1] = 0x00000000, [ $t2] = 0x00000000, [ $t3] = 0x00000000
23 [ $t4] = 0x00000000, [ $t5] = 0x00000000, [ $t6] = 0x00000000
24 [ $t7] = 0x00000000, [ $t8] = 0x00000000, [ $t9] = 0x00000000
25 Time:         1, CLK = 1, PC = 0x00000004
26 [ $s0] = 0x00000000, [ $s1] = 0x00000000, [ $s2] = 0x00000000
27 [ $s3] = 0x00000000, [ $s4] = 0x00000000, [ $s5] = 0x00000000
28 [ $s6] = 0x00000000, [ $s7] = 0x00000000, [ $t0] = 0x00000020
29 [ $t1] = 0x00000000, [ $t2] = 0x00000000, [ $t3] = 0x00000000
30 [ $t4] = 0x00000000, [ $t5] = 0x00000000, [ $t6] = 0x00000000
31 [ $t7] = 0x00000000, [ $t8] = 0x00000000, [ $t9] = 0x00000000
32 =====
33 Time:         2, CLK = 0, PC = 0x00000004
34 [ $s0] = 0x00000000, [ $s1] = 0x00000000, [ $s2] = 0x00000000
35 [ $s3] = 0x00000000, [ $s4] = 0x00000000, [ $s5] = 0x00000000

```

```

36 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
37 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
38 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
39 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
40 Time:          2, CLK = 1, PC = 0x00000008
41 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
42 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
43 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
44 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
45 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
46 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
47 =====
48 Time:          3, CLK = 0, PC = 0x00000008
49 [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
50 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
51 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
52 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
53 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
54 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
55 Time:          3, CLK = 1, PC = 0x0000000c
56 [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
57 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
58 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
59 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
60 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
61 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
62 =====
63 Time:          4, CLK = 0, PC = 0x0000000c
64 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
65 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
66 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
67 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
68 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
69 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
70 Time:          4, CLK = 1, PC = 0x00000010
71 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
72 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
73 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
74 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
75 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
76 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
77 =====
78 Time:          5, CLK = 0, PC = 0x00000010
79 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
80 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
81 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
82 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
83 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
84 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
85 Time:          5, CLK = 1, PC = 0x00000014
86 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
87 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
88 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
89 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
90 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
91 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
92 =====
93 Time:          6, CLK = 0, PC = 0x00000014

```

```

94 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
95 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
96 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
97 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
98 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
99 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
100 Time:          6, CLK = 1, PC = 0x00000018
101 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
102 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
103 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
104 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
105 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
106 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
107 =====
108 Time:          7, CLK = 0, PC = 0x00000018
109 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
110 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
111 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
112 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
113 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
114 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
115 Time:          7, CLK = 1, PC = 0x0000001c
116 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
117 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
118 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
119 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
120 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
121 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
122 =====
123 Time:          8, CLK = 0, PC = 0x0000001c
124 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff
125 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
126 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
127 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
128 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
129 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
130 Time:          8, CLK = 1, PC = 0x00000020
131 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff
132 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
133 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
134 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
135 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
136 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
137 =====
138 Time:          9, CLK = 0, PC = 0x00000020
139 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff
140 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
141 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
142 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
143 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
144 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
145 Time:          9, CLK = 1, PC = 0x00000024
146 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff
147 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
148 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
149 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
150 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
151 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000

```

```

152 =====
153 Time:          10, CLK = 0, PC = 0x00000024
154 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
155 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
156 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
157 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
158 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
159 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
160 Time:          10, CLK = 1, PC = 0x00000028
161 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
162 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
163 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
164 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
165 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
166 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
167 =====
168 Time:          11, CLK = 0, PC = 0x00000028
169 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
170 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
171 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
172 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
173 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
174 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
175 Time:          11, CLK = 1, PC = 0x0000002c
176 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
177 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
178 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
179 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
180 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
181 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
182 =====
183 Time:          12, CLK = 0, PC = 0x0000002c
184 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
185 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
186 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
187 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
188 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
189 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
190 Time:          12, CLK = 1, PC = 0x00000030
191 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
192 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
193 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
194 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
195 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
196 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
197 =====
198 Time:          13, CLK = 0, PC = 0x00000030
199 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9
200 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
201 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
202 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
203 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
204 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
205 Time:          13, CLK = 1, PC = 0x00000034
206 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9
207 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
208 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
209 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```



[illegible]

```

268 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
269 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
270 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
271 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
272 =====
273 Time:          18, CLK = 0, PC = 0x00000044
274 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
275 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
276 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
277 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
278 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
279 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
280 Time:          18, CLK = 1, PC = 0x00000048
281 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
282 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
283 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
284 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
285 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
286 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
287 =====
288 Time:          19, CLK = 0, PC = 0x00000048
289 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
290 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
291 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
292 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
293 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
294 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
295 Time:          19, CLK = 1, PC = 0x0000004c
296 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
297 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
298 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
299 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
300 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
301 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
302 =====
303 Time:          20, CLK = 0, PC = 0x0000004c
304 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
305 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
306 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
307 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
308 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
309 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
310 Time:          20, CLK = 1, PC = 0x00000050
311 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
312 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
313 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
314 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
315 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
316 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
317 =====
318 Time:          21, CLK = 0, PC = 0x00000050
319 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
320 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
321 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
322 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
323 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
324 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
325 Time:          21, CLK = 1, PC = 0x00000054

```

```

326 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
327 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
328 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
329 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
330 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
331 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
332 =====
333 Time:          22, CLK = 0, PC = 0x00000054
334 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
335 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
336 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
337 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
338 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
339 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
340 Time:          22, CLK = 1, PC = 0x00000058
341 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
342 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
343 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
344 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
345 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
346 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
347 =====
348 Time:          23, CLK = 0, PC = 0x00000058
349 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
350 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
351 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
352 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
353 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
354 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
355 Time:          23, CLK = 1, PC = 0x0000005c
356 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
357 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
358 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
359 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
360 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
361 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
362 =====
363 Time:          24, CLK = 0, PC = 0x0000005c
364 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
365 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
366 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
367 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
368 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
369 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
370 Time:          24, CLK = 1, PC = 0x00000060
371 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
372 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
373 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
374 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
375 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
376 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
377 =====
378 Time:          25, CLK = 0, PC = 0x00000060
379 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
380 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
381 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
382 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
383 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000

```

[illegible]



```

500 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
501 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
502 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
503 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
504 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
505 Time:          33, CLK = 1, PC = 0x00000068
506 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
507 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
508 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
509 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
510 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
511 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
512 =====
513 Time:          34, CLK = 0, PC = 0x00000068
514 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
515 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
516 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
517 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
518 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
519 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
520 Time:          34, CLK = 1, PC = 0x0000006c
521 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
522 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
523 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
524 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
525 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
526 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
527 =====
528 Time:          35, CLK = 0, PC = 0x0000006c
529 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
530 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
531 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
532 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
533 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
534 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
535 Time:          35, CLK = 1, PC = 0x000000ac
536 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
537 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
538 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
539 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
540 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
541 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
542 =====
543 Time:          36, CLK = 0, PC = 0x000000ac
544 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
545 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
546 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
547 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
548 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
549 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
550 Time:          36, CLK = 1, PC = 0x000000b0
551 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
552 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
553 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
554 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
555 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
556 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
557 =====

```

## 3 Pipelined

### 3.1 Verilog Code Realisation

The Verilog code with comments is given in the appendix. For Pipelined Processor, the basic components are the same as the components for single-cycle process:

1. **Adder:** to add two numbers A and B. And we use adder in single-cycle processor to get next address (PC+4).
2. **Sign extension:** to extend a 16-bit input to a 32-bit output so that we can calculate the target address or use it as another input for ALU.
3. **Register:** to have an access to Register File so that content can be read from registers or written into registers.  
**There is a change for pipeline processor: since we need to display the content in SSD, thus we add an extra input and output to obtain the content in corresponding register represented by the switched on FPGA board.**
4. **ALU control:** to output the 4-bit ALU control signal so that ALU can do the corresponding operation. With given ALUOp and function code, ALU control can give the 4-bit ALU control signal.
5. **ALU:** to do the corresponding operation with two 32-bit inputs and a 4-bit ALU control signal.
6. **Data Memory:** to have an access to Data Memory so that the processor can store data into data memory or read data from data memory.
7. **Instruction Memory:** to store instructions and with an input PC address, it can output the corresponding instruction.
8. **Control:** with the input instruction code, it can give the corresponding values of control signals that are needed in future stages (like MemRead, MemWrite, etc).
9. **Program Counter:** to have an access to PC register so that it can output the correct PC address in IF stage.
10. **N bit MUX:** to help choose the correct data depending on the control signal (used for branch instructions or choose rt/rd).

And we add some additional components to deal with some data/control hazard:

1. **Forwarding:** to forward the correct data for operation in case of WAR (write after read) so that some cases of data hazard will be solved without adding bubbles. By comparison of rs, rt and rd in (ID for beq/bne) EX, MEM, WB stages, it will output a control signal for forward mux to choose correct data.
2. **Forward Mux:** to choose the correct data among three data with the control signal output from forwarding unit.
3. **Hazard Detection:** to detect control/data hazard in ID stage. And if hazard appears, a bubble will be inserted into ID stage.

4. **Pipe write:** to push the data/ control signals in one stage to the next future stage. (And for IF/ID register, a control signal IF\_write or IF\_Flush will control the component, if one of the two signal is zero, then the data pushed into ID stage will be all zero.)
5. **ID\_EX Pipeline:** similar function as pipe\_write, just to push all needed data/signal into next EX stage.

And to realize SSD drive, there are also some extra components added in the pipeline processor:

1. **Clock Divider:** to divide the input clock (from FPGA board) into a lower frequency clock. And the output clock will be used for SSD display.
2. **Ring Counter:** to change the digit on SSD periodically.

The main function “pipeline” wired all components according to Figure 2 and realized the whole process with given instructions. The RTL Implementation is shown as below:

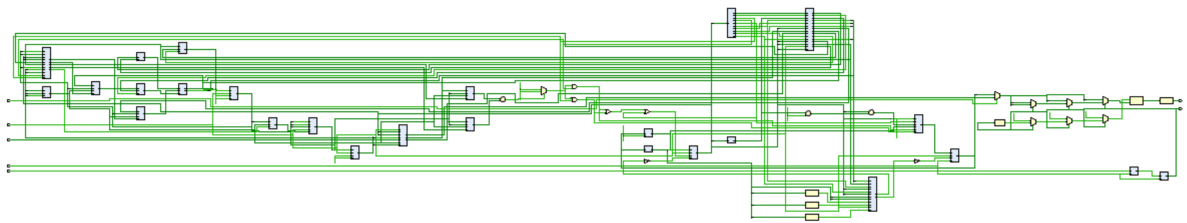


Figure 3: Pipelined RTL Implementation

## 3.2 Control and Data Hazard Detection and Solution

As data/control hazard is a critical problem for pipeline processor, in this part, we will explain our code for control/data hazard in detail.

In pipeline processors, there exist some hazard due to the adjacent instructions which may cause wrong outputs, such as (lw \$t0, 0(\$t1), and \$s0, \$t0, \$t2). To avoid such cases, the pipelined process needs a hazard detection in **ID** stage.

Our group write a hazard detection unit including both data and control hazard detection. (Note: the rule for hazard detection in our program is a little different from the method shown in the slide: if a hazard is detected, then we will insert bubble in ID stage instead of in EX stage.)

### 3.2.1 Hazard Detection

Pipeline\_HazardDetection.v

```

1 module HazardDtection (
2     //input IF_instruction ,
3     input  clk ,
4     input  bne ,
5     input  branch ,
6     input  jump ,

```



```

7  input ID_MemRead ,
8  input ID_RegWrite ,
9  input EX_MemRead ,
10 input [4:0] EX_Rt ,
11 input [4:0] ID_Rt ,
12 input [4:0] ID_Rd ,
13 input [4:0] IF_Rs ,
14 input [4:0] IF_Rt ,
15 output IF_Write ,
16 output PC_Write
17 );
18
19 reg PCtem ;
20     reg IFIDtem ;
21
22     initial begin
23         PCtem<=1;
24         IFIDtem <=1;
25     end

```

Output **PC\_Write** and **IF\_Write** will control the PC Register and IF/IF pipeline register. If **PC\_Write=0**, the PC address will be hold, or it will load next PC address.

And if **IF\_Write=1**, the IF/ID pipeline register will push data in IF stage into ID stage, otherwise, the instruction pushed into ID stage is 0x00000000, which indicates a bubble inserted into ID stage. (PC\_Write and IF\_Write always happen at the same time.)

1. Case 1: lw in ID stage, and destination register of lw is also the source register for instruction in IF stage. (e.g.:lw \$t0, 0(\$t1), add \$s0, \$t0, \$t2 )

```

1      always@(posedge clk)
2      begin
3          if (ID_MemRead && ((ID_Rt==IF_Rs) | (ID_Rt==IF_Rt))&&(branch
==0)&&(jump==0)&&(bne==0))
4              begin
5                  PCtem<=0;
6                  IFIDtem <=0;
7              end

```

2. Case 2: beq or bne in IF stage, and the destination register of lw instruction in EX stage is also a source register for beq/bne in ID stage. (e.g.:add \$t0, \$t1; \$t2, addi \$t3, \$t3, 0; beq \$t0, \$t1, EXIT)

Since beq/bne should be solved in ID stage, and the needed register (MEM stage now)is still not be overwritten.

```

1      else if ((bne || branch) && EX_MemRead &&((EX_Rt==IF_Rs) | (
EX_Rt==IF_Rt))&&(jump==0))
2          begin
3              PCtem<=0;
4              IFIDtem <=0;
5          end

```

3. Case 3: beq or bne in IF stage, and the destination register of instruction in ID stage is also a source register for beq/bne in ID stage. (e.g.:add \$t0, \$t1; beq \$t0, \$t1, EXIT)

And for I-type instruction, destination register is ID\_rt and for R-type, destination register is ID\_rd.

```

1      //I/lw and adjacent beq
2      else if ((bne || branch) && ID_RegWrite && ((ID_Rt==IF_Rs && IF_Rs
   !=0) | (ID_Rt==IF_Rt && IF_Rt !=0)) && (jump==0))
3          begin
4              PCtem <=0;
5              IFIDtem <=0;
6          end
7      // r-type and adjacent beq
8      else if ((bne || branch) && ID_RegWrite && ((ID_Rd==IF_Rs && IF_Rs
   !=0) | (ID_Rd==IF_Rt && IF_Rt !=0)) && (jump==0))
9          begin
10             PCtem <=0;
11             IFIDtem <=0;
12         end
13     else

```

4. Case 4: Without any hazard, then PC\_Write = 1 and IF\_Write = 1.

```

1      begin
2          PCtem <=1;
3          IFIDtem <=1;
4      end
5      end
6      assign IF_Write=IFIDtem;
7      assign PC_Write=PCtem;
8  endmodule

```

### 3.2.2 Hazard Solution: Forwarding

To solve some cases of data hazard, we can use forwarding unit. Thus, these cases of data hazard are not included in hazard detection part. Pipeline\_forwarding.v

```

1      module forwarding (
2          input beq ,
3          input bne ,
4          input [4:0] ID_rs ,
5          input [4:0] ID_rt ,
6          input EMRW,
7          input MWRW,
8          input MEM_MemWrite,
9          input [4:0] EM_Rd,
10         input [4:0] IE_Rs ,
11         input [4:0] IE_Rt ,
12         input [4:0] MW_Rd,
13         output [1:0] beqA ,
14         output [1:0] beqB ,
15         output [1:0] ForwardA ,
16         output [1:0] ForwardB ,
17         output sw_src
18     );
19
20
21     reg [1:0] tmpA, tmp_beqA ;
22     reg [1:0] tmpB, tmp_beqB ;
23     reg tmp_sw_src ;
24     initial begin
25         tmp_beqA <=0;
26         tmp_beqB <=0;

```

```

27 tmpA<=0;
28 tmpB<=0;
29 temp_sw_src <=0;
30 end

```

ForwardA and ForwardB are the control signals for the muxes to choose inputs of ALU in EX stage. beqA and beqB are the control signals of the muxes to choose data for comparison. And sw\_src is the control signal to choose the store data in Data Memory for instruction sw.

1. Case 1: Content in destination register for instruction in EX/MEM stage is a source register for beq/bne. (e.g: add \$t0, \$t1, \$t2; beq \$s0, \$t0, EXIT).

```

1 always@ (*)
2 begin
3 if (EMRW && (EM_Rd!=0) && (EM_Rd==ID_rs) && (bne || beq))
4     begin
5         tmp_beqA <=2'b10;
6     end
7 else if ((bne || beq) && (MWRW && (MW_Rd!=0) && (MW_Rd==ID_rs)) && ~(EMRW
&& (EM_Rd!=0) && (EM_Rd==IE_Rs)))
8     begin
9         tmp_beqA <=2'b01;
10    end
11 else
12     begin
13         tmp_beqA <=2'b00;
14     end
15 if (EMRW && (EM_Rd!=0) && (EM_Rd==ID_rt) && (bne || beq))
16     begin
17         tmp_beqB <=2'b10;
18     end
19 else if ((bne || beq) && (MWRW && (MW_Rd!=0) && (MW_Rd==ID_rt)) && ~(
EMRW && (EM_Rd!=0) && (EM_Rd==IE_Rs)))
20     begin
21         tmp_beqB <=2'b01;
22     end
23 else
24     begin
25         tmp_beqB <=2'b00;
26     end

```

2. Case 2: Due to the different condition for readReg and writeReg, it may be possible that WAR (write after read) will happen, which will cause an incorrect data to be stored in Data Memory. And to avoid such case, we need to use a signal to select the correct data for sw.

```

1 if (MEM_MemWrite && MWRW && (EM_Rd==MW_Rd))
2     begin
3         temp_sw_src <=1'b1;
4     end
5 else
6     begin
7         temp_sw_src <=1'b0;
8     end

```

3. Case 3: Content in destination register for instruction in MEM/WB stage is input for ALU (EX stage). (e.g: add \$t0, \$t1, \$t2; sub \$s0, \$t0, \$t1).

```

1  if (EMRW && (EM_Rd!=0) && (EM_Rd==IE_Rs))
2      begin
3          tmpA<=2'b10;
4      end
5  else if ((MWRW && (MW_Rd!=0) && (MW_Rd==IE_Rs)) && ~(EMRW && (EM_Rd
6      !=0) && (EM_Rd==IE_Rs)))
7      begin
8          tmpA<=2'b01;
9      end
10 else
11     begin
12         tmpA<=2'b00;
13     end
14 if (EMRW && (EM_Rd!=0) && (EM_Rd==IE_Rt))
15     begin
16         tmpB<=2'b10;
17     end
18 else if ((MWRW && (MW_Rd!=0) && (MW_Rd==IE_Rt)) && ~(EMRW && (EM_Rd
19     !=0) && (EM_Rd==IE_Rt)))
20     begin
21         tmpB<=2'b01;
22     end
23 else
24     begin
25         tmpB<=2'b00;
26     end
27 end
28 assign beqA=tmp_beqA;
29 assign beqB=tmp_beqB;
30 assign ForwardA=tmpA;
31 assign ForwardB=tmpB;
32 assign sw_src=temp_sw_src;
33 endmodule

```

### 3.3 Simulation Result and Discussion

We used Vivado to write a simulation file which is shown below:

```

1  `timescale 1ns / 1ps
2  module test_pipe;
3      parameter half_period = 1;
4      reg clk, reset=0;
5      pipeline pipeline1 (clk, reset);
6      initial begin
7          clk<=1'b0;
8      end
9
10     always
11     begin
12         #half_period clk = ~clk;
13         $display("Time: %d, CIK = %b, PC = 0x%h,", $time, clk, test_pipe.
14             pipeline1.PC_out);
15         $display("[ $s0 ] = 0x%h, [ $s1 ] = 0x%h, [ $s2 ] = 0x%h",
16             test_pipe.pipeline1.get_reg.register_memory[16],
17             test_pipe.pipeline1.get_reg.register_memory[17],
18             test_pipe.pipeline1.get_reg.register_memory[18]);

```

```

18     $display("[ $s3] = 0x%h, [ $s4] = 0x%h, [ $s5] = 0x%h", test_pipe .
19     pipeline1.get_reg.register_memory[19],
20     test_pipe.pipeline1.get_reg .
21     register_memory[20],
22     test_pipe.pipeline1.get_reg .
23     register_memory[21]);
24     $display("[ $s6] = 0x%h, [ $s7] = 0x%h, [ $t0] = 0x%h", test_pipe .
25     pipeline1.get_reg.register_memory[22],
26     test_pipe .
27     pipeline1.get_reg.register_memory[23],
28     test_pipe .
29     pipeline1.get_reg.register_memory[8]);
30     $display("[ $t1] = 0x%h, [ $t2] = 0x%h, [ $t3] = 0x%h", test_pipe .
31     pipeline1.get_reg.register_memory[9],
32     test_pipe.pipeline1.get_reg.register_memory[10],
33     test_pipe.pipeline1.get_reg.register_memory[11]);
34     $display("[ $t4] = 0x%h, [ $t5] = 0x%h, [ $t6] = 0x%h", test_pipe .
35     pipeline1.get_reg.register_memory[12],
36     test_pipe.pipeline1.get_reg.register_memory[13],
37     test_pipe.pipeline1.get_reg.register_memory[14]);
38     $display("[ $t7] = 0x%h, [ $t8] = 0x%h, [ $t9] = 0x%h", test_pipe .
39     pipeline1.get_reg.register_memory[15],
40     test_pipe.pipeline1.get_reg.register_memory[24],
41     test_pipe.pipeline1.get_reg.register_memory[25]);
42 if (~clk) $display("
=====");
43 end
44 initial begin
45     $display("=====");
46 );
47 #41 $stop;
48 end
49 endmodule

```

The following is the simulation result of the pipelined processor. We can see that the simulation result is the same as single-cycle processor, but takes much shorter time.

## 4 SSD Realization

For demonstration on FPGA board, we also write the code for SSD driver in the pipeline processor as a part in pipeline.v:

```

1 module pipeline(
2     clk, reset, C, AN, clock, reg_switch, PC_switch
3 );
4     input  clk, reset, clock, PC_switch;
5     input  [4:0] reg_switch;
6     output reg [6:0] C;
7     output [3:0] AN;
8     wire [31:0] Q2;
9     initial begin
10         C=7'b1111111;
11     end

```

```

12 // code for pipeline processor is omitted.
13
14 Registers get_reg(
15     .read_register_1 (ID_instruction[25:21]),
16     .read_register_2 (ID_instruction[20:16]),
17     .switch(reg_switch),
18     .regWrite(WB_RegWrite),
19     .write_register(WB_rd),
20     .write_data(write_data),
21     .read_data_1 (ID_read1),
22     .read_data_2 (ID_read2),
23     .clk(clk),
24     .Q2(Q2)
25 );
26 //SSD
27 wire clock_d;
28 reg [3:0] Q;
29
30 clock_divider clock1(clock_d, clock, reset);
31
32 ring_counter ring(AN, clock_d, reset);
33
34 wire [31:0] ssd_out;
35 // here only needs to use 4 hexo numbers because first 4 numbers must be
36 // 0 for the small instruction value
37 assign ssd_out = (PC_switch)? PC_out:Q2;
38
39 always @ (*) begin
40     if (AN == 4'b1110) Q <= ssd_out[3:0];
41     else if (AN == 4'b1101) Q <= ssd_out[7:4];
42     else if (AN == 4'b1011) Q <= ssd_out[11:8];
43     else if (AN == 4'b0111) Q <= ssd_out[15:12];
44 end
45 // ssd numbers
46 always @ (*) begin
47     case (Q)
48         4'h0: C = 7'b1000000;
49         4'h1: C = 7'b1111001;
50         4'h2: C = 7'b0100100;
51         4'h3: C = 7'b0110000;
52         4'h4: C = 7'b0011001;
53         4'h5: C = 7'b0010010;
54         4'h6: C = 7'b0000010;
55         4'h7: C = 7'b1111000;
56         4'h8: C = 7'b0000000;
57         4'h9: C = 7'b0010000;
58         4'ha: C = 7'b0001000;
59         4'hb: C = 7'b0000011;
60         4'hc: C = 7'b1000110;
61         4'hd: C = 7'b0100001;
62         4'he: C = 7'b0000110;
63         4'hf: C = 7'b0001110;
64         default: C = 7'b1111111;
65     endcase
66 end
67 endmodule

```

And the following lists the usage of every variable:

- **clk**: a clock control for the pipeline processor;
- **reset**: if reset = 1, then PC will be reset to zero, and pipeline processor begins at the first instruction;
- **C**: a 7-bit output to determine the number shown on SSD;
- **AN**: a 4-bit output to control whether digit to shown for SSD on FPGA board;
- **clock**: a clock from FPGA board for SSD display (and will be divided into a lower frequency);
- **reg\_switch**: a 5-bit input to determine which register to be displayed by SSD;
- **PC\_switch**: if PC\_switch = 1, then SSD will show the PC address, otherwise, it will display contents of registers;
- **Q2**: a 31-bit wire to save the content in the corresponding register represented by reg\_switch;
- **ssd\_out**: a 31-bit wire to save the content to be displayed by SSD (whether it saves the PC address or the content in register depending on the value of PC\_switch).

And in order to get the content in the register represented by reg\_switch, we change a bit in Registers.v and add an extra input (reg\_switch) and output (Q2) so that we can have an access to Register File.

## 5 Appendix

### 5.1 Single-cycle

single\_cycle.v

```

1  'ifndef MODULE_SINGLE_CYCLE
2  'define MODULE_SINGLE_CYCLE
3  'include "adder.v"
4  'include "ALU_control.v"
5  'include "ALU.v"
6  'include "control.v"
7  'include "data_memory.v"
8  'include "instruction_memory.v"
9  'include "program_counter.v"
10 'include "register.v"
11 'include "sign_extension.v"
12 'include "Mux_N_bit.v"
13
14 module single_cycle(clk, reset);
15     input clk;
16     input reset;
17     // Input PC Signals
18     wire [31:0] PC_in, PC_out, PC_plus_four;
19     // Instructions memory signals
20     wire [31:0] Instructions;
21     // Register signals
22     wire [31:0] write_data, reg_read_data_1, reg_read_data_2;
23     wire [4:0] write_address;

```

```

24 // ALU
25 wire [31:0] ALU_a,ALU_b,ALU_out;
26 wire [3:0] ALU_control;
27 wire output_zero;
28 assign ALU_a = reg_read_data_1;
29 // D-mem
30 wire [31:0] D_MEM_read_data;
31 // Control
32 wire RegDst, Jump, Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
33 wire [1:0] ALUOp;
34 wire [3:0] ALU_control_out;
35 // Branch
36 wire [31:0] extended_imm,shifted_imm,branch_result,branch_out;
37 wire branch_mux;
38 assign branch_mux = output_zero & Branch;
39 // Jump
40 wire [31:0] jump_addr;
41 // PC + 4[31:28]
42 assign jump_addr = {PC_plus_four[31:28],Instructions[25:0],2'b0};
43 assign branch_out = PC_plus_four + (extended_imm<<2);
44 // Connect PC
45 program_counter PC(
46     .PC_in(PC_in),
47     .PC_out(PC_out),
48     .reset(reset),
49     .clk(clk)
50 );
51 // Calculate PC + 4
52 adder add_PC_by_four(
53     .a(PC_out),
54     .b(32'b0100),
55     .sum(PC_plus_four)
56 );
57 // Get instruction
58 instruction_memory get_ins(
59     .address(PC_out),
60     .instruction(Instructions)
61 );
62 // Read Register
63 Registers get_reg(
64     .read_register_1(Instructions[25:21]),
65     .read_register_2(Instructions[20:16]),
66     .regWrite(RegWrite),
67     .write_register(write_address),
68     .write_data(write_data),
69     .read_data_1(reg_read_data_1),
70     .read_data_2(reg_read_data_2),
71     .clk(clk)
72 );
73 // ALU
74 ALU alu(
75     .ALUCtrl(ALU_control_out),
76     .a(ALU_a),
77     .b(ALU_b),
78     .ALU_result(ALU_out),
79     .zero(output_zero)
80 );
81 // Control

```



```

82     Control con(
83         .op_code(Instructions[31:26]),
84         .RegDst(RegDst),
85         .Branch(Branch),
86         .Jump(Jump),
87         .MemRead(MemRead),
88         .MemtoReg(MemtoReg),
89         .MemWrite(MemWrite),
90         .ALUSrc(ALUSrc),
91         .RegWrite(RegWrite),
92         .ALUOp(ALUOp)
93     );
94     // ALU control
95     ALU_control alu_ctrl(
96         .funct(Instructions[5:0]),
97         .ALUOp(ALUOp),
98         .ALUCtrl(ALU_control_out)
99     );
100    // Sign Extension
101    sign_extension sign_ext(
102        .shortInput(Instructions[15:0]),
103        .longOutput(extended_imm)
104    );
105    // Data memory
106    data_memory dm(
107        .clk(clk),
108        .MemRead(MemRead),
109        .MemWrite(MemWrite),
110        .address(ALU_out),
111        .write_data(reg_read_data_2),
112        .read_data(D_MEM_read_data)
113    );
114
115    // Mux
116    Mux_N_bit#(5) mux_select_write_reg(
117        .in1(Instructions[20:16]),
118        .in2(Instructions[15:11]),
119        .select(RegDst),
120        .out(write_address)
121    );
122    Mux_N_bit#(32) mux_select_ALU_in(
123        .in1(reg_read_data_2),
124        .in2(extended_imm),
125        .select(ALUSrc),
126        .out(ALU_b)
127    );
128    assign write_data = (MemtoReg == 1'b0) ? ALU_out : D_MEM_read_data;
129    Mux_N_bit#(32) mux_select_PC_one(
130        .in1(PC_plus_four),
131        .in2(branch_out),
132        .select(branch_mux),
133        .out(branch_result)
134    );
135    assign PC_in = (Jump == 1'b0) ? branch_result : jump_addr;
136 endmodule
137 `endif

```

adder.v

```

1 'ifndef MODULE_ADDER
2 'define MODULE_ADDER
3 'timescale 1ns / 1ps
4 module adder(
5     input [31:0] a,
6     input [31:0] b,
7     output [31:0] sum
8 );
9     reg [31:0] sum;
10    always @(a or b)
11        begin
12            sum = a + b;
13        end
14 endmodule
15 'endif

```

#### ALU\_control.v

```

1 'ifndef MODULE_ALU_CONTROL
2 'define MODULE_ALU_CONTROL
3 'timescale 1ns / 1ps
4 module ALU_control(
5     funct ,ALUOp, ALUCtrl
6 );
7     input [5:0] funct;
8     input [1:0] ALUOp;
9     output [3:0] ALUCtrl;
10    reg [3:0] ALUCtrl;
11    always @ (funct or ALUOp)
12        begin
13            case(ALUOp)
14                2'b00: assign ALUCtrl = 4'b0010;
15                2'b01: assign ALUCtrl = 4'b0110;
16                2'b10:
17                    begin
18                        if (funct == 6'b100000) assign ALUCtrl = 4'b0010;
19                        else if (funct == 6'b100010) assign ALUCtrl = 4'b0110;
20                        else if (funct == 6'b100100) assign ALUCtrl = 4'b0000;
21                        else if (funct == 6'b100101) assign ALUCtrl = 4'b0001;
22                        else if (funct == 6'b101010) assign ALUCtrl = 4'b0111;
23                    end
24                2'b11:
25                    assign ALUCtrl = 4'b0000;
26            endcase
27        end
28 endmodule
29 'endif

```

#### ALU.v

```

1 'ifndef MODULE_ALU
2 'define MODULE_ALU
3 'timescale 1ns / 1ps
4 module ALU(
5     ALUCtrl ,a ,b ,zero ,ALU_result
6 );
7     input [3:0] ALUCtrl;
8     input [31:0] a , b;
9     output zero;
10    output [31:0] ALU_result;

```

```

11 reg zero;
12 reg [31:0] ALU_result;
13 always @ (a or b or ALUCtrl)
14 begin
15     case (ALUCtrl)
16     4'b0000:
17         begin
18             assign ALU_result = a & b;
19             assign zero = (a & b == 0) ? 1:0;
20         end
21     4'b0001:
22         begin
23             assign ALU_result = a | b;
24             assign zero = (a | b == 0) ? 1:0;
25         end
26     4'b0010:
27         begin
28             assign ALU_result = a + b;
29             assign zero = (a + b == 0) ? 1:0;
30         end
31     4'b0110:
32         begin
33             assign ALU_result = a - b;
34             assign zero = (a == b) ? 1:0;
35         end
36     4'b0111:
37         begin
38             assign ALU_result = (a < b) ? 1:0;
39             assign zero = (a < b) ? 0:1;
40         end
41     default:
42         begin
43             assign ALU_result = a;
44             assign zero = (a == 0) ? 1:0;
45         end
46     endcase
47 end
48 endmodule
49 `endif

```

#### control.v

```

1 `ifndef MODULE_CONTROL
2 `define MODULE_CONTROL
3 `timescale 1ns / 1ps
4 module Control (
5     input      [5:0]    op_code ,
6     output reg  [1:0]    ALUOp,
7     output reg  RegDst ,
8     output reg  Jump ,
9     output reg  Branch ,
10    output reg  MemRead ,
11    output reg  MemtoReg ,
12    output reg  MemWrite ,
13    output reg  ALUSrc ,
14    output reg  RegWrite
15 );
16
17 initial begin

```

```

18     RegDst      = 1'b0;
19     Jump        = 1'b0;
20     Branch      = 1'b0;
21     MemRead     = 1'b0;
22     MemtoReg    = 1'b0;
23     MemWrite    = 1'b0;
24     ALUSrc      = 1'b0;
25     RegWrite    = 1'b0;
26     ALUOp       = 2'b00;
27 end
28
29 always @ ( op_code ) begin
30     case ( op_code )
31         6'b000000: begin // R-type
32             RegDst      <= 1'b1;
33             Jump        <= 1'b0;
34             Branch      <= 1'b0;
35             MemRead     <= 1'b0;
36             MemtoReg    <= 1'b0;
37             MemWrite    <= 1'b0;
38             ALUSrc      <= 1'b0;
39             RegWrite    <= 1'b1;
40             ALUOp       <= 2'b10;
41         end
42         6'b000010: begin // j
43             RegDst      <= 1'b1;
44             Jump        <= 1'b1;
45             Branch      <= 1'b0;
46             MemRead     <= 1'b0;
47             MemtoReg    <= 1'b0;
48             MemWrite    <= 1'b0;
49             ALUSrc      <= 1'b0;
50             RegWrite    <= 1'b0;
51             ALUOp       <= 2'b10;
52         end
53         6'b000100: begin // beq
54             RegDst      <= 1'b1;
55             Jump        <= 1'b0;
56             Branch      <= 1'b1;
57             MemRead     <= 1'b0;
58             MemtoReg    <= 1'b0;
59             MemWrite    <= 1'b0;
60             ALUSrc      <= 1'b0;
61             RegWrite    <= 1'b0;
62             ALUOp       <= 2'b01;
63         end
64         6'b000100: begin // bne
65             RegDst      <= 1'b1;
66             Jump        <= 1'b0;
67             Branch      <= 1'b0;
68             MemRead     <= 1'b0;
69             MemtoReg    <= 1'b0;
70             MemWrite    <= 1'b0;
71             ALUSrc      <= 1'b0;
72             RegWrite    <= 1'b0;
73             ALUOp       <= 2'b01;
74         end
75         6'b001000: begin // addi

```

```

76         RegDst      <= 1'b0;
77         Jump        <= 1'b0;
78         Branch      <= 1'b0;
79         MemRead     <= 1'b0;
80         MemtoReg    <= 1'b0;
81         MemWrite    <= 1'b0;
82         ALUSrc      <= 1'b1;
83         RegWrite    <= 1'b1;
84         ALUOp       <= 2'b00;
85     end
86     6'b001100: begin // andi
87         RegDst      <= 1'b0;
88         Jump        <= 1'b0;
89         Branch      <= 1'b0;
90         MemRead     <= 1'b0;
91         MemtoReg    <= 1'b0;
92         MemWrite    <= 1'b0;
93         ALUSrc      <= 1'b1;
94         RegWrite    <= 1'b1;
95         ALUOp       <= 2'b11;
96     end
97     6'b100011: begin // lw
98         RegDst      <= 1'b0;
99         Jump        <= 1'b0;
100        Branch      <= 1'b0;
101        MemRead     <= 1'b1;
102        MemtoReg    <= 1'b1;
103        MemWrite    <= 1'b0;
104        ALUSrc      <= 1'b1;
105        RegWrite    <= 1'b1;
106        ALUOp       <= 2'b00;
107    end
108    6'b101011: begin // sw
109        RegDst      <= 1'b0;
110        Jump        <= 1'b0;
111        Branch      <= 1'b0;
112        MemRead     <= 1'b0;
113        MemtoReg    <= 1'b0;
114        MemWrite    <= 1'b1;
115        ALUSrc      <= 1'b1;
116        RegWrite    <= 1'b0;
117        ALUOp       <= 2'b00;
118    end
119
120        default: ;
121    endcase
122 end
123
124 endmodule
125 'endif

```

data\_memory.v

```

1 'ifndef MODULE_DATA_MEMORY
2 'define MODULE_DATA_MEMORY
3 'timescale 1ns / 1ps
4 module data_memory (
5     input          clk ,
6     input          MemRead,

```

```

7           MemWrite ,
8   input    [31:0]  address ,
9           write_data ,
10  output   [31:0]  read_data
11 );
12  parameter      size = 64; // size of data register_memory
13  integer         i;
14  wire           [31:0]  index;
15  reg            [31:0]  register_memory [0:size-1];
16
17  assign index = address >> 2; // address/4
18  initial begin
19      for (i = 0; i < size; i = i + 1)
20          register_memory[i] = 32'b0;
21      // read_data = 32'b0;
22  end
23  always @ ( posedge clk ) begin
24      if (MemWrite == 1'b1) begin
25          register_memory[index] = write_data;
26      end
27  end
28 endmodule
29 `endif

```

#### instruction\_memory.v

```

1  `ifndef MODULE_INSTRUCTION_MEMORY
2  `define MODULE_INSTRUCTION_MEMORY
3  module instruction_memory (
4      input    [31:0]  address ,
5      output   [31:0]  instruction
6  );
7      parameter size = 128;
8      // initialize memory
9      reg [31:0] memory [0:size-1];
10     // clear all memory to nop
11     initial begin
12         for (i = 0; i < size; i = i + 1)
13             memory[i] = 32'b0;
14         // include the instruction_memory
15         `include "InstructionMem_for_P2_Demo_updated.txt"
16     end
17     // Output the memory at address
18     assign instruction = memory[address >> 2];
19 endmodule
20 `endif

```

#### program\_program\_counter.v

```

1  `ifndef MODULE_PC
2  `define MODULE_PC
3  `timescale 1ns / 1ps
4  module program_counter(
5      PC_in , PC_out , reset , clk
6  );
7      input [31:0] PC_in;
8      output [31:0] PC_out;
9      input reset , clk;
10     reg [31:0] PC_out;
11     initial begin

```

```

12     PC_out = -4;
13 end
14 always @(posedge clk)
15 begin
16     if(reset)
17         begin
18             PC_out <= 32'b0;
19         end
20     else PC_out <= PC_in[31:0];
21 end
22 endmodule
23 'endif

```

#### register.v

```

1 'ifndef MODULE_REGISTERS
2 'define MODULE_REGISTERS
3 'timescale 1ns / 1ps
4
5 module Registers (
6     input      clk , regWrite ,
7     input      [4:0]  read_register_1 ,  read_register_2 ,
8     input      [4:0]  write_register ,
9     input      [31:0] write_data ,
10    output     [31:0]  read_data_1 , read_data_2
11 );
12     parameter size = 32;           // 32-bit CPU, $0 - $31
13     reg [31:0] register_memory [0:size-1];
14     integer i;
15
16     initial begin
17         for (i = 0; i < size; i = i + 1)
18             register_memory[i] = 32'b0;
19     end
20     assign read_data_1 = register_memory[read_register_1];
21     assign read_data_2 = register_memory[read_register_2];
22     always @(negedge clk) begin
23         if (regWrite == 1)
24             register_memory[write_register] <= write_data;
25     end
26 endmodule // registers
27 'endif

```

#### sign\_extension.v

```

1 'ifndef MODULE_SIGN_EXT
2 'define MODULE_SIGN_EXT
3 'timescale 1ns / 1ps
4 module sign_extension(
5     shortInput , longOutput
6 );
7     input [15:0] shortInput;
8     output [31:0] longOutput;
9     assign longOutput[15:0] = shortInput[15:0];
10    assign longOutput[31:16] = shortInput[15]?16'b1111_1111_1111_1111:16'b0
11    ;
12 endmodule
13 'endif

```

#### Mux\_N\_bit.v

```

1 'ifndef MODULE_Mux
2 'define MODULE_Mux
3 module Mux_N_bit(in1 ,in2 ,out ,select);
4     parameter N = 32;
5     input [N-1:0] in1 ,in2;
6     input select;
7     output [N-1:0] out;
8     assign out = select?in2:in1;
9 endmodule
10 'endif

```

### Simulation Result:

```

1 Textual result of single cycle:
2 =====
3 Time:          0, CLK = 0, PC = 0xffffffffc
4 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
5 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
6 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
7 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
8 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
9 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
10 Time:          0, CLK = 1, PC = 0x00000000
11 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
12 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
13 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
14 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
15 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
16 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
17 =====
18 Time:          1, CLK = 0, PC = 0x00000000
19 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
20 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
21 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
22 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
23 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
24 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
25 Time:          1, CLK = 1, PC = 0x00000004
26 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
27 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
28 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
29 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
30 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
31 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
32 =====
33 Time:          2, CLK = 0, PC = 0x00000004
34 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
35 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
36 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
37 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
38 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
39 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
40 Time:          2, CLK = 1, PC = 0x00000008
41 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
42 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
43 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
44 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
45 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
46 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000

```



```

47 =====
48 Time:          3, CLK = 0, PC = 0x00000008
49 [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
50 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
51 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
52 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
53 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
54 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
55 Time:          3, CLK = 1, PC = 0x0000000c
56 [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
57 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
58 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
59 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
60 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
61 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
62 =====
63 Time:          4, CLK = 0, PC = 0x0000000c
64 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
65 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
66 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
67 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
68 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
69 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
70 Time:          4, CLK = 1, PC = 0x00000010
71 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
72 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
73 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
74 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
75 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
76 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
77 =====
78 Time:          5, CLK = 0, PC = 0x00000010
79 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
80 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
81 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
82 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
83 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
84 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
85 Time:          5, CLK = 1, PC = 0x00000014
86 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
87 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
88 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
89 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
90 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
91 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
92 =====
93 Time:          6, CLK = 0, PC = 0x00000014
94 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
95 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
96 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
97 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
98 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
99 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
100 Time:          6, CLK = 1, PC = 0x00000018
101 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
102 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
103 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
104 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```

[illegible]

```

163 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
164 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
165 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
166 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
167 =====
168 Time:          11, CLK = 0, PC = 0x00000028
169 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff9
170 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
171 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
172 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
173 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
174 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
175 Time:          11, CLK = 1, PC = 0x0000002c
176 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff9
177 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
178 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
179 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
180 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
181 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
182 =====
183 Time:          12, CLK = 0, PC = 0x0000002c
184 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff9
185 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
186 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
187 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
188 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
189 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
190 Time:          12, CLK = 1, PC = 0x00000030
191 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff9
192 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
193 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
194 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
195 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
196 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
197 =====
198 Time:          13, CLK = 0, PC = 0x00000030
199 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffff9
200 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
201 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
202 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
203 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
204 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
205 Time:          13, CLK = 1, PC = 0x00000034
206 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffff9
207 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
208 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
209 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
210 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
211 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
212 =====
213 Time:          14, CLK = 0, PC = 0x00000034
214 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
215 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
216 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
217 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
218 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
219 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
220 Time:          14, CLK = 1, PC = 0x00000038

```

[illegible]

```

279 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
280 Time:          18, CLK = 1, PC = 0x00000048
281 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
282 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
283 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
284 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
285 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
286 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
287 =====
288 Time:          19, CLK = 0, PC = 0x00000048
289 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
290 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
291 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
292 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
293 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
294 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
295 Time:          19, CLK = 1, PC = 0x0000004c
296 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
297 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
298 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
299 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
300 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
301 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
302 =====
303 Time:          20, CLK = 0, PC = 0x0000004c
304 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
305 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
306 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
307 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
308 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
309 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
310 Time:          20, CLK = 1, PC = 0x00000050
311 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
312 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
313 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
314 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
315 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
316 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
317 =====
318 Time:          21, CLK = 0, PC = 0x00000050
319 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
320 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
321 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
322 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
323 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
324 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
325 Time:          21, CLK = 1, PC = 0x00000054
326 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
327 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
328 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
329 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
330 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
331 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
332 =====
333 Time:          22, CLK = 0, PC = 0x00000054
334 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
335 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
336 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020

```

[illegible]

395	[ \$s3 ]	=	0x00000020	,	[ \$s4 ]	=	0x00000001	,	[ \$s5 ]	=	0x00000000
396	[ \$s6 ]	=	0x00000000	,	[ \$s7 ]	=	0x00000000	,	[ \$t0 ]	=	0x00000020
397	[ \$t1 ]	=	0x00000037	,	[ \$t2 ]	=	0x00000000	,	[ \$t3 ]	=	0x00000000
398	[ \$t4 ]	=	0x00000000	,	[ \$t5 ]	=	0x00000000	,	[ \$t6 ]	=	0x00000000
399	[ \$t7 ]	=	0x00000000	,	[ \$t8 ]	=	0x00000000	,	[ \$t9 ]	=	0x00000000
400	Time :		26	,	CLK =	1	,	PC =	0x00000068		
401	[ \$s0 ]	=	0x00000037	,	[ \$s1 ]	=	0x00000037	,	[ \$s2 ]	=	0x00000000
402	[ \$s3 ]	=	0x00000020	,	[ \$s4 ]	=	0x00000001	,	[ \$s5 ]	=	0x00000000
403	[ \$s6 ]	=	0x00000000	,	[ \$s7 ]	=	0x00000000	,	[ \$t0 ]	=	0x00000020
404	[ \$t1 ]	=	0x00000037	,	[ \$t2 ]	=	0x00000000	,	[ \$t3 ]	=	0x00000000
405	[ \$t4 ]	=	0x00000000	,	[ \$t5 ]	=	0x00000000	,	[ \$t6 ]	=	0x00000000
406	[ \$t7 ]	=	0x00000000	,	[ \$t8 ]	=	0x00000000	,	[ \$t9 ]	=	0x00000000
407	=====										
408	Time :		27	,	CLK =	0	,	PC =	0x00000068		
409	[ \$s0 ]	=	0x00000037	,	[ \$s1 ]	=	0x00000037	,	[ \$s2 ]	=	0x00000000
410	[ \$s3 ]	=	0x00000020	,	[ \$s4 ]	=	0x00000001	,	[ \$s5 ]	=	0x00000000
411	[ \$s6 ]	=	0x00000000	,	[ \$s7 ]	=	0x00000000	,	[ \$t0 ]	=	0x00000020
412	[ \$t1 ]	=	0x00000037	,	[ \$t2 ]	=	0x00000000	,	[ \$t3 ]	=	0x00000000
413	[ \$t4 ]	=	0x00000000	,	[ \$t5 ]	=	0x00000000	,	[ \$t6 ]	=	0x00000000
414	[ \$t7 ]	=	0x00000000	,	[ \$t8 ]	=	0x00000000	,	[ \$t9 ]	=	0x00000000
415	Time :		27	,	CLK =	1	,	PC =	0x0000006c		
416	[ \$s0 ]	=	0x00000037	,	[ \$s1 ]	=	0x00000037	,	[ \$s2 ]	=	0x00000000
417	[ \$s3 ]	=	0x00000020	,	[ \$s4 ]	=	0x00000001	,	[ \$s5 ]	=	0x00000000
418	[ \$s6 ]	=	0x00000000	,	[ \$s7 ]	=	0x00000000	,	[ \$t0 ]	=	0x00000020
419	[ \$t1 ]	=	0x00000037	,	[ \$t2 ]	=	0x00000000	,	[ \$t3 ]	=	0x00000000
420	[ \$t4 ]	=	0x00000000	,	[ \$t5 ]	=	0x00000000	,	[ \$t6 ]	=	0x00000000
421	[ \$t7 ]	=	0x00000000	,	[ \$t8 ]	=	0x00000000	,	[ \$t9 ]	=	0x00000000
422	=====										
423	Time :		28	,	CLK =	0	,	PC =	0x0000006c		
424	[ \$s0 ]	=	0x00000037	,	[ \$s1 ]	=	0x00000037	,	[ \$s2 ]	=	0x00000000
425	[ \$s3 ]	=	0x00000020	,	[ \$s4 ]	=	0x00000001	,	[ \$s5 ]	=	0x00000000
426	[ \$s6 ]	=	0x00000000	,	[ \$s7 ]	=	0x00000000	,	[ \$t0 ]	=	0x00000020
427	[ \$t1 ]	=	0x00000037	,	[ \$t2 ]	=	0x00000000	,	[ \$t3 ]	=	0x00000000
428	[ \$t4 ]	=	0x00000000	,	[ \$t5 ]	=	0x00000000	,	[ \$t6 ]	=	0x00000000
429	[ \$t7 ]	=	0x00000000	,	[ \$t8 ]	=	0x00000000	,	[ \$t9 ]	=	0x00000000
430	Time :		28	,	CLK =	1	,	PC =	0x00000070		
431	[ \$s0 ]	=	0x00000037	,	[ \$s1 ]	=	0x00000037	,	[ \$s2 ]	=	0x00000000
432	[ \$s3 ]	=	0x00000020	,	[ \$s4 ]	=	0x00000001	,	[ \$s5 ]	=	0x00000000
43											

```

453 Time:          30, CLK = 0, PC = 0x00000074
454 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
455 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
456 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
457 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
458 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
459 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
460 Time:          30, CLK = 1, PC = 0x0000005c
461 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
462 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
463 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
464 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
465 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
466 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
467 =====
468 Time:          31, CLK = 0, PC = 0x0000005c
469 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
470 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
471 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
472 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
473 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
474 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
475 Time:          31, CLK = 1, PC = 0x00000060
476 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
477 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
478 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
479 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
480 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
481 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
482 =====
483 Time:          32, CLK = 0, PC = 0x00000060
484 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
485 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
486 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
487 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
488 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
489 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
490 Time:          32, CLK = 1, PC = 0x00000064
491 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
492 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
493 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
494 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
495 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
496 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
497 =====
498 Time:          33, CLK = 0, PC = 0x00000064
499 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
500 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
501 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
502 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
503 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
504 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
505 Time:          33, CLK = 1, PC = 0x00000068
506 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
507 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
508 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
509 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
510 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000

```



```

511 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
512 =====
513 Time:          34, CLK = 0, PC = 0x00000068
514 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
515 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
516 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
517 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
518 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
519 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
520 Time:          34, CLK = 1, PC = 0x0000006c
521 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
522 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
523 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
524 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
525 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
526 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
527 =====
528 Time:          35, CLK = 0, PC = 0x0000006c
529 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
530 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
531 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
532 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
533 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
534 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
535 Time:          35, CLK = 1, PC = 0x000000ac
536 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
537 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
538 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
539 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
540 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
541 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
542 =====
543 Time:          36, CLK = 0, PC = 0x000000ac
544 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
545 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
546 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
547 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
548 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
549 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
550 Time:          36, CLK = 1, PC = 0x000000b0
551 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
552 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
553 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
554 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
555 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
556 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
557 =====

```

## 5.2 Pipelined

Many files of pipeline processor are the same as single-cycle processor. The different files are shown below. clock\_clock\_divider.v

```

1 'timescale 1ns / 1ps
2
3 module clock_divider(divided_clock, clock, reset);
4

```

```

5  input clock , reset;
6  output divided_clock;
7
8  reg divided_clock1;
9
10 initial begin
11     divided_clock1 <= 0;
12 end
13
14 parameter r = 100;
15 integer now = 1;
16
17 //notice that reset is not used here
18 always @ (posedge clock)
19     begin
20         if (now <= r/2) divided_clock1 = 1;
21         else divided_clock1 =0;
22         if (now == r) now =1;
23         else now = now + 1;
24     end
25
26     assign divided_clock = divided_clock1;
27
28 endmodule

```

#### control.v

```

1  `ifndef MODULE_CONTROL
2  `define MODULE_CONTROL
3  `timescale 1ns / 1ps
4
5  module Control (
6      input      [5:0]  op_code ,
7      output reg  [1:0]  ALUOp,
8      output reg  RegDst ,
9                      Jump ,
10                     Branch ,
11                     Bne ,
12                     MemRead ,
13                     MemtoReg ,
14                     MemWrite ,
15                     ALUSrc ,
16                     RegWrite
17 );
18
19     initial begin
20         RegDst      = 1'b0;
21         Jump        = 1'b0;
22         Branch      = 1'b0;
23         Bne         = 1'b0;
24         MemRead     = 1'b0;
25         MemtoReg    = 1'b0;
26         MemWrite    = 1'b0;
27         ALUSrc      = 1'b0;
28         RegWrite    = 1'b0;
29         ALUOp       = 2'b00;
30     end
31
32     always @ ( * ) begin

```

```

33     case (op_code)
34         6'b000000: begin // R-type
35             RegDst      <= 1'b1;
36             Jump        <= 1'b0;
37             Branch      <= 1'b0;
38             Bne         <= 1'b0;
39             MemRead     <= 1'b0;
40             MemtoReg    <= 1'b0;
41             MemWrite    <= 1'b0;
42             ALUSrc      <= 1'b0;
43             RegWrite    <= 1'b1;
44             ALUOp       <= 2'b10;
45         end
46         6'b000010: begin // j
47             RegDst      <= 1'b1;
48             Jump        <= 1'b1;
49             Branch      <= 1'b0;
50             Bne         <= 1'b0;
51             MemRead     <= 1'b0;
52             MemtoReg    <= 1'b0;
53             MemWrite    <= 1'b0;
54             ALUSrc      <= 1'b0;
55             RegWrite    <= 1'b0;
56             ALUOp       <= 2'b10;
57     end
58     6'b000100: begin // beq
59         RegDst      <= 1'b1;
60         Jump        <= 1'b0;
61         Branch      <= 1'b1;
62         Bne         <= 1'b0;
63         MemRead     <= 1'b0;
64         MemtoReg    <= 1'b0;
65         MemWrite    <= 1'b0;
66         ALUSrc      <= 1'b0;
67         RegWrite    <= 1'b0;
68         ALUOp       <= 2'b01;
69     end
70     6'b000100: begin // bne
71         RegDst      <= 1'b1;
72         Jump        <= 1'b0;
73         Branch      <= 1'b0;
74         Bne         <= 1'b1;
75         MemRead     <= 1'b0;
76         MemtoReg    <= 1'b0;
77         MemWrite    <= 1'b0;
78         ALUSrc      <= 1'b0;
79         RegWrite    <= 1'b0;
80         ALUOp       <= 2'b01;
81     end
82     6'b001000: begin // addi
83         RegDst      <= 1'b0;
84         Jump        <= 1'b0;
85         Branch      <= 1'b0;
86         Bne         <= 1'b0;
87         MemRead     <= 1'b0;
88         MemtoReg    <= 1'b0;
89         MemWrite    <= 1'b0;
90         ALUSrc      <= 1'b1;

```

```

91         RegWrite    <= 1'b1;
92         ALUOp       <= 2'b00;
93     end
94     6'b001100: begin // andi
95         RegDst      <= 1'b0;
96         Jump        <= 1'b0;
97         Branch      <= 1'b0;
98         Bne         <=1'b0;
99         MemRead     <= 1'b0;
100        MemtoReg    <= 1'b0;
101        MemWrite    <= 1'b0;
102        ALUSrc      <= 1'b1;
103        RegWrite    <= 1'b1;
104        ALUOp       <= 2'b11;
105    end
106    6'b100011: begin // lw
107        RegDst      <= 1'b0;
108        Jump        <= 1'b0;
109        Branch      <= 1'b0;
110        Bne         <=1'b0;
111        MemRead     <= 1'b1;
112        MemtoReg    <= 1'b1;
113        MemWrite    <= 1'b0;
114        ALUSrc      <= 1'b1;
115        RegWrite    <= 1'b1;
116        ALUOp       <= 2'b00;
117    end
118    6'b101011: begin // sw
119        RegDst      <= 1'b0;
120        Jump        <= 1'b0;
121        Branch      <= 1'b0;
122        Bne         <=1'b0;
123        MemRead     <= 1'b0;
124        MemtoReg    <= 1'b0;
125        MemWrite    <= 1'b1;
126        ALUSrc      <= 1'b1;
127        RegWrite    <= 1'b0;
128        ALUOp       <= 2'b00;
129    end
130
131        default: ;
132    endcase
133 end
134
135 endmodule
136 'endif

```

#### forward\_mux.v

```

1  'timescale 1ns / 1ps
2  'ifndef MODULE_FORWARD_MUX
3  'define MODULE_FORWARD_MUX
4  module forward_mux (
5  EX_read ,
6  WB_result ,
7  MEM_result ,
8  out ,
9  select
10 );

```

```

11     parameter N = 32;
12     input [N-1:0] EX_read , MEM_result , WB_result ;
13     input  [1:0] select ;
14     output [N-1:0] out ;
15     reg [N-1:0] out ;
16     always@ (*)
17     begin
18         if ( select==2'b00) out<=EX_read ;
19         else if ( select==2'b01) out<=WB_result ;
20         else if ( select==2'b10) out<=MEM_result ;
21     end
22 endmodule
23 `endif

```

#### forwarding.v

```

1  `timescale 1ns / 1ps
2  module forwarding (
3      input beq ,
4      input bne ,
5      input [4:0] ID_rs ,
6      input [4:0] ID_rt ,
7      input EMRW,
8      input MWRW,
9      input MEM_MemWrite ,
10     input [4:0] EM_Rd ,
11     input [4:0] IE_Rs ,
12     input [4:0] IE_Rt ,
13     input [4:0] MW_Rd ,
14     output [1:0] beqA ,
15     output [1:0] beqB ,
16     output [1:0] ForwardA ,
17     output [1:0] ForwardB ,
18     output sw_src
19
20 );
21
22 reg [1:0] tmpA , tmp_beqA ;
23 reg [1:0] tmpB , tmp_beqB ;
24 reg temp_sw_src ;
25 initial begin
26     tmp_beqA <=0;
27     tmp_beqB <=0;
28     tmpA <=0;
29     tmpB <=0;
30     temp_sw_src <=0;
31 end
32
33 always@ (*)
34 begin
35     if (EMRW && (EM_Rd!=0) && (EM_Rd==ID_rs)&& (bne || beq))
36         begin
37             tmp_beqA <=2'b10;
38         end
39     else if ((bne || beq)&&(MWRW && (MW_Rd!=0) && (MW_Rd==ID_rs))&& ~(EMRW && (
40         EM_Rd!=0) && (EM_Rd==IE_Rs)))
41         begin
42             tmp_beqA <=2'b01;
43         end
44     end

```

```

43 else
44     begin
45         tmp_beqA <=2'b00;
46     end
47 if (EMRW && (EM_Rd!=0) && (EM_Rd==ID_rt)&& (bne || beq))
48     begin
49         tmp_beqB <=2'b10;
50     end
51     else if ((bne || beq)&&(MWRW && (MW_Rd!=0) && (MW_Rd==ID_rt))&& ~(EMRW
&& (EM_Rd!=0) && (EM_Rd==IE_Rs)))
52     begin
53         tmp_beqB <=2'b01;
54     end
55     else
56     begin
57         tmp_beqB <=2'b00;
58     end
59
60
61
62 if (MEM_MemWrite && MWRW &&(EM_Rd==MW_Rd))
63     begin
64         temp_sw_src <=1'b1;
65     end
66 else
67 begin
68     temp_sw_src <=1'b0;
69 end
70 if (EMRW && (EM_Rd!=0) && (EM_Rd==IE_Rs))
71     begin
72         tmpA<=2'b10;
73     end
74 else if ((MWRW && (MW_Rd!=0) && (MW_Rd==IE_Rs))&& ~(EMRW && (EM_Rd!=0) &&
(EM_Rd==IE_Rs)))
75     begin
76         tmpA<=2'b01;
77     end
78 else
79     begin
80         tmpA<=2'b00;
81     end
82
83 if (EMRW && (EM_Rd!=0) && (EM_Rd==IE_Rt))
84     begin
85         tmpB<=2'b10;
86     end
87 else if ((MWRW && (MW_Rd!=0) && (MW_Rd==IE_Rt))&& ~(EMRW && (EM_Rd!=0) &&
(EM_Rd==IE_Rt)))
88     begin
89         tmpB<=2'b01;
90     end
91 else
92     begin
93         tmpB<=2'b00;
94     end
95 end
96
97 assign beqA=tmp_beqA;

```

```

98 assign beqB=tmp_beqB;
99 assign ForwardA=tmpA;
100 assign ForwardB=tmpB;
101 assign sw_src=temp_sw_src;
102 endmodule
103 /*module forwarding(
104 EX_rs,EX_rt,
105 MEM_rd, MEM_RegWrite,
106 WB_rd, WB_RegWrite,
107 selectA, selectB
108 );
109 input EX_rs,EX_rt, MEM_rd, MEM_RegWrite, WB_rd, WB_RegWrite;
110 output [1:0] selectA, selectB;
111 reg [1:0] selectA, selectB;
112 initial begin
113 selectA <=2'b00;
114 selectB <=2'b00;
115 end
116 always@(*)
117 begin
118 selectA <=2'b00;
119 selectB <=2'b00;
120 if ((MEM_RegWrite==1'b1)&&(MEM_rd!=0)) //EX hazard
121 begin
122 if (EX_rs==MEM_rd) selectA <=2'b10;
123 if (EX_rt==MEM_rd) selectB <=2'b10;
124 end
125 if ((WB_RegWrite==1'b1)&&(WB_rd!=0)) // MEM hazard
126 begin
127 if (selectA==2'b00)
128 begin
129 if (EX_rs==MEM_rd) selectA <=2'b01;
130 end
131 if (selectA==2'b00)
132 begin
133 if (EX_rt==MEM_rd) selectB <=2'b01;
134 end
135 end
136 end
137 endmodule*/

```

#### HazardDtection.v

```

1 `timescale 1ns / 1ps
2
3
4 module HazardDtection(
5     //input IF_instruction,
6     input clk,
7     input bne,
8     input branch,
9     input jump,
10    input ID_MemRead,
11    input ID_RegWrite,
12    input EX_MemRead,
13    input [4:0] EX_Rt,
14    input [4:0] ID_Rt,
15    input [4:0] ID_Rd,
16    input [4:0] IF_Rs,

```

```

17  input [4:0] IF_Rt ,
18  output IF_Write ,
19  output PC_Write
20  //output Controlsrc
21  );
22
23  reg PCtem;
24  // reg Controltem;
25  reg IFIDtem;
26
27  initial begin
28      PCtem<=1;
29      // Controltem <=0;
30      IFIDtem <=1;
31  end
32
33
34  always@(posedge clk)
35  begin
36      //load use hazard:lw $2,0($1);and $4,$2,$5
37      if (ID_MemRead && ((ID_Rt==IF_Rs) | (ID_Rt==IF_Rt))&&(branch==0)&&(
jump==0)&&(bne==0))
38      begin
39          PCtem<=0;
40          // Controltem <=1;
41          IFIDtem <=0;
42      end
43      //lw and not adjacent beq
44      else if ((bne || branch) && EX_MemRead &&((EX_Rt==IF_Rs) | (EX_Rt==
IF_Rt))&&(jump==0))
45      begin
46          PCtem<=0;
47          // Controltem <=1;
48          IFIDtem <=0;
49      end
50      //l/lw and adjacent beq
51      else if ((bne || branch) && ID_RegWrite&& ((ID_Rt==IF_Rs&& IF_Rs !=0)
|(ID_Rt==IF_Rt&& IF_Rt !=0))&&(jump==0))
52      begin
53          PCtem<=0;
54          // Controltem=1;
55          IFIDtem <=0;
56      end
57      // r-type and adjacent beq
58      else if ((bne || branch) && ID_RegWrite && ((ID_Rd==IF_Rs&&IF_Rs !=0) |(
ID_Rd==IF_Rt&& IF_Rt !=0))&&(jump==0))
59      begin
60          PCtem<=0;
61          // Controltem=1;
62          IFIDtem <=0;
63      end
64      /* else if (jump)
65      begin
66          PCtem=1;
67          Controltem=0;
68          IFIDtem=1;
69      end */
70

```



```

71     else
72     begin
73         PCtem<=1;
74         IFIDtem<=1;
75         // Controltem<=0;
76     end
77
78
79
80     end
81
82
83     assign IF_Write=IFIDtem;
84     assign PC_Write=PCtem;
85     // assign Controlsrc=Controltem;
86 endmodule

```

#### ID\_EXpipeline.v

```

1  `timescale 1ns / 1ps
2
3
4  module ID_EXpipeline(
5      input  RegWrite ,
6      input  MemtoReg ,
7      input  MemRead ,
8      input  MemWrite ,
9      input  [1:0] ALUop ,
10     input  ALUsrc ,
11     input  RegDst ,
12     input  [31:0] Readdata1 ,
13     input  [31:0] Readdata2 ,
14     input  [31:0] ExtendResult ,
15     input  [4:0] Rd ,
16     input  [4:0] Rt ,
17     input  [4:0] Rs ,
18     input  [5:0] func ,
19
20     output reg  RegWrite1 ,
21     output reg  MemtoReg1 ,
22     output reg  MemRead1 ,
23     output reg  MemWrite1 ,
24     output reg  [1:0] ALUop1 ,
25     output reg  ALUsrc1 ,
26     output reg  RegDst1 ,
27     output reg  [31:0] RData1 ,
28     output reg  [31:0] RData2 ,
29     output reg  [31:0] ER ,
30     output reg  [4:0] Rd1 ,
31     output reg  [4:0] Rt1 ,
32     output reg  [4:0] Rs1 ,
33     output reg  [5:0] func1 ,
34     input  clk
35 );
36
37     always@(negedge clk)
38     begin
39         Rs1<=Rs ;
40         RegWrite1<=RegWrite ;

```

```

41     MemtoReg1<=MemtoReg;
42     MemRead1<=MemRead;
43     MemWrite1<=MemWrite;
44     ALUOp1<=ALUOp;
45     ALUSrc1<=ALUSrc;
46     RData1<=Readdata1;
47     RData2<=Readdata2;
48     ER<=ExtendResult;
49     Rd1<=Rd;
50     Rt1<=Rt;
51     RegDst1<=RegDst;
52     func1<=func;
53 end
54
55 endmodule

```

## PC\_MUX.v

```

1  `timescale 1ns / 1ps
2  module PC_MUX(
3  IF_PC_plus4 ,
4  bne_mux ,
5  beq_mux ,
6  ID_Jump ,
7  ID_Jaddress ,
8  ID_target ,
9  PC_in
10 );
11 input [31:0] IF_PC_plus4 , ID_Jaddress , ID_target;
12 input beq_mux ,bne_mux , ID_Jump;
13 output [31:0] PC_in;
14 reg [31:0] PC_in;
15 always@ (*)
16 begin
17 if (beq_mux==1||bne_mux ==1) PC_in<=ID_target;
18 else if (ID_Jump==1) PC_in<=ID_Jaddress;
19 else PC_in<=IF_PC_plus4;
20 end
21 endmodule

```

## pipe\_write.v

```

1  `timescale 1ns / 1ps
2  `ifndef MODULE_PIPE_WRITE
3  `define MODULE_PIPE_WRITE
4  module pipe_write(
5  Reg_in ,
6  Reg_out ,
7  flush ,
8  clk ,
9  hold
10 );
11 //after finish the stage , push the items into next pipelined register(every
    stage execute at the same time)
12 //IFtoID: instruction , PC+4
13 //IDtoEX:Jump , Branch , MemRead , MemtoReg ,ALUOp ,MemWrite ,ALUSrc ,RegWrite ,
    ID_rs , ID_rt , ID_read1 , ID_read2 , ID_rd ,PC+4
14 //EXtoMEM: MemRead , MemtoReg , ,MemWrite ,RegWrite ,EX_out , EX_rd
15 //MEMtoWB: MemtoReg , ,RegWrite ,WB_rd
16 parameter size = 64; //size of the pipelined registers

```

```

17  input [size-1:0] Reg_in;
18  input flush,clk,hold;
19  output [size-1:0] Reg_out;
20  reg [size-1:0] Reg_out;
21  always@(negedge clk) // second half clock for write
22  begin
23  if(flush==1 || hold ==1) Reg_out<=0;
24  else Reg_out<=Reg_in;
25  end
26 endmodule
27 `endif

```

## pipeline.v

```

1  `timescale 1ns / 1ps
2  module pipeline(
3  clk,reset,C,AN,clock,reg_switch,PC_switch
4  );
5  input clk,reset,clock,PC_switch;
6  input [4:0] reg_switch;
7  output reg [6:0] C;
8  output [3:0] AN;
9  initial begin
10     C=7'b1111111;
11  end
12
13 // ALU
14 wire [31:0] ALU_a,ALU_b,ALU_out,EX_result,MEM_result,WB_result,temp_ALU_b;
15 wire [3:0] ALU_control;
16 wire [5:0] EX_func;
17 wire [1:0] ID_ALUOp, EX_ALUOp,selectA,selectB, beqA, beqB;
18 // Control Signals
19 wire ID_RegDst,ID_Jump,ID_Branch,ID_ALUSrc,ID_MemRead,ID_MemWrite,
    ID_MemtoReg,ID_RegWrite,ID_Bne,
20 EX_RegDst,EX_Jump,EX_Branch,EX_ALUSrc,EX_MemRead,EX_MemWrite,EX_MemtoReg,
    EX_RegWrite,EX_Bne,
21 MEM_Jump,MEM_Branch,MEM_MemRead,MEM_MemWrite,MEM_MemtoReg,MEM_RegWrite,
    MEM_Bne,
22 WB_MemtoReg,WB_RegWrite,EX_Zero,MEM_Zero;
23 // address
24 wire [31:0] ID_extend, EX_extend,ID_Jaddress,ID_target;
25 //PC Signals
26 wire [31:0] PC_in, PC_out, IF_PC_plus4, ID_PC_plus4;
27 // Instructions memory signals
28 wire [31:0] IF_instruction, ID_instruction, sw_data, beq1, beq2;
29 // Register data
30 wire [31:0] ID_read1, ID_read2, EX_read1,EX_read2,write_data,MEM_read2,
    MEM_data,WB_data;
31 // Registers
32 wire [4:0] EX_rs,EX_rt,EX_rd,temp_rd, MEM_rd, WB_rd;
33 //bne,neq
34 wire beq_mux,bne_mux,ID_Zero,sw_src;
35 assign ID_Zero= (beq1==beq2)? 1:0;
36 assign beq_mux = ID_Zero & ID_Branch;
37 assign bne_mux = (~ID_Zero) & ID_Bne;
38 //hazard
39 wire PC_write, IF_Flush, IF_Write,beq,bne,jump;
40 assign beq = (IF_instruction[31:26]==6'b000100)?1:0;
41 assign bne = (IF_instruction[31:26]==6'b000101)?1:0;

```

```

42 assign jump = (IF_instruction[31:26]==6'b000010)?1:0;
43 assign IF_Flush = beq_mux | bne_mux | ID_Jump;
44 // Connect PC
45 program_counter PC(PC_in,PC_out,reset,clk,~PC_write);
46 //IF
47 //instruction memory
48 instruction_memory get_ins(
49     .address(PC_out),
50     .instruction(IF_instruction)
51 );
52 // Cal PC + 4
53 adder add_PC_plus4(
54     .a(PC_out),
55     .b(32'b0100),
56     .sum(IF_PC_plus4)
57 );
58
59 //Jump target address
60 assign ID_Jaddress={ID_PC_plus4[31:28],ID_instruction[25:0],2'b00};
61 //Hazard Detection Unit
62 HazardDtection HD(
63     // IF_instruction ,
64     clk ,
65     bne ,
66     beq ,
67     jump ,
68     ID_MemRead ,
69     ID_RegWrite ,
70     EX_MemRead ,
71     EX_rt ,
72     ID_instruction[20:16] ,
73     ID_instruction[15:11] ,
74     IF_instruction[26:21] ,
75     IF_instruction[20:16] ,
76     IF_Write ,
77     PC_write
78 );
79 // Register File
80 wire [31:0] Q2;
81 assign ID_target=ID_PC_plus4 + ID_extend * 4;
82 Registers get_reg(
83     .read_register_1 (ID_instruction[25:21]),
84     .read_register_2 (ID_instruction[20:16]),
85     .switch(reg_switch),
86     .regWrite(WB_RegWrite),
87     .write_register(WB_rd),
88     .write_data(write_data),
89     .read_data_1 (ID_read1),
90     .read_data_2 (ID_read2),
91     .clk(clk),
92     .Q2(Q2)
93 );
94 Control controller(ID_instruction[31:26], ID_ALUOp,ID_RegDst,ID_Jump,
    ID_Branch,ID_Bne,ID_MemRead, ID_MemtoReg,ID_MemWrite,ID_ALUSrc,
    ID_RegWrite);
95 sign_extension sign_ext(
96     .shortInput(ID_instruction[15:0]),
97     .longOutput(ID_extend)

```

```

98 );
99 //EX
100 // ALU
101 ALU alu (
102     .ALUCtrl( ALU_control ) ,
103     .a( ALU_a ) ,
104     .b( ALU_b ) ,
105     .ALU_result( EX_result ) ,
106     .zero( EX_Zero )
107 );
108 // ALU control
109 ALU_control alu_ctrl (
110     .funct( EX_func ) ,
111     .ALUOp( EX_ALUOp ) ,
112     .ALUCtrl( ALU_control )
113 );
114 // forwarding
115
116 forwarding forwarding_unit (
117     ID_Branch ,
118     ID_Bne ,
119     ID_instruction [25:21] ,
120     ID_instruction [20:16] ,
121     MEM_RegWrite ,
122     WB_RegWrite ,
123     MEM_MemWrite ,
124     MEM_rd ,
125     EX_rs ,
126     EX_rt ,
127     WB_rd ,
128     beqA ,
129     beqB ,
130     selectA ,
131     selectB ,
132     sw_src
133 );
134 //MEM
135 data_memory dm (
136     .clk( clk ) ,
137     .MemRead( MEM_MemRead ) ,
138     .MemWrite( MEM_MemWrite ) ,
139     .address( MEM_result ) ,
140     .write_data( sw_data ) ,
141     .read_data( MEM_data )
142 );
143 // Mux
144 Mux_N_bit#(5) mux_select_write_reg (
145     .in1( EX_rt ) ,
146     .in2( EX_rd ) ,
147     .select( EX_RegDst ) ,
148     .out( temp_rd )
149 );
150 Mux_N_bit#(32) mux_select_ALU_in (
151     .in1( temp_ALU_b ) ,
152     .in2( EX_extend ) ,
153     .select( EX_ALUSrc ) ,
154     .out( ALU_b )
155 );

```

```

156 forward_mux BEQ_MUX1(ID_read1 , write_data , MEM_result , beq1 , beqA );
157 forward_mux BEQ_MUX2(ID_read2 , write_data , MEM_result , beq2 , beqB );
158 forward_mux ALU_MUX1(EX_read1 , write_data , MEM_result , ALU_a , selectA );
159 forward_mux ALU_MUX2(EX_read2 , write_data , MEM_result , temp_ALU_b , selectB );
160 Mux_N_bit#(32) mux_write(
161     .in1 (WB_result) ,
162     .in2 (WB_data) ,
163     .select (WB_MemtoReg) ,
164     .out (write_data)
165 );
166 Mux_N_bit#(32) mux_sw(
167     .in1 (MEM_read2) ,
168     .in2 (write_data) ,
169     .select (sw_src) ,
170     .out (sw_data)
171 );
172 PC_MUX next_PC (IF_PC_plus4 , bne_mux , beq_mux , ID_Jump , ID_Jaddress , ID_target ,
    PC_in );
173 //overwrite register
174 pipe_write#(64) IF_ID ( { IF_instruction , IF_PC_plus4 } , { ID_instruction ,
    ID_PC_plus4 } , IF_Flush , clk , ~ IF_Write );
175 ID_EXpipeline ID_EX(
176     ID_RegWrite ,
177     ID_MemtoReg ,
178     ID_MemRead ,
179     ID_MemWrite ,
180     ID_ALUOp ,
181     ID_ALUSrc ,
182     ID_RegDst ,
183     ID_read1 ,
184     ID_read2 ,
185     ID_extend ,
186     ID_instruction [15:11] ,
187     ID_instruction [20:16] ,
188     ID_instruction [25:21] ,
189     ID_instruction [5:0] ,
190     EX_RegWrite ,
191     EX_MemtoReg ,
192     EX_MemRead ,
193     EX_MemWrite ,
194     EX_ALUOp ,
195     EX_ALUSrc ,
196     EX_RegDst ,
197     EX_read1 ,
198     EX_read2 ,
199     EX_extend ,
200     EX_rd ,
201     EX_rt ,
202     EX_rs ,
203     EX_func ,
204     clk
205 );
206
207 pipe_write#(85) EXMEM({ temp_rd , EX_result , EX_read2 , EX_Zero , EX_MemRead ,
    EX_MemWrite , EX_MemtoReg , EX_RegWrite } ,
208 { MEM_rd , MEM_result , MEM_read2 , MEM_Zero , MEM_MemRead , MEM_MemWrite , MEM_MemtoReg
    , MEM_RegWrite } ,
209 0 , clk , 0 );

```

```

210 pipe_write #(71) MEMWB({ MEM_result , MEM_data , MEM_rd , MEM_MemtoReg ,
    MEM_RegWrite } ,
211 { WB_result , WB_data , WB_rd , WB_MemtoReg , WB_RegWrite } ,
212 0 , clk , 0 ) ;
213 //SSD
214 wire clock_d ;
215 reg [3:0] Q ;
216
217 clock_divider clock1 ( clock_d , clock , reset ) ;
218
219 ring_counter ring ( AN , clock_d , reset ) ;
220
221 wire [31:0] ssd_out ;
222 // here only needs to use 4 hexo numbers because first 4 numbers must be
    0 for the small instruction value
223 assign ssd_out = ( PC_switch ) ? PC_out : Q2 ;
224
225 always @ ( * ) begin
226     if ( AN == 4'b1110 ) Q <= ssd_out [3:0] ;
227     else if ( AN == 4'b1101 ) Q <= ssd_out [7:4] ;
228     else if ( AN == 4'b1011 ) Q <= ssd_out [11:8] ;
229     else if ( AN == 4'b0111 ) Q <= ssd_out [15:12] ;
230 end
231 // ssd numbers
232 always @ ( * ) begin
233     case ( Q )
234         4'h0 : C = 7'b1000000 ;
235         4'h1 : C = 7'b1111001 ;
236         4'h2 : C = 7'b0100100 ;
237         4'h3 : C = 7'b0110000 ;
238         4'h4 : C = 7'b0011001 ;
239         4'h5 : C = 7'b0010010 ;
240         4'h6 : C = 7'b0000010 ;
241         4'h7 : C = 7'b1111000 ;
242         4'h8 : C = 7'b0000000 ;
243         4'h9 : C = 7'b0010000 ;
244         4'ha : C = 7'b0001000 ;
245         4'hb : C = 7'b0000011 ;
246         4'hc : C = 7'b1000110 ;
247         4'h d : C = 7'b0100001 ;
248         4'he : C = 7'b0000110 ;
249         4'h f : C = 7'b0001110 ;
250         default : C = 7'b1111111 ;
251     endcase
252 end
253
254 endmodule

```

#### program\_counter.v

```

1 `ifndef MODULE_PC
2 `define MODULE_PC
3 `timescale 1ns / 1ps
4 module program_counter (
5     PC_in , PC_out , reset , clk , hold
6 );
7     input [31:0] PC_in ;
8     output [31:0] PC_out ;
9     input reset , clk , hold ;

```

```

10 reg [31:0] PC_out;
11 initial begin
12     PC_out <= 0;
13 end
14 always @(negedge clk)
15 begin
16     if(reset)
17         begin
18             PC_out <= 32'b0;
19         end
20     else if (!hold)
21         begin
22             PC_out<=PC_in[31:0];
23         end
24     end
25 endmodule
26 `endif

```

#### register.v

```

1 `ifndef MODULE_REGISTERS
2 `define MODULE_REGISTERS
3 `timescale 1ns / 1ps
4
5 module Registers (
6     input      clk ,
7     regWrite ,
8     input      [4:0]  read_register_1 ,  read_register_2 , switch ,
9     input      [4:0]  write_register ,
10    input      [31:0]  write_data ,
11    output     [31:0]  read_data_1 ,read_data_2 ,Q2
12 );
13 parameter size = 32;          // 32-bit CPU, $0 - $31
14 reg [31:0] register_memory [0:size-1];
15 integer i;
16
17 initial begin
18     for (i = 0; i < size; i = i + 1)
19         register_memory[i] = 32'b0;
20 end
21 assign read_data_1 = register_memory[read_register_1];
22 assign read_data_2 = register_memory[read_register_2];
23 assign Q2 = register_memory[switch];
24 always @(posedge clk) begin
25     if (regWrite == 1)
26         register_memory[write_register] <= write_data;
27 end
28 // falling edge for reading
29 // always @(negedge clk) begin
30 //     assign read_data_1 = register_memory[read_register_1];
31 //     assign read_data_2 = register_memory[read_register_2];
32 // end
33
34 // Waiting for fall edge may cause some problem.
35 endmodule // registers
36 `endif

```

#### ring\_counter.v

```

1 `timescale 1ns / 1ps

```



```

2
3 module ring_counter(Q, clock, reset);
4     input clock, reset;
5     output [3:0]Q;
6
7     reg [3:0]Q;
8
9     initial begin
10    Q <= 4'b1110;
11    end
12
13    always @ (posedge reset or posedge clock)
14    begin
15        if (reset == 1'b1) Q <= 4'b1110;
16        else if (Q == 4'b1110) Q <= 4'b0111;
17        else if (Q == 4'b1101) Q <= 4'b1110;
18        else if (Q == 4'b1011) Q <= 4'b1101;
19        else if (Q == 4'b0111) Q <= 4'b1011;
20    end
21
22 endmodule

```