



JOINT INSTITUTE  
交大密西根学院

# VE370 Introduction to Computer Organization

## Project 2

### Digital Design Using Verilog to Implement Single- cycle & Pipelined Processors

*Team Members:*

- *QI Fubo 516370910175*
- *HU Bingcheng 516021910219*
- *NG Jing Ni Ashley 515370990007*

# Contents

## VE370 Introduction to Computer Organization

### Project 2

### Digital Design Using Verilog to Implement Single-cycle & Pipelined Processors

#### Contents

#### I. Objectives

Introduction

#### II. Circuit Design

i. Single-cycle datapath

ii. Pipelined datapath

#### III. Design of Components

i. IF Stage

1. PC MUX

2. PC Register

3. PC Adder 4

4. Instruction Memory

ii. EX Stage

1. Forwarding Unit and MUX

2. ALU Control

3. ALU

iii. Memory Stage and Write Back Stage

1. Data Memory

#### VI. Control and Data Hazard

i. EXstage

Data Hazard

ii. ID stage

Data Hazard

Control Hazard

#### VII. Instruction Implementation

i. Data tables

#### VIII. SSD and Top Module

#### IX. RTL Schematic (Optional)

#### X. Textual Result

#### XI. Conclusion and Discussion

#### X. Reference

#### XII. Appendix

1. TextualResult

*Single*

*Pipline*

2. adder.v

3. ALU\_control.v

4. ALU.v

5. data\_memory.v

6. forwarding\_unit.v

7. hazard\_detection.v

8. instruction\_memory.v

9. Mux\_N\_bit.v
10. pc.v
11. pipeline.v
12. pipe\_test.v
13. register.v
14. sign\_extension.v
15. state\_register.v
16. control.v

# I. Objectives

- To build up both single-cycle and pipelined datapaths and construct a simple version of a processor sufficient to implement a subset of the MIPS instruction set that includes:
  1. The **memory-reference instructions** load word (`lw`) and store word (`sw`)
  2. The **arithmetic-logical instructions** `add`, `addi`, `sub`, `and`, `andi`, `or`, and `slt`
  3. The **jumping instructions** branch equal (`beq`), branch not equal (`bne`), and jump (`j`)
- To model and simulate the single-cycle and pipelined datapaths in Verilog HDL
- To synthesize the results by using Xilinx synthesis tools

## Introduction

Aiming at enhancing our understanding of how the central processing unit (CPU) works, this project invites us to build a MIPS single-cycle processor and a MIPS pipelined processor using Verilog.

The single-cycle implementation executes all instructions in one clock cycle. This means that no datapath resource can be used more than once per instruction, so any element needed more than once must be duplicated. Therefore, a memory for instructions is separated from one for data. Although some of the functional units will need to be duplicated, many of the elements can be shared by different instruction flows. In order to share a datapath element between two or more different instruction classes, multiple connections to the input of an element are allowed, and multiplexors and control signals are designed carefully to select among the multiple inputs.

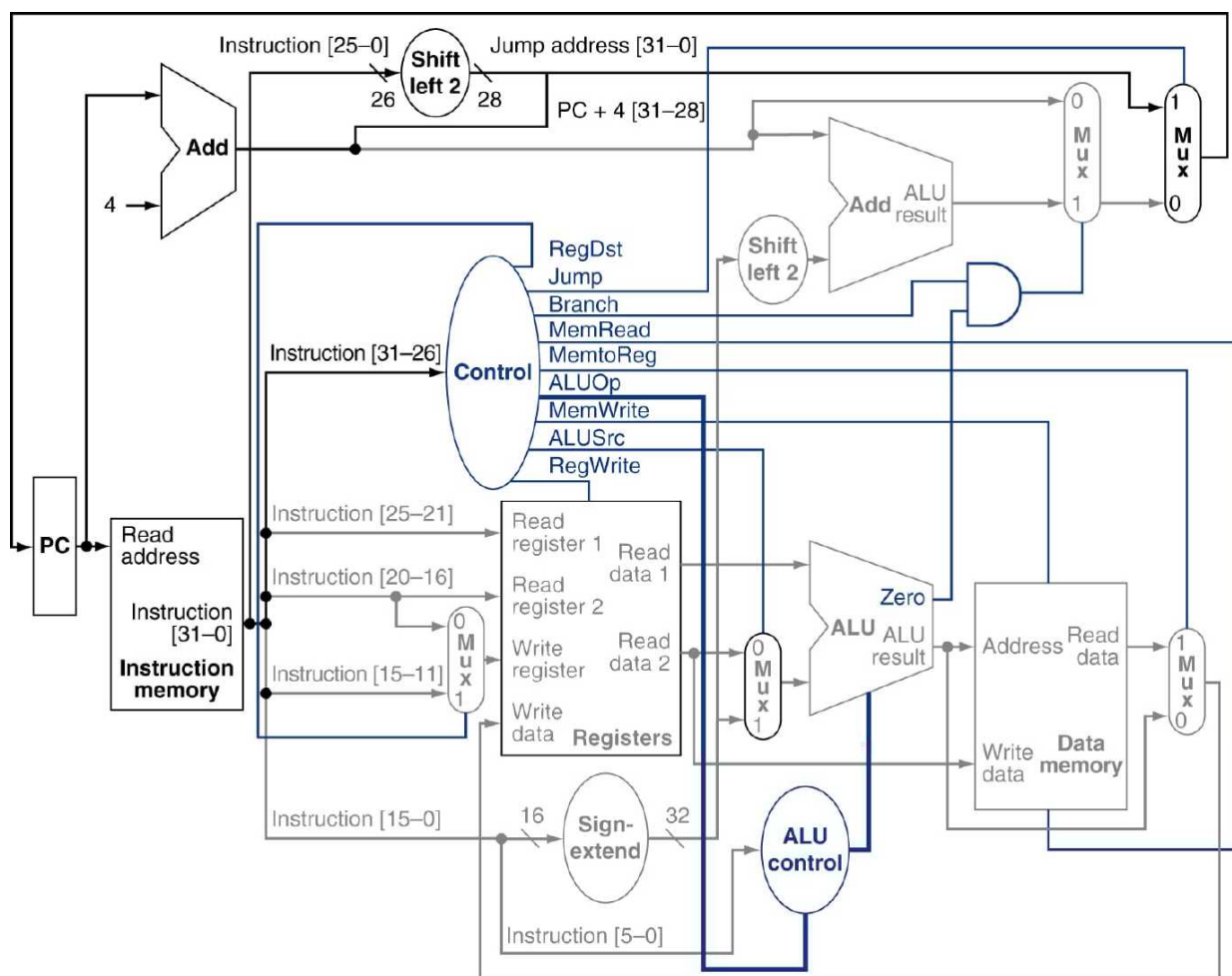
However, the single-cycle implementation has an unsatisfactory performance due to its inefficiency; the execution time of each instruction is one clock cycle, which is determined by the longest possible path in the processor. Using the same hardware components, the pipelined implementation allows different functional units of a system to run concurrently. As a result, pipelining technique significantly reduces the execution time of a same program compared to the single-cycle implementation.

In order to prevent data hazards in the pipeline and minimize the delay created by stalls, a forwarding unit and a hazard detection unit are implemented. The forwarding technique retrieves the missing data element from internal buffers rather than waiting for it to arrive from programmer-visible registers or memory. The hazard detection unit operates during the ID stage so that it can insert the stall between the load and its use; in this case, stalling is inevitable. For control hazards that might occur during the operation of jumping instructions, we assume branch is not taken and stall if the assumption is not correct during the ID stage via the hazard detection unit. For the case when there is a load before a branch, unfortunately, as stated before, stalls are added so that the word is successfully written in the instruction memory.

## II. Circuit Design

### i. Single-cycle datapath

Our project follows the following single-cycle processor schematic from the textbook. This version of the MIPS single-cycle processor can execute the following instructions: add, sub, and, or, slt, lw, sw, beq, bne, addi, andi and j.

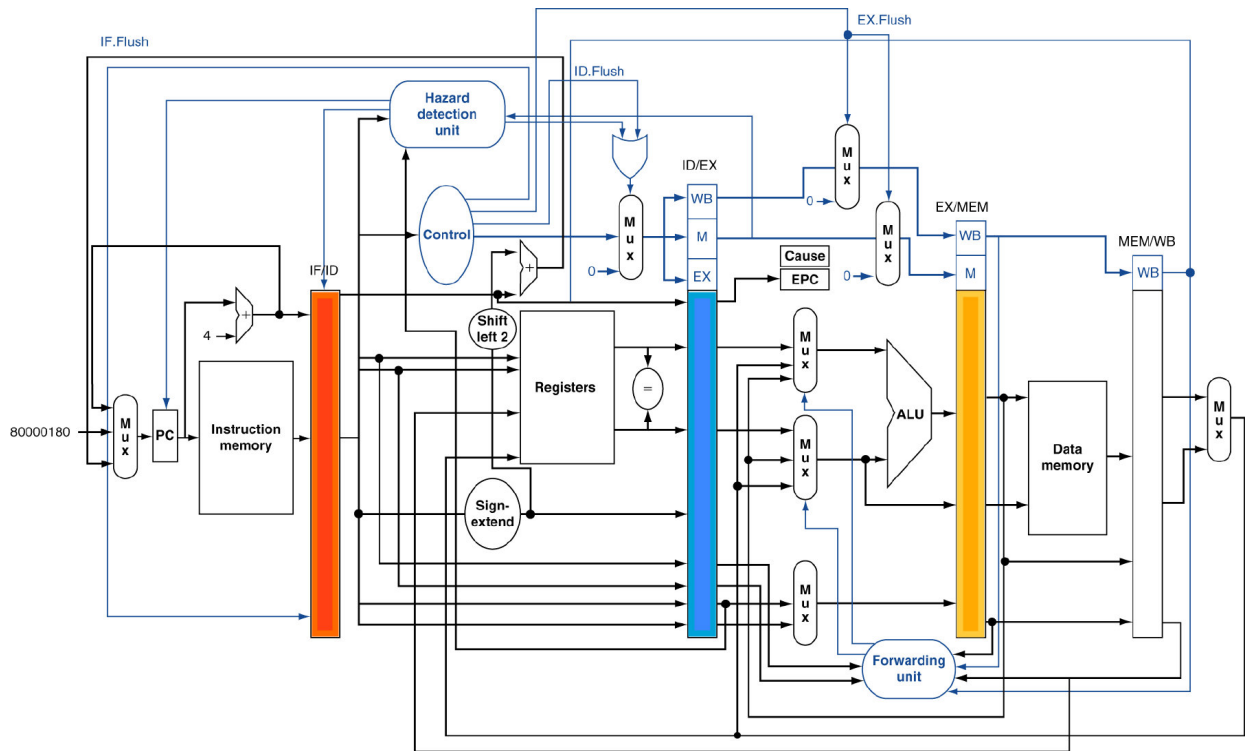


The model divides the machine into two main units: control and data paths. Each unit consists of various functional blocks, such as a 32-bit ALU, register file, sign extension logic, and five multiplexers to select the appropriate operand.

For the ALU control unit and the 32-bit ALU, our design follows six combinations of four control inputs in the textbook. The figure below shows how to set the ALU control bits according to the different function codes of the ALUOp control bit and the R type command.

## ii. Pipelined datapath

Our project follows the following pipelined processor schematic from the textbook. This version of the MIPS single-cycle processor can execute the following instructions: add, sub, and, or, slt, lw, sw, beq, bne, addi, andi and j.



## III. Design of Components

### i. IF Stage

#### 1. PC MUX

The PC multiplexor controls what value replaces the PC (PC+4, the branch destination address or the jump destination address). The Verilog code for the PC multiplexor is

```

1 | module Mux_N_bit(in1,in2,out,select);
2 |     parameter N = 32;
3 |     input [N-1:0] in1,in2;
4 |     input select;
5 |     output [N-1:0] out;
6 |     assign out = select?in2:in1;
7 | endmodule

```

#### 2. PC Register

The program counter is a 32-bit register that is written at the end of every clock cycle and thus does not need a write control signal. The Verilog code for PC register is

```

1 | module PC (
2 |     input          clk,
3 |                 PCWrite,
4 |     input          [31:0] in,
5 |     output reg     [31:0] out

```

```

6   );
7
8   initial begin
9       out = 32'b0;
10  end
11
12  always @ (posedge clk) begin
13      if (PCWrite)
14          out <= in;
15  end
16
17  endmodule

```

### 3. PC Adder 4

The PC adder is an ALU wired to always add its two 32-bit inputs and place the sum on its output. The Verilog code for the adder is

```

1  module adder(
2      input [31:0] a,
3      input [31:0] b,
4      output [31:0] sum
5  );
6      reg [31:0] sum;
7      always @(a or b)
8          begin
9              sum = a + b;
10         end
11  endmodule

```

### 4. Instruction Memory

The instruction memory only reads, we treat it as a combinational logic: the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed. The Verilog code for the instruction memory is

```

1  module instruction_memory (
2      input [31:0] address,
3      output [31:0] instruction
4  );
5
6      parameter size = 128; // you can change here, size is the max size of memory
7      integer i;
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.txt"
16     end
17     // Output the memory at address
18     assign instruction = memory[address >> 2];
19
20 endmodule

```

## ii. EX Stage

### 1. Forwarding Unit and MUX

The forwarding unit detects the hazards in the EX and MEM stage and assigns the ALU forwarding control of the multiplexors. According to the conditions for the two kinds of data hazards, the Verilog code for the forwarding unit module is written as follow.

```

1  module Forward (
2      input      [4:0]  registerRsID,
3                  registerRtID,
4                  registerRsEX,
5                  registerRtEX,
6                  registerRdMEM,
7                  registerRdWB,
8      input      regWriteMEM,
9                  regWriteWB,
10     output reg  [1:0] forwardA,
11                  forwardB,
12     output reg  forwardC,
13                  forwardD
14 );
15
16     initial begin
17         forwardA = 2'b00;
18         forwardB = 2'b00;
19         forwardC = 1'b0;
20         forwardD = 1'b0;
21     end
22
23     always @ ( * ) begin
24         if (regWriteMEM && registerRdMEM && registerRdMEM == registerRsEX)
25             forwardA = 2'b10;
26         else if (regWriteWB && registerRdWB && registerRdWB == registerRsEX)
27             forwardA = 2'b01;
28         else
29             forwardA = 2'b00;
30
31         if (regWriteMEM && registerRdMEM && registerRdMEM == registerRtEX)
32             forwardB = 2'b10;
33         else if (regWriteWB && registerRdWB && registerRdWB == registerRtEX)

```



```

34         forwardB = 2'b01;
35     else
36         forwardB = 2'b00;
37
38     if (regWriteMEM && registerRdMEM && registerRdMEM == registerRsID)
39         forwardC = 1'b1;
40     else
41         forwardC = 1'b0;
42
43     if (regWriteMEM && registerRdMEM && registerRdMEM == registerRtID)
44         forwardD = 1'b1;
45     else
46         forwardD = 1'b0;
47     end
48
49 endmodule // Forward

```

As the pipeline registers hold the data to be forwarding, we take inputs to the ALU from any pipeline register rather than just ID/EX, so that the proper data can be forwarded to the next stage. By adding multiplexors to the input of the ALU, and with the proper controls determined by the forwarding unit shown in the following figure, the pipeline can run at full speed in the presence of the data dependences.

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

The Verilog code for each 3-to-1 Mux is

```

1  module Mux_32bit_3to1 (in00, in01, in10, mux_out, control);
2      input [31:0] in00, in01, in10;
3      output [31:0] mux_out;
4      input [1:0] control;
5      reg [31:0] mux_out;
6      always @(in00 or in01 or in10 or control)
7          begin
8              case(control)
9                  2'b00:mux_out<=in00;
10                 2'b01:mux_out<=in01;
11                 2'b10:mux_out<=in10;
12                 default: mux_out<=in00;
13             endcase
14         end

```

## 2. ALU Control

The ALU control unit generates a 4-bit control input to the ALU with the function field of the instruction and a 2-bit control field ALUOp determined by the main control unit. Our design follows the that in the textbook that shows how the ALU control inputs are set based on the 2-bit ALUOp control and the 6-bit function code.

Instruction opcode	ALUOp	Instruction operation	Func field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

The Verilog code for the ALU control module is

```

1  module ALU_control(
2      funct,ALUOp,ALUCtrl
3  );
4      input [5:0] funct;
5      input [1:0] ALUOp;
6      output [3:0] ALUCtrl;
7      reg [3:0] ALUCtrl;
8      always @ (funct or ALUOp)
9          begin
10             case(ALUOp)
11                 2'b00: assign ALUCtrl = 4'b0010;
12                 2'b01: assign ALUCtrl = 4'b0110;
13                 2'b10:
14                     begin
15                         if (funct == 6'b100000)    assign ALUCtrl = 4'b0010;
16                         else if (funct == 6'b100010) assign ALUCtrl = 4'b0110;
17                         else if (funct == 6'b100100) assign ALUCtrl = 4'b0000;
18                         else if (funct == 6'b100101) assign ALUCtrl = 4'b0001;
19                         else if (funct == 6'b101010) assign ALUCtrl = 4'b0111;
20                     end
21                 2'b11:
22                     assign ALUCtrl = 4'b0000;
23                     // default: assign ALUCtrl = 4'b1111;
24             endcase
25         end
26     endmodule

```

### 3. ALU

Depending on the ALU control lines, the ALU performs one of the functions shown in the following table.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

For load word and store word instructions, ALU computes the memory address by addition. For the R-type instructions, ALU performs one of the five actions: and, or add, subtract, or set on less than. For the instruction beq, the ALU performs subtraction. The Verilog code for the ALU is

```
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

```

### iii. Memory Stage and Write Back Stage

#### 1. Data Memory

This part is used to save the data to a data memory. The data memory must be written on store instructions; hence, data memory has read and write control signals, an address input, and an input for the data to be written into memory.

```

1  module data_memory (
2      input          clk,
3      input          MemRead,
4                      MemWrite,
5      input  [31:0]  address,
6                      write_data,
7      output [31:0]  read_data
8  );
9      parameter      size = 64; // size of data register_memory
10     integer          i;
11     wire  [31:0]     index;
12     reg   [31:0]     register_memory [0:size-1];
13     assign index = address >> 2; // address/4
14     initial begin
15         for (i = 0; i < size; i = i + 1)
16             register_memory[i] = 32'b0;
17         // read_data = 32'b0;
18         // wire can not be set within a initial.
19     end

```

```

20     always @ ( posedge clk ) begin
21         if (MemWrite == 1'b1) begin
22             register_memory[index] = write_data;
23         end
24     end
25     assign read_data = (MemRead == 1'b1)?register_memory[index]:32'b0;
26 endmodule

```

## VI. Control and Data Hazard

### i. EXstage

#### Data Hazard

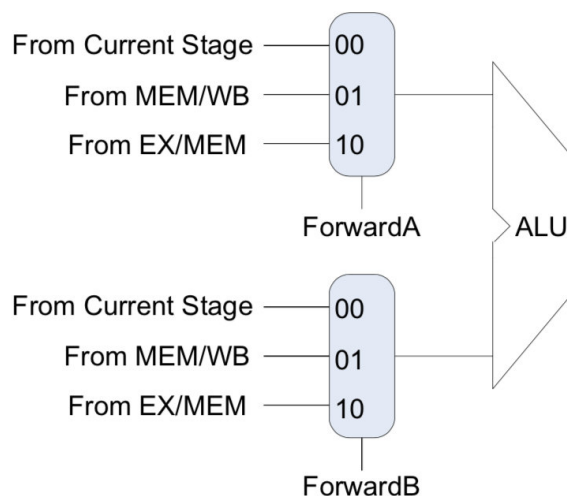
Branch prediction and forwarding help make a computer fast while still getting the right answers. This forwarding control will be in the EX stage, because the ALU forwarding multiplexors are found in that stage. Thus, we must pass the operand register numbers from the ID stage via the ID/EX pipeline register to determine whether to forward values.

```

1  if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
2  and (MEM/WB.RegisterRd = ID/EX.RegisterRs)
3  and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd =
4  ID/EX.RegisterRs)) )
5  ForwardA = 01
6
7  if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)
8  and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd =
9  ID/EX.RegisterRt)) )
10 ForwardB = 01

```

According to the figure below and the logic above, we can get the verilog program belowe.



```

1  module Forwarding(
2      // Input Register
3      input  [4:0] ID_EX_Reg_Rt,
4              ID_EX_Reg_Rs,

```

```

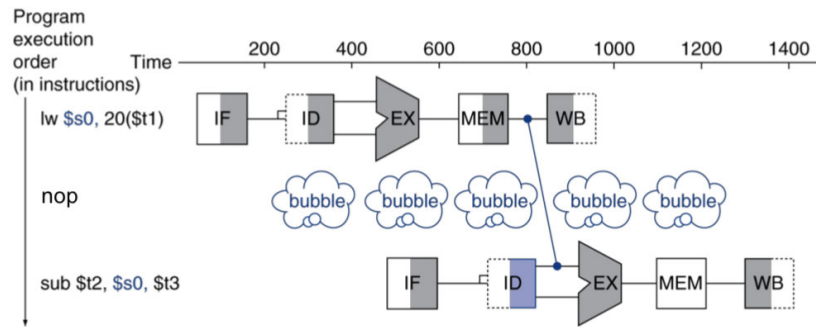
5         IF_ID_Reg_Rs,
6         IF_ID_Reg_Rt,
7         EX_MEM_Reg_Rd,
8         MEM_WB_Reg_Rd,
9     // Input control signal
10    input     EX_MEM_RegWrite,
11           MEM_WB_RegWrite,
12    // Forward for ALU input
13    output reg [1:0] Forward_ALU_A,
14           Forward_ALU_B,
15    // Forward for state reg input
16    output reg Forward_C,
17           Forward_D
18 );
19     always @(*)
20     begin
21         if(EX_MEM_RegWrite && (EX_MEM_Reg_Rd != 0) && (EX_MEM_Reg_Rd == ID_EX_Reg_Rs))
22             Forward_ALU_A = 2'b10;
23         // ID_EX for ALU
24         else if(MEM_WB_RegWrite && (MEM_WB_Reg_Rd != 0) && (MEM_WB_Reg_Rd ==
ID_EX_Reg_Rs))
25             Forward_ALU_A = 2'b01;
26         // MEM_WB for ALU
27         else
28             Forward_ALU_A = 2'b00;
29         // Register for ALU
30
31         if(EX_MEM_RegWrite && (EX_MEM_Reg_Rd != 0) && (EX_MEM_Reg_Rd == ID_EX_Reg_Rt))
32             Forward_ALU_B = 2'b10;
33         else if(MEM_WB_RegWrite && (MEM_WB_Reg_Rd != 0) && (MEM_WB_Reg_Rd ==
ID_EX_Reg_Rt))
34             Forward_ALU_B = 2'b01;
35         else
36             Forward_ALU_B = 2'b00;
37         //Select the data input of ID_EX_State_reg
38         if(EX_MEM_RegWrite && (MEM_WB_Reg_Rd != 0) && (MEM_WB_Reg_Rd == IF_ID_Reg_Rs))
39             Forward_C = 1;
40         else
41             Forward_C = 0;
42
43         if(EX_MEM_RegWrite && (MEM_WB_Reg_Rd != 0) && (MEM_WB_Reg_Rd == IF_ID_Reg_Rt))
44             Forward_D = 1;
45         else
46             Forward_D = 0;
47     end
48 endmodule

```

## ii. ID stage

## Data Hazard

We use the Hazard Detection Unit to detect the *Load-Use Hazard* and stall the pipeline by one clock cycle.



```

1  ID/EX.MemRead and
2  ((ID/EX.RegisterRt == IF/ID.RegisterRs) or
3  (ID/EX.RegisterRt == IF/ID.RegisterRt))

```

With the logic shown above, we can get the verilog program below.

```

1  module hazard_detection(
2      input  [4:0] ID_Reg_Rt,
3              IF_Reg_Rs,
4              IF_Reg_Rt,
5
6      input      EX_MemRead,
7              EX_regWirte,
8              MEM_MemRead,
9              Ins_Beq,
10             Ins_Bne,
11
12     output reg    stall,
13                 flush
14 );
15     initial begin
16         stall = 1'b0;
17         flush = 1'b0;
18     end
19     always @(*) begin
20         if(EX_MemRead&& ID_Reg_Rt&& ((ID_Reg_Rt == IF_Reg_Rs) || (ID_Reg_Rt ==
IF_Reg_Rt)))
21             begin
22                 stall = 1'b1;
23                 flush = 1'b1;
24             end else if(Ins_Beq || Ins_Bne) begin
25                 if(EX_regWirte && ID_Reg_Rt&&((ID_Reg_Rt == IF_Reg_Rs) || (ID_Reg_Rt ==
IF_Reg_Rt))) begin
26                     stall = 1'b1;
27                     flush = 1'b1;

```

```

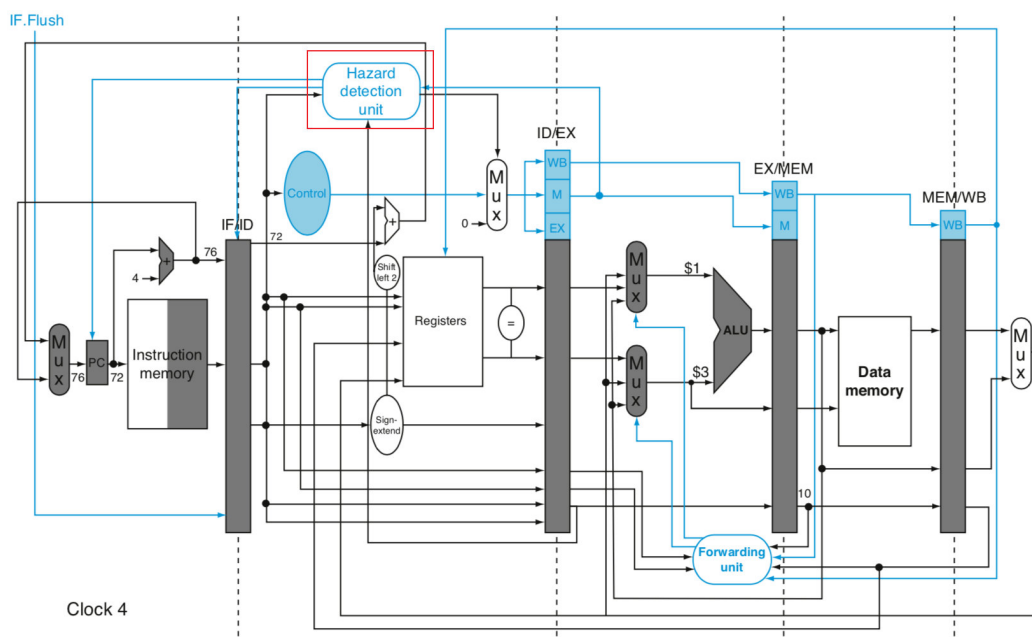
28         end else if (MEM_MemRead &&ID_Reg_Rt&& ((ID_Reg_Rt == IF_Reg_Rs) || (ID_Reg_Rt
== IF_Reg_Rt))) begin
29             stall = 1'b1;
30             flush = 1'b1;
31         end else begin
32             stall = 1'b1;
33             flush = 1'b1;
34         end
35     end else begin
36         stall = 1'b0;
37         flush = 1'b0;
38     end
39 end
40 endmodule

```

## Control Hazard

We have limited our concern to hazards involving arithmetic operations and data transfers. However, there are also pipeline hazards involving branches. An instruction must be fetched at every clock cycle to sustain the pipeline, yet in our design the decision about whether to branch doesn't occur until the MEM pipeline stage. This delay in determining the proper instruction to fetch is called a control hazard or branch hazard, in contrast to the data hazards we have just examined.

We assume branch not taken and use dynamic branch prediction. We only change prediction on two successive mispredictions.



```

1 module HazardDetection (
2     input          branchEqID,
3                   branchNeID,
4                   memReadEX,
5                   regWriteEX,
6                   memReadMEM,
7     input [4:0]   registerRsID,
8                   registerRtID,

```



```

9         registerRtEX,
10        registerRdEX,
11        registerRdMEM,
12    output reg    stall,
13                flush
14 );
15
16    initial begin
17        stall = 1'b0;
18        flush = 1'b0;
19    end
20
21    always @ ( * ) begin
22        if (memReadEX && registerRtEX && (registerRtEX == registerRsID || registerRtEX ==
registerRtID)) begin
23            stall = 1'b1;
24            flush = 1'b1;
25        end else if (branchEqID || branchNeID) begin
26            if (regWriteEX && registerRdEX && (registerRdEX == registerRsID ||
registerRdEX == registerRtID)) begin
27                stall = 1'b1;
28                flush = 1'b1;
29            end else if (memReadMEM && registerRdMEM && (registerRdMEM == registerRsID ||
registerRdMEM == registerRtID)) begin
30                stall = 1'b1;
31                flush = 1'b1;
32            end else begin
33                stall = 1'b0;
34                flush = 1'b0;
35            end
36        end else begin
37            stall = 1'b0;
38            flush = 1'b0;
39        end
40    end
41
42 endmodule

```

## VII. Instruction Implementation

### i. Data tables

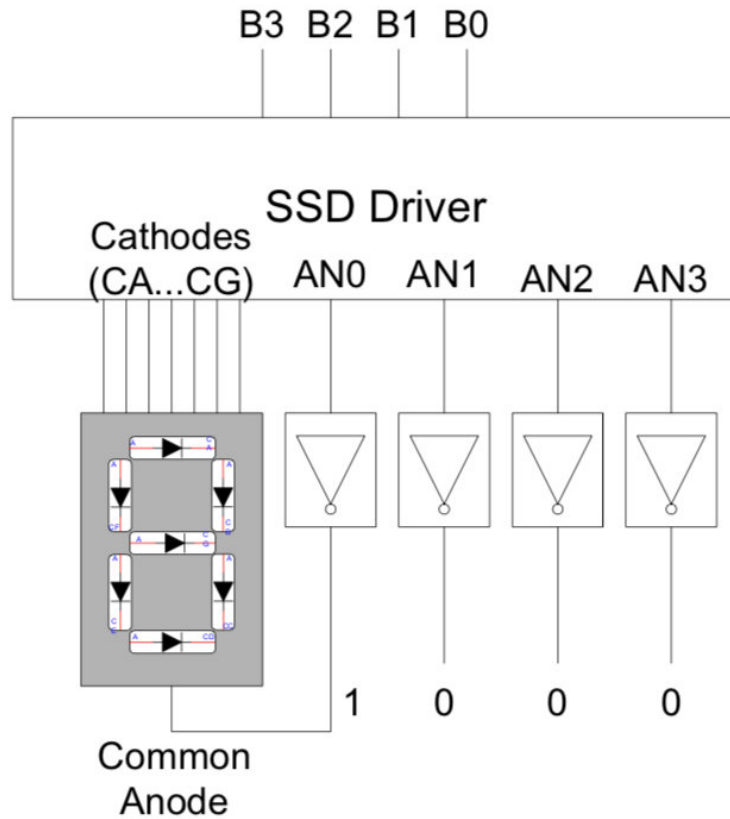
opcode	Operation	ALUOp	funct	ALU Control	ALU function
lw	load word	00	XXXXXX	0010	add
sw	store word				
beq	branch equal	01	XXXXXX	0110	subtract
R-type	add	10	100000	0010	add
	subtract		100010	0110	subtract
	AND		100100	0000	AND
	OR		100101	0001	OR
	set-on-less-than		101010	0111	set-on-less-than

The following table shows the setting of the control lines for each instruction in our design.

	Jump	ALUSrc	RegDst	ALUOp	MemWrite	MemRead	Branch	MemtoReg	RegWrite
R-type	0	0	1	10	0	0	0	0	1
addi	0	1	0	00	0	0	0	0	1
andi	0	1	0	00	0	0	0	0	1
slt	0	1	0	00	0	0	0	0	1
beq	0	0	0	01	0	0	1	0	0
bne	0	0	0	01	0	0	1	0	0
lw	0	1	0	00	0	1	0	1	1
sw	0	1	0	00	1	0	0	0	0
j	1	0	0	00	0	0	0	0	0

## VIII. SSD and Top Module

The Basys 3 board contains a four-digit common anode SSD. The anodes of the seven LEDs forming each digit are tied together into one “common anode” circuit node, but the LED cathodes remain separate. The common anode signals are available as four “digit enable” input signals to the 4-digit display. The cathodes of similar segments on all four displays are connected into seven circuit nodes labeled CA through CG (so, for example, the four d cathodes from the four digits are grouped together into a single circuit node called “CD”). These seven cathode signals are available as inputs to the 4-digit display. This signal connection scheme makes the cathode signals common to all digits but they can only illuminate the segments of a digit whose corresponding anode signal is asserted.



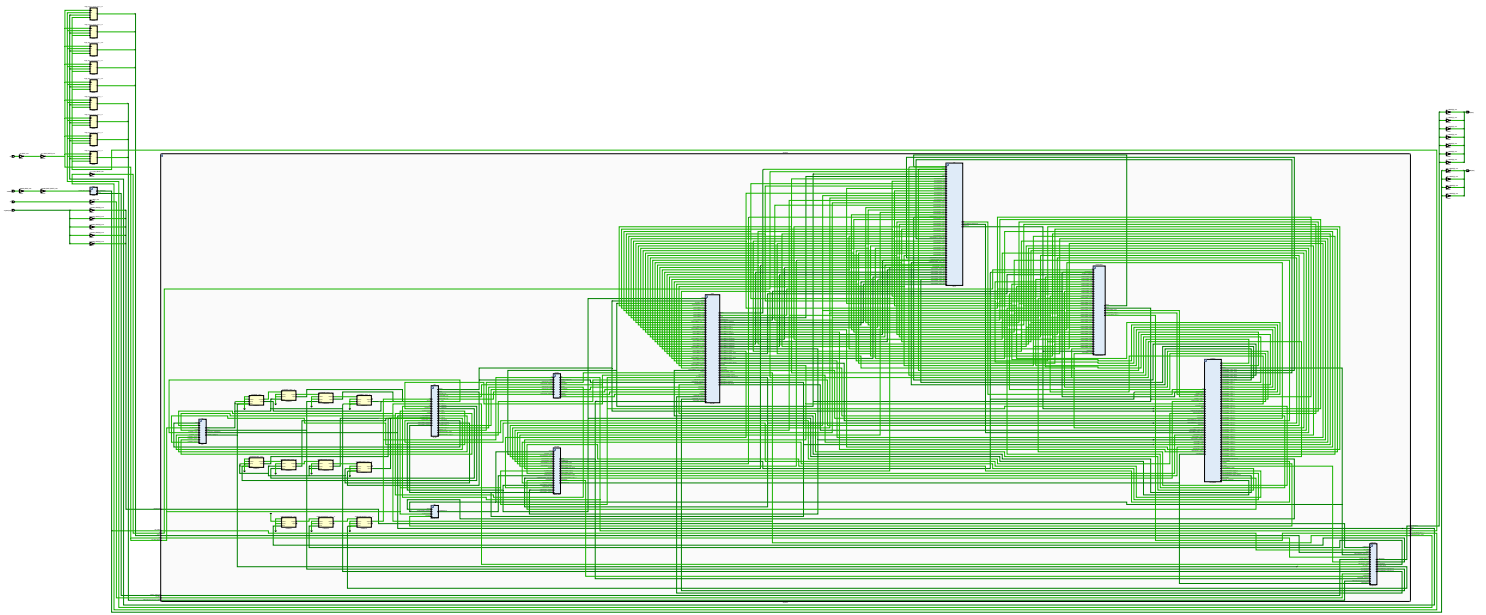
```

1  module SSD(Q,out);
2      input  [3:0] Q;
3      output [6:0] out;
4      reg  [6:0] out;
5      always @(Q) begin
6          case(Q)
7              4'b0000: out = 7'b0000001;
8              4'b0001: out = 7'b1001111;
9              4'b0010: out = 7'b0010010;
10             4'b0011: out = 7'b0000110;
11             4'b0100: out = 7'b1001100;
12             4'b0101: out = 7'b0100100;
13             4'b0110: out = 7'b0100000;
14             4'b0111: out = 7'b0001111;
15             4'b1000: out = 7'b0000000;
16             4'b1001: out = 7'b0000100;
17             default: out = 7'b1111110;
18         endcase
19     end
20 endmodule

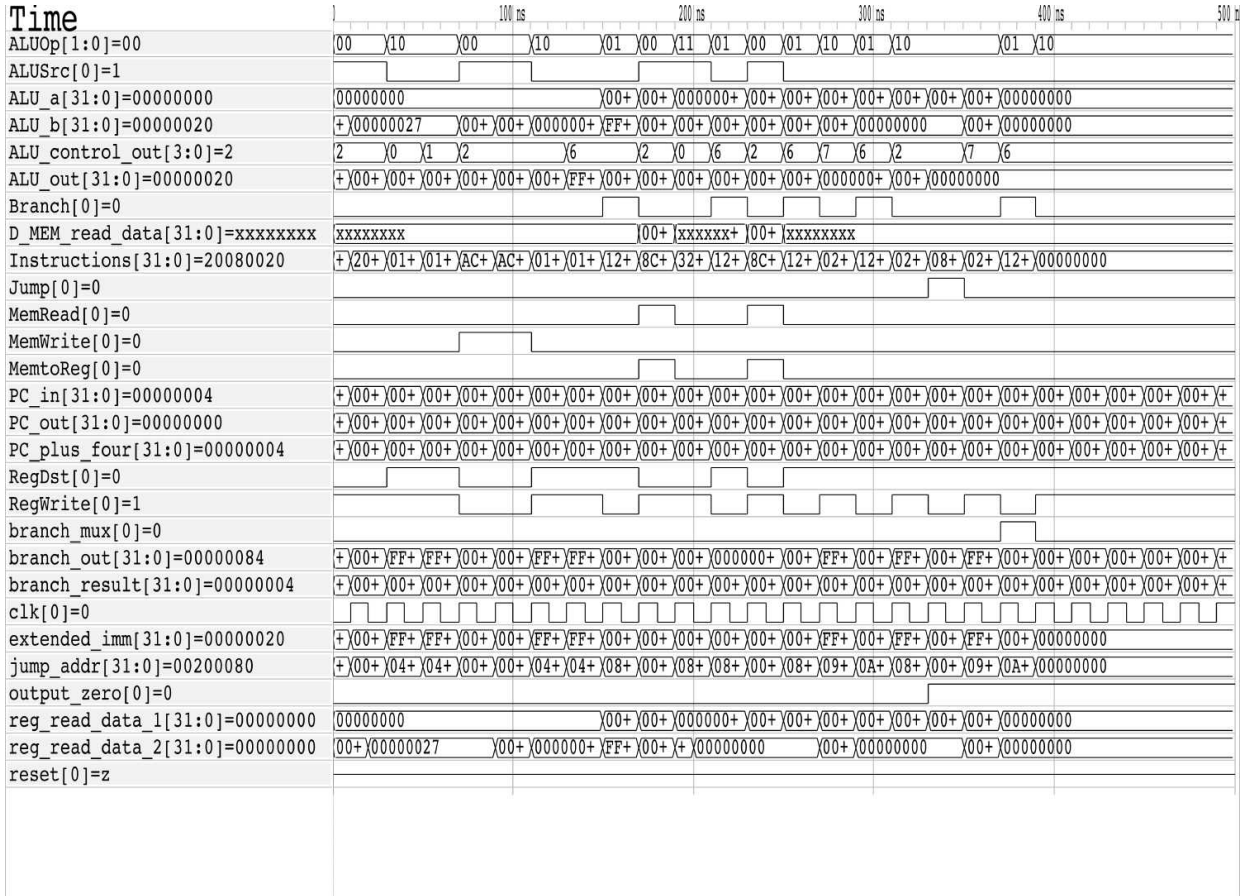
```

## IX. RTL Schematic (Optional)

Please check the RTL schematic on the next two pages.



# X. Textual Result



By running the instruction memory, we can achieve appendix part.

# XI. Conclusion and Discussion

In this project, a MIPS single-cycle processor and a MIPS pipelined processor are modeled, implemented using Verilog and simulated in Vivado. The pipelined processor program is run on a Xilinx FPGA board, which is portal to the Vivado Design Suite. The results on board coincide with the simulation results.

Throughout the implementation process, some unexpected events occurred and time were spent on debugging the modules. The events can be divided into two categories. The following discusses the reasons that led to the events.

## 1. Simulation errors in Verilog HDL

- For the implementation of the 'and' instruction in the ALU module, logical AND '&&' should be used rather than bitwise AND '&'. Using bitwise AND will lead to problematic results. They are two different logical operations.
- In the single cycle processor, due to the rising edge in the first clock cycle, if the first instruction is 'lw', error would occur. Therefore, the program counter starts from '-4' instead of '0'.
- When coding the main processor module, where a large number of wires are connected to run the program, the naming of wires was not paid attention to. This led to error in wire

connections. The reason is that Verilog is a case-sensitive language. For example, 'WireReg' and 'wireReg' actually refer to different wires, resulting in the error.

- The naming of the pipeline register wasn't consistent for different components, some represented the output of a pipeline and some were the input of a pipeline. It led to some errors when simulating the processor too.
- Besides, during the debug process, we learned that the 'assign' statement used for continuous assignment in Verilog can only drive a value to a net but it cannot assign a value to a register.

## 2. Unexpected results on FPGA board

- Because the events in Verilog mostly happen at 'posedge Clock' or 'negedge Clock', when the switch spends more than half a clock cycle time in between the 'on' and 'off' position, the program counter will not be determined and hence leads to varying random numbers.

By accomplishing the project, our understanding of how the central processing unit (CPU) works is deepened through practice. The mechanisms of forwarding and flushing in order to prevent hazards are carefully comprehended during the design process. While theoretical knowledge is the foundation, as programmers, the use of programming language has a series of rules and conventions to follow. Especially for naming, the habit of consistency is important and will save a lot of time in the debug process, no matter what programming project we work on in the future.

## X. Reference

*Hennessy, & JohnL. (1998). Computer organization and design:the hardware/software interface. Morgan Kaufmann Publishers.*

*Patterson, D. A., & Hennessy, J. L. (2014). Computer organization and design, fourth edition. Tailieu Vn.*

## XII. Appendix

### 1. TextualResult

#### *Single*

```
1 LXT2 info: dumpfile single_cycle.vcd opened for output.
2 Textual result of single cycle:
3 =====
4 Time:          0, CLK = 0, PC = 0xffffffffc
5 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
6 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
7 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
8 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
9 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
10 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
11 -----
```

```

12 Time:          0, CLK = 1, PC = 0x00000000
13 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
14 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
15 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
16 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
17 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
18 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
19 -----
20 Time:          1, CLK = 0, PC = 0x00000000
21 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
22 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
23 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
24 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
25 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
26 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
27 -----
28 Time:          1, CLK = 1, PC = 0x00000004
29 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
30 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
31 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
32 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
33 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
34 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
35 -----
36 Time:          2, CLK = 0, PC = 0x00000004
37 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
38 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
39 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
40 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
41 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
42 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
43 -----
44 Time:          2, CLK = 1, PC = 0x00000008
45 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
46 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
47 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
48 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
49 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
50 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
51 -----
52 Time:          3, CLK = 0, PC = 0x00000008
53 [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
54 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
55 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
56 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
57 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
58 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
59 -----
60 Time:          3, CLK = 1, PC = 0x0000000c

```

```

61  [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
62  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
63  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
64  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
65  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
66  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
67  -----
68  Time:          4, CLK = 0, PC = 0x0000000c
69  [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
70  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
71  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
72  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
73  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
74  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
75  -----
76  Time:          4, CLK = 1, PC = 0x00000010
77  [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
78  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
79  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
80  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
81  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
82  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
83  -----
84  Time:          5, CLK = 0, PC = 0x00000010
85  [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
86  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
87  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
88  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
89  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
90  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
91  -----
92  Time:          5, CLK = 1, PC = 0x00000014
93  [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
94  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
95  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
96  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
97  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
98  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
99  -----
100 Time:          6, CLK = 0, PC = 0x00000014
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:          6, CLK = 1, PC = 0x00000018
109 [$s0] = 0x00000037, [$s1] = 0x00000000, [$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, [$t9] = 0x00000000
115  -----
116  Time:           7, CLK = 0, PC = 0x00000018
117  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
118  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
119  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
120  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
121  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
122  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
123  -----
124  Time:           7, CLK = 1, PC = 0x0000001c
125  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
126  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
127  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
128  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
129  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
130  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
131  -----
132  Time:           8, CLK = 0, PC = 0x0000001c
133  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
134  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
135  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
136  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
137  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
138  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
139  -----
140  Time:           8, CLK = 1, PC = 0x00000020
141  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
142  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
143  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
144  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
145  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
146  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
147  -----
148  Time:           9, CLK = 0, PC = 0x00000020
149  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
150  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
151  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
152  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
153  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
154  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
155  -----
156  Time:           9, CLK = 1, PC = 0x00000024
157  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
158  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
```

```

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

```

```

208  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
209  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
210  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
211  -----
212  Time:           13, CLK = 0, PC = 0x00000030
213  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
214  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
215  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
216  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
217  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
218  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
219  -----
220  Time:           13, CLK = 1, PC = 0x00000034
221  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
222  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
223  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
224  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
225  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
226  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
227  -----
228  Time:           14, CLK = 0, PC = 0x00000034
229  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
230  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
231  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
232  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
233  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
234  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
235  -----
236  Time:           14, CLK = 1, PC = 0x00000038
237  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
238  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
239  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
240  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
241  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
242  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
243  -----
244  Time:           15, CLK = 0, PC = 0x00000038
245  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
246  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
247  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
248  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
249  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
250  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
251  -----
252  Time:           15, CLK = 1, PC = 0x0000003c
253  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
254  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
255  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
256  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```

```

257  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
258  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
259  -----
260  Time:          16, CLK = 0, PC = 0x0000003c
261  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
262  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
263  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
264  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
265  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
266  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
267  -----
268  Time:          16, CLK = 1, PC = 0x00000040
269  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
270  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
271  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
272  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
273  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
274  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
275  -----
276  Time:          17, CLK = 0, PC = 0x00000040
277  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
278  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
279  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
280  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
281  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
282  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
283  -----
284  Time:          17, CLK = 1, PC = 0x00000044
285  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
286  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
287  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
288  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
289  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
290  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
291  -----
292  Time:          18, CLK = 0, PC = 0x00000044
293  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
294  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
295  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
296  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
297  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
298  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
299  -----
300  Time:          18, CLK = 1, PC = 0x00000038
301  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
302  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
303  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
304  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
305  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000

```

```
306  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
307  -----
308  Time:          19, CLK = 0, PC = 0x00000038
309  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
310  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
311  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
312  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
313  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
314  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
315  -----
316  Time:          19, CLK = 1, PC = 0x0000003c
317  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
318  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
319  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
320  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
321  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
322  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
323  -----
324  Time:          20, CLK = 0, PC = 0x0000003c
325  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
326  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
327  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
328  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
329  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
330  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
331  -----
332  Time:          20, CLK = 1, PC = 0x0000007c
333  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
334  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
335  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
336  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
337  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
338  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
339  -----
340  Time:          21, CLK = 0, PC = 0x0000007c
341  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
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, [$t9] = 0x00000000
347  -----
348  Time:          21, CLK = 1, PC = 0x00000080
349  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
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 -----
356 Time:          22, CLK = 0, PC = 0x00000080
357 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
358 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
359 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
360 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
361 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
362 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
363 -----
364 Time:          22, CLK = 1, PC = 0x00000084
365 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
366 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
367 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
368 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
369 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
370 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
371 -----
372 Time:          23, CLK = 0, PC = 0x00000084
373 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
374 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
375 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
376 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
377 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
378 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
379 -----
380 Time:          23, CLK = 1, PC = 0x00000088
381 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
382 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
383 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
384 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
385 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
386 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
387 -----
388 Time:          24, CLK = 0, PC = 0x00000088
389 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
390 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
391 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
392 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
393 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
394 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
395 -----
396 ** VVP Stop(0) **
397 ** Flushing output streams.
398 ** Current simulation time is 500000 ticks.
399 > finish
400 ** Continue **

```

## Pipeline

```
1 VCD info: dumpfile pipeline.vcd opened for output.
2 Textual result of pipeline:
3 =====
4 Time:          0, CLK = 0, PC = 0x00000000
5 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
6 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
7 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
8 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
9 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
10 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
11 =====
12 Time:          0, CLK = 1, PC = 0x00000004
13 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
14 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
15 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
16 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
17 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
18 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
19 =====
20 Time:          1, CLK = 0, PC = 0x00000004
21 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
22 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
23 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
24 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
25 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
26 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
27 =====
28 Time:          1, CLK = 1, PC = 0x00000008
29 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
30 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
31 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
32 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
33 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
34 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
35 =====
36 Time:          2, CLK = 0, PC = 0x00000008
37 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
38 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
39 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
40 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
41 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
42 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
43 =====
44 Time:          2, CLK = 1, PC = 0x0000000c
45 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
46 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
47 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
48 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
49 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
```

```

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

```



```

99 =====
100 Time:          6, CLK = 0, PC = 0x00000018
101 [$s0] = 0x00000020, [$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:          6, CLK = 1, PC = 0x0000001c
109 [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
110 [$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 =====
116 Time:          7, CLK = 0, PC = 0x0000001c
117 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
118 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
119 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
120 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
121 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
122 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
123 =====
124 Time:          7, CLK = 1, PC = 0x00000020
125 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
126 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
127 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
128 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
129 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
130 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
131 =====
132 Time:          8, CLK = 0, PC = 0x00000020
133 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
134 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
135 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
136 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
137 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
138 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
139 =====
140 Time:          8, CLK = 1, PC = 0x00000024
141 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
142 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
143 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
144 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
145 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
146 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
147 =====

```

```

148 Time:          9, CLK = 0, PC = 0x00000024
149 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
150 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
151 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
152 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
153 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
154 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
155 =====
156 Time:          9, CLK = 1, PC = 0x00000024
157 [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
158 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
159 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
160 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
161 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
162 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
163 =====
164 Time:          10, CLK = 0, PC = 0x00000024
165 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
166 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
167 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
168 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
169 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
170 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
171 =====
172 Time:          10, CLK = 1, PC = 0x00000028
173 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
174 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
175 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
176 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
177 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
178 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
179 =====
180 Time:          11, CLK = 0, PC = 0x00000028
181 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
182 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
183 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
184 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
185 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
186 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
187 =====
188 Time:          11, CLK = 1, PC = 0x0000002c
189 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
190 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
191 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
192 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
193 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
194 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
195 =====
196 Time:          12, CLK = 0, PC = 0x0000002c

```

```

197  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
198  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
199  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
200  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
201  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
202  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
203  =====
204  Time:          12, CLK = 1, PC = 0x0000002c
205  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
206  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
207  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
208  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
209  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
210  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
211  =====
212  Time:          13, CLK = 0, PC = 0x0000002c
213  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
214  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
215  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
216  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
217  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
218  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
219  =====
220  Time:          13, CLK = 1, PC = 0x00000030
221  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
222  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
223  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
224  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
225  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
226  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
227  =====
228  Time:          14, CLK = 0, PC = 0x00000030
229  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9
230  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
231  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
232  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
233  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
234  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
235  =====
236  Time:          14, CLK = 1, PC = 0x00000030
237  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9
238  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
239  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
240  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
241  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
242  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
243  =====
244  Time:          15, CLK = 0, PC = 0x00000030
245  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9

```

```
246  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
247  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
248  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
249  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
250  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
251  =====
252  Time:          15, CLK = 1, PC = 0x00000034
253  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9
254  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
255  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
256  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
257  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
258  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
259  =====
260  Time:          16, CLK = 0, PC = 0x00000034
261  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
262  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
263  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
264  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
265  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
266  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
267  =====
268  Time:          16, CLK = 1, PC = 0x00000038
269  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
270  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
271  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
272  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
273  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
274  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
275  =====
276  Time:          17, CLK = 0, PC = 0x00000038
277  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
278  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
279  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
280  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
281  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
282  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
283  =====
284  Time:          17, CLK = 1, PC = 0x00000038
285  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
286  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
287  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
288  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
289  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
290  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
291  =====
292  Time:          18, CLK = 0, PC = 0x00000038
293  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
294  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
```

```

295  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
296  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
297  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
298  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
299  =====
300  Time:          18, CLK = 1, PC = 0x00000038
301  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
302  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
303  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
304  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
305  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
306  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
307  =====
308  Time:          19, CLK = 0, PC = 0x00000038
309  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
310  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
311  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
312  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
313  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
314  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
315  =====
316  Time:          19, CLK = 1, PC = 0x0000003c
317  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
318  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
319  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
320  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
321  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
322  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
323  =====
324  Time:          20, CLK = 0, PC = 0x0000003c
325  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
326  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
327  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
328  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
329  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
330  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
331  =====
332  Time:          20, CLK = 1, PC = 0x00000040
333  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
334  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
335  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
336  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
337  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
338  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
339  =====
340  Time:          21, CLK = 0, PC = 0x00000040
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, [$t9] = 0x00000000
347  =====
348  Time:           21, CLK = 1, PC = 0x00000040
349  [$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, [$t9] = 0x00000000
355  =====
356  Time:           22, CLK = 0, PC = 0x00000040
357  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
358  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
359  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
360  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
361  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
362  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
363  =====
364  Time:           22, CLK = 1, PC = 0x00000044
365  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
366  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
367  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
368  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
369  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
370  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
371  =====
372  Time:           23, CLK = 0, PC = 0x00000044
373  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
374  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
375  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
376  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
377  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
378  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
379  =====
380  Time:           23, CLK = 1, PC = 0x00000048
381  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
382  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
383  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
384  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
385  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
386  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
387  =====
388  Time:           24, CLK = 0, PC = 0x00000048
389  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
390  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
391  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
392  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```

```

393  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
394  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
395  =====
396  Time:          24, CLK = 1, PC = 0x00000038
397  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
398  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
399  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
400  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
401  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
402  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
403  =====
404  Time:          25, CLK = 0, PC = 0x00000038
405  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
406  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
407  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
408  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
409  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
410  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
411  =====
412  Time:          25, CLK = 1, PC = 0x0000003c
413  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
414  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
415  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
416  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
417  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
418  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
419  =====
420  Time:          26, CLK = 0, PC = 0x0000003c
421  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
422  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
423  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
424  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
425  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
426  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
427  =====
428  Time:          26, CLK = 1, PC = 0x00000040
429  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
430  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
431  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
432  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
433  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
434  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
435  =====
436  Time:          27, CLK = 0, PC = 0x00000040
437  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
438  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
439  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
440  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
441  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000

```

```
442  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
443  =====
444  Time:          27, CLK = 1, PC = 0x00000040
445  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
446  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
447  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
448  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
449  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
450  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
451  =====
452  Time:          28, CLK = 0, PC = 0x00000040
453  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
454  [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
455  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
456  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
457  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
458  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
459  =====
460  Time:          28, CLK = 1, PC = 0x0000007c
461  [$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, [$t9] = 0x00000000
467  =====
468  Time:          29, CLK = 0, PC = 0x0000007c
469  [$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, [$t9] = 0x00000000
475  =====
476  Time:          29, CLK = 1, PC = 0x00000080
477  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
478  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
479  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
480  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
481  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
482  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
483  =====
484  Time:          30, CLK = 0, PC = 0x00000080
485  [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
486  [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
487  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
488  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
489  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
490  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```



```

491 =====
492 Time:          30, CLK = 1, PC = 0x00000084
493 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
494 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
495 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
496 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
497 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
498 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
499 =====
500 Time:          31, CLK = 0, PC = 0x00000084
501 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
502 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
503 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
504 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
505 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
506 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
507 =====
508 Time:          31, CLK = 1, PC = 0x00000088
509 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
510 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
511 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
512 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
513 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
514 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
515 =====
516 Time:          32, CLK = 0, PC = 0x00000088
517 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
518 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
519 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
520 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
521 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
522 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
523 =====
524 Time:          32, CLK = 1, PC = 0x0000008c
525 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
526 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
527 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
528 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
529 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
530 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
531 =====
532 Time:          33, CLK = 0, PC = 0x0000008c
533 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
534 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
535 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
536 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
537 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
538 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
539 =====

```

```

540 Time:          33, CLK = 1, PC = 0x00000090
541 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
542 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
543 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
544 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
545 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
546 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
547 =====
548 Time:          34, CLK = 0, PC = 0x00000090
549 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
550 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
551 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
552 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
553 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
554 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
555 =====
556 ** VVP Stop(0) **
557 ** Flushing output streams.
558 ** Current simulation time is 700000 ticks.
559 > finish
560 ** Continue **

```

## 2. adder.v

```

1 module adder(
2     input [31:0] a,
3     input [31:0] b,
4     output [31:0] sum
5 );
6     reg [31:0] sum;
7     always @(a or b)
8         begin
9             sum = a + b;
10        end
11 endmodule

```

## 3. ALU\_control.v

```

1 module ALU_control(
2     funct,ALUOp,ALUCtrl
3 );
4     input [5:0] funct;
5     input [1:0] ALUOp;
6     output [3:0] ALUCtrl;
7     reg [3:0] ALUCtrl;

```

```

8   always @ (funct or ALUOp)
9   begin
10      case(ALUOp)
11         2'b00: assign ALUCtrl = 4'b0010;
12         2'b01: assign ALUCtrl = 4'b0110;
13         2'b10:
14             begin
15                 if (funct == 6'b100000)    assign ALUCtrl = 4'b0010;
16                 else if (funct == 6'b100010) assign ALUCtrl = 4'b0110;
17                 else if (funct == 6'b100100) assign ALUCtrl = 4'b0000;
18                 else if (funct == 6'b100101) assign ALUCtrl = 4'b0001;
19                 else if (funct == 6'b101010) assign ALUCtrl = 4'b0111;
20             end
21         2'b11:
22             assign ALUCtrl = 4'b0000;
23     endcase
24 end
25 endmodule

```

## 4. ALU.v

```

1  module ALU(
2      ALUCtrl,a,b,zero,ALU_result
3  );
4      input [3:0] ALUCtrl;
5      input [31:0] a, b;
6      output zero;
7      output [31:0] ALU_result;
8      reg zero;
9      reg [31:0] ALU_result;
10     always @ (a or b or ALUCtrl)
11     begin
12         case (ALUCtrl)
13             4'b0000:
14                 begin
15                     assign ALU_result = a & b;
16                     assign zero = (a & b == 0) ? 1:0;
17                 end
18             4'b0001:
19                 begin
20                     assign ALU_result = a | b;
21                     assign zero = (a | b == 0) ? 1:0;
22                 end
23             4'b0010:
24                 begin
25                     assign ALU_result = a + b;
26                     assign zero = (a + b == 0) ? 1:0;
27                 end
28             4'b0110:

```

```

29     begin
30         assign ALU_result = a - b;
31         assign zero = ( a == b ) ? 1:0;
32     end
33     4'b0111:
34     begin
35         assign ALU_result = (a < b) ? 1:0;
36         assign zero = (a < b) ? 0:1;
37     end
38     default:
39     begin
40         assign ALU_result = a;
41         assign zero = (a == 0) ? 1:0;
42     end
43     endcase
44 end
45 endmodule

```

## 5. data\_memory.v

```

1  module data_memory (
2      input          clk,
3      input          MemRead,
4                      MemWrite,
5      input  [31:0]  address,
6                      write_data,
7      output [31:0]  read_data
8  );
9
10     parameter      size = 64; // size of data register_memory
11     integer         i;
12     wire  [31:0]    index;
13     reg   [31:0]    register_memory [0:size-1];
14
15     assign index = address >> 2; // address/4
16
17
18     initial begin
19         for (i = 0; i < size; i = i + 1)
20             register_memory[i] = 32'b0;
21         // read_data = 32'b0;
22         // wire can not be set within a initial.
23     end
24
25     always @ ( posedge clk ) begin
26         if (MemWrite == 1'b1) begin
27             register_memory[index] = write_data;
28         end
29     end

```

```

30
31     assign read_data = (MemRead == 1'b1)?register_memory[index]:32'b0;
32 endmodule

```

## 6. forwarding\_unit.v

```

1  module Forward (
2      input      [4:0]  registerRsID,
3                  registerRtID,
4                  registerRsEX,
5                  registerRtEX,
6                  registerRdMEM,
7                  registerRdWB,
8      input      regWriteMEM,
9                  regWriteWB,
10     output reg  [1:0] forwardA,
11                  forwardB,
12     output reg  forwardC,
13                  forwardD
14 );
15
16     initial begin
17         forwardA = 2'b00;
18         forwardB = 2'b00;
19         forwardC = 1'b0;
20         forwardD = 1'b0;
21     end
22
23     always @ ( * ) begin
24         if (regWriteMEM && registerRdMEM && registerRdMEM == registerRsEX)
25             forwardA = 2'b10;
26         else if (regWriteWB && registerRdWB && registerRdWB == registerRsEX)
27             forwardA = 2'b01;
28         else
29             forwardA = 2'b00;
30
31         if (regWriteMEM && registerRdMEM && registerRdMEM == registerRtEX)
32             forwardB = 2'b10;
33         else if (regWriteWB && registerRdWB && registerRdWB == registerRtEX)
34             forwardB = 2'b01;
35         else
36             forwardB = 2'b00;
37
38         if (regWriteMEM && registerRdMEM && registerRdMEM == registerRsID)
39             forwardC = 1'b1;
40         else
41             forwardC = 1'b0;
42
43         if (regWriteMEM && registerRdMEM && registerRdMEM == registerRtID)

```

```

44         forwardD = 1'b1;
45     else
46         forwardD = 1'b0;
47     end
48
49 endmodule // Forward

```

## 7. hazard\_detection.v

```

1  module HazardDetection (
2      input          branchEqID,
3                  branchNeID,
4                  memReadEX,
5                  regWriteEX,
6                  memReadMEM,
7      input          [4:0] registerRsID,
8                  registerRtID,
9                  registerRtEX,
10                 registerRdEX,
11                 registerRdMEM,
12     output reg     stall,
13                 flush
14 );
15
16     initial begin
17         stall = 1'b0;
18         flush = 1'b0;
19     end
20
21     always @ ( * ) begin
22         if (memReadEX && registerRtEX && (registerRtEX == registerRsID || registerRtEX ==
registerRtID)) begin
23             stall = 1'b1;
24             flush = 1'b1;
25         end else if (branchEqID || branchNeID) begin
26             if (regWriteEX && registerRdEX && (registerRdEX == registerRsID ||
registerRdEX == registerRtID)) begin
27                 stall = 1'b1;
28                 flush = 1'b1;
29             end else if (memReadMEM && registerRdMEM && (registerRdMEM == registerRsID ||
registerRdMEM == registerRtID)) begin
30                 stall = 1'b1;
31                 flush = 1'b1;
32             end else begin
33                 stall = 1'b0;
34                 flush = 1'b0;
35             end
36         end else begin
37             stall = 1'b0;

```

```

38         flush = 1'b0;
39     end
40 end
41
42 endmodule // HazardDetection

```

## 8. instruction\_memory.v

```

1  module instruction_memory (
2      input      [31:0]  address,
3      output     [31:0]  instruction
4  );
5
6      parameter size = 128; // you can change here, size is the max size of memory
7      integer i;
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.txt"
16     end
17     // Output the memory at address
18     assign instruction = memory[address >> 2];
19
20 endmodule

```

## 9. Mux\_N\_bit.v

```

1  module Mux_N_bit(in1,in2,out,select);
2      parameter N = 32;
3      input [N-1:0] in1,in2;
4      input select;
5      output [N-1:0] out;
6      assign out = select?in2:in1;
7  endmodule

```

## 10. pc.v

```

1  module PC (
2      input      clk,
3              PCWrite,
4      input      [31:0] in,
5      output reg [31:0] out
6  );
7

```

```

8     initial begin
9         out = 32'b0;
10    end
11
12    always @ (posedge clk) begin
13        if (PCWrite)
14            out <= in;
15    end
16
17 endmodule

```

## 11. pipeline.v

```

1  module Pipeline(
2      input clk
3  );
4
5      wire          reset;
6      assign        reset = 1'b0;
7      // Wire in IF stage
8      wire  [31:0]  PC_in_IF,
9                  PC_out_IF,
10                 PC_add4_IF,
11                 instrustion_IF,
12                 branch_result_IF;
13
14     wire          stall,
15                 IF_ID_Write,
16                 IF_ID_flush;
17     // wire in ID stage input of IDEX state regsiter, output of state regsiter start with
EX
18     wire  [31:0]  ID_EX_PC_add4,
19                 ID_EX_PC_add4_res,
20                 ID_EX_instrustion,
21                 ID_EX_reg_read_data_1,
22                 ID_EX_reg_read_data_2,
23                 ID_EX_reg_read_new_data_1,
24                 ID_EX_reg_read_new_data_2,
25                 ID_EX_sign_extend,
26                 ID_EX_jump_addr;
27
28     wire  [4:0]   ID_EX_Reg_Rs,
29                 ID_EX_Reg_Rt,
30                 ID_EX_Reg_Rd;
31
32     wire  [1:0]   ID_EX_alu_op;
33     // Control signals
34     wire          ID_EX_regDst,
35                 ID_EX_branchEq,

```



```

36         ID_EX_branchNeq,
37         ID_EX_memRead,
38         ID_EX_memtoReg,
39         ID_EX_memWrite,
40         ID_EX_aluSrc,
41         ID_EX_regWrite,
42         ID_EX_branch,
43         ID_EX_jump,
44     // comparasions
45         ID_EX_equal_reg_read_data;
46
47     // Wire in EX/MEM
48     wire [31:0] EX_MEM_reg_read_data_1,
49               EX_MEM_reg_read_data_2,
50
51               EX_MEM_sign_extend,
52               // output of mux
53               EX_MEM_alu_in_one,
54               EX_MEM_alu_in_two,
55               EX_MEM_alu_temp_two,
56               EX_MEM_alu_result;
57
58     wire [4:0] EX_MEM_reg_Rs,
59               EX_MEM_reg_Rt,
60               EX_MEM_reg_Rd,
61               EX_reg;
62
63     wire [3:0] EX_MEM_alu_control;
64
65
66     wire [1:0] EX_MEM_alu_op;
67     wire      EX_MEM_regDst,
68               EX_MEM_memRead,
69               EX_MEM_memtoReg,
70               EX_MEM_memWrite,
71               EX_MEM_regWrite,
72               EX_MEM_aluSrc,
73               EX_MEM_aluZero;
74
75     // wire in MEM/WB
76     wire [31:0] MEM_WB_alu_result,
77               MEM_WB_reg_read_data_2,
78               MEM_WB_D_MEM_read_data;
79
80     wire [4:0] MEM_register;
81
82     wire      MEM_WB_memRead,
83               MEM_WB_memtoReg,
84               MEM_WB_memWrite,

```

```

85             MEM_WB_regWrite;
86
87 // wire in WB
88
89 wire    [31:0]  WB_alu_result,
90             WB_D_MEM_read_data,
91             WB_D_MEM_read_addr,
92             WB_write_data;
93
94 wire    [4:0]   WB_register;
95 wire                WB_memoReg,
96             WB_regWrite;
97 // Data hazard
98 wire    [1:0]   forward_A,
99             forward_B;
100
101 wire                forward_C,
102             forward_D;
103
104
105
106 assign PC_add4_IF = PC_out_IF + 4;
107 // assign PC_in_IF = PC_add4_IF;
108 // IF stage
109 assign pc_wirte = ~stall;
110 PC program_counter(
111     .clk(clk),
112     .PCWrite(pc_wirte),
113     .in(PC_in_IF),
114     .out(PC_out_IF)
115 );
116
117 instruction_memory ins_mem(
118     .address(PC_out_IF),
119     .instruction(instruction_IF)
120 );
121
122 // IF/ID
123 wire IF_Flush;
124 assign IF_ID_Write = ~stall;
125 IF_ID_Reg State_IF_ID(
126     .clk(clk),
127     .reset(reset),
128     .PC_plus4_in(PC_add4_IF),
129     .instruction_in(instruction_IF),
130     .IF_ID_Write(IF_ID_Write),
131     .IF_Flush(IF_Flush),
132     .PC_plus4_out(ID_EX_PC_add4),
133     .instruction_out(ID_EX_instruction)

```

```

134     );
135     // ID stage
136     assign ID_EX_Reg_Rs = ID_EX_instruction[25:21];
137     assign ID_EX_Reg_Rt = ID_EX_instruction[20:16];
138     assign ID_EX_Reg_Rd = ID_EX_instruction[15:11];
139
140     wire ins_Beq,ins_Bne;
141
142     Control control(
143         .op_code(ID_EX_instruction[31:26]),
144         .ALUOp(ID_EX_alu_op),
145         .RegDst(ID_EX_regDst),
146         .Jump(ID_EX_jump),
147         .Ins_Beq(ins_Beq),
148         .Ins_Bne(ins_Bne),
149         .MemRead(ID_EX_memRead),
150         .MemtoReg(ID_EX_memtoReg),
151         .MemWrite(ID_EX_memWrite),
152         .ALUSrc(ID_EX_aluSrc),
153         .RegWrite(ID_EX_regWrite)
154     );
155     Registers register(
156         .clk(clk),
157         .regWrite(WB_regWrite),
158         // .write_data(WB_D_MEM_read_data),
159         .read_register_1(ID_EX_Reg_Rs),
160         .read_register_2(ID_EX_Reg_Rt),
161         .write_register(WB_register),
162         .write_data(WB_write_data),
163         .read_data_1(ID_EX_reg_read_data_1),
164         .read_data_2(ID_EX_reg_read_data_2)
165     );
166
167     sign_extension sign(
168         .shortInput(ID_EX_instruction[15:0]),
169         .longOutput(ID_EX_sign_extend)
170     );
171     assign ID_EX_reg_read_new_data_1 = (forward_C)?
MEM_WB_alu_result:ID_EX_reg_read_data_1;
172     assign ID_EX_reg_read_new_data_2 = (forward_D)?
MEM_WB_alu_result:ID_EX_reg_read_data_2;
173     assign ID_EX_PC_add4_res = ID_EX_PC_add4 + (ID_EX_sign_extend<<2);
174     assign ID_EX_jump_addr = {ID_EX_PC_add4[31:28],ID_EX_instruction[25:0],2'b0};
175     assign ID_EX_equal_reg_read_data = (ID_EX_reg_read_new_data_1 ==
ID_EX_reg_read_new_data_2);
176     assign ID_EX_branch = (ins_Beq && ID_EX_equal_reg_read_data) || (ins_Bne && !
ID_EX_equal_reg_read_data);
177
178     assign branch_result_IF = (ID_EX_branch == 1'b0)?PC_add4_IF:ID_EX_PC_add4_res;

```

```

179     assign PC_in_IF = (ID_EX_jump == 1'b0)?branch_result_IF:ID_EX_jump_addr;
180     assign IF_Flush = (ID_EX_jump||ID_EX_branch);
181     wire ID_EX_flush;
182     ID_EX_Reg idex(
183     //    Input
184         .clk(clk),
185         .reset(reset),
186         .flush(ID_EX_flush),
187         .RegDst(ID_EX_regDst),
188         .MemtoReg(ID_EX_memtoReg),
189         .MemRead(ID_EX_memRead),
190         .MemWrite(ID_EX_memWrite),
191         .ALUSrc(ID_EX_aluSrc),
192         .RegWrite(ID_EX_regWrite),
193         .ALUop(ID_EX_alu_op),
194         .reg_read_data_1(ID_EX_reg_read_data_1),
195         .reg_read_data_2(ID_EX_reg_read_data_2),
196         .extended_imm(ID_EX_sign_extend),
197         .IF_ID_Register_Rs(ID_EX_Reg_Rs),
198         .IF_ID_Register_Rt(ID_EX_Reg_Rt),
199         .IF_ID_Register_Rd(ID_EX_Reg_Rd),
200
201     //    output
202         .out_RegDst(EX_MEM_regDst),
203         .out_MemRead(EX_MEM_memRead),
204         .out_MemtoReg(EX_MEM_memtoReg),
205         .out_MemWrite(EX_MEM_memWrite),
206         .out_ALUSrc(EX_MEM_aluSrc),
207         .out_RegWrite(EX_MEM_regWrite),
208         .reg_read_data_1_out(EX_MEM_reg_read_data_1),
209         .reg_read_data_2_out(EX_MEM_reg_read_data_2),
210         .extended_imm_out(EX_MEM_sign_extend),
211         .IF_ID_Register_Rs_out(EX_MEM_reg_Rs),
212         .IF_ID_Register_Rd_out(EX_MEM_reg_Rd),
213         .IF_ID_Register_Rt_out(EX_MEM_reg_Rt),
214         .ALUop_out(EX_MEM_alu_op)
215     );
216
217     HazardDetection hazard(
218         .branchEqID(ins_Beq),
219         .branchNeID(ins_Bne),
220         .memReadEX(EX_MEM_memRead),
221         .regWriteEX(EX_MEM_regWrite),
222         .memReadMEM(MEM_WB_memRead),
223         .registerRsID(ID_EX_Reg_Rs),
224         .registerRtID(ID_EX_Reg_Rt),
225         .registerRtEX(EX_MEM_reg_Rt),
226         .registerRdEX(EX_reg),
227         .registerRdMEM(MEM_register),

```

```

228     .stall(stall),
229     .flush(ID_EX_flush)
230 );
231
232
233
234 // EX
235 ALU_control alu_control(
236     .funct(EX_MEM_sign_extend[5:0]),
237     .ALUOp(EX_MEM_alu_op),
238     .ALUCtrl(EX_MEM_alu_control)
239 );
240
241 Mux_32bit_3to1 forward_A_Mux(
242     .in00(EX_MEM_reg_read_data_1),
243     .in01(WB_write_data),
244     .in10(MEM_WB_alu_result),
245     .control(forward_A),
246     .mux_out(EX_MEM_alu_in_one)
247 );
248
249 Mux_32bit_3to1 forward_B_Mux(
250     .in00(EX_MEM_reg_read_data_2),
251     .in01(WB_write_data),
252     .in10(MEM_WB_alu_result),
253     .control(forward_B),
254     .mux_out(EX_MEM_alu_temp_two)
255 );
256
257 assign EX_MEM_alu_in_two = (EX_MEM_aluSrc == 1'b0)?
EX_MEM_alu_temp_two:EX_MEM_sign_extend;
258 // Register stored in EX/MEM
259 assign EX_reg = (EX_MEM_regDst == 0)?EX_MEM_reg_Rt:EX_MEM_reg_Rd;
260
261 ALU alu(
262     .a(EX_MEM_alu_in_one),
263     .b(EX_MEM_alu_in_two),
264     .ALUCtrl(EX_MEM_alu_control),
265     .ALU_result(EX_MEM_alu_result),
266     .zero(EX_MEM_aluZero)
267 );
268
269 // EX/MEM
270 EX_MEM_Reg exmem(
271     .clk(clk),
272     .reset(reset),
273     .ALU_result(EX_MEM_alu_result),
274     .reg_read_data_2(EX_MEM_alu_temp_two),
275     .ID_EX_Regsiter_Rd(EX_reg),

```

```

276         // output
277         .ALU_result_out(MEM_WB_alu_result),
278         .reg_read_data_2_out(MEM_WB_reg_read_data_2),
279         .EX_MEM_Regsiter_Rd_out(MEM_register)
280     );
281
282     reg [31:0] buffer,buffer1,buffer2,buffer3;
283     always @(posedge clk)begin
284         buffer <= EX_MEM_regWrite;
285         buffer1 <= EX_MEM_memRead;
286         buffer2 <= EX_MEM_memtoReg;
287         buffer3 <= EX_MEM_memWrite;
288     end
289     assign MEM_WB_regWrite = buffer;
290     assign MEM_WB_memRead = buffer1;
291     assign MEM_WB_memtoReg = buffer2;
292     assign MEM_WB_memWrite = buffer3;
293     data_memory dm(
294         .clk(clk),
295         .MemRead(MEM_WB_memRead),
296         .MemWrite(MEM_WB_memWrite),
297         .address(MEM_WB_alu_result),
298         .write_data(MEM_WB_reg_read_data_2),
299         .read_data(MEM_WB_D_MEM_read_data)
300     );
301
302     // MEM/WB
303     MEM_WB_Reg mem_wb(
304         .RegWrite(MEM_WB_regWrite),
305         .MemtoReg(MEM_WB_memtoReg),
306         .D_MEM_read_data_in(MEM_WB_D_MEM_read_data),
307         .ALU_result(MEM_WB_alu_result),
308         .EX_MEM_Reg_Rd(MEM_register),
309         .clk(clk),
310         .reset(reset),
311         .D_MEM_read_data_out(WB_D_MEM_read_data),
312         .D_MEM_read_addr_out(WB_D_MEM_read_addr),
313         .RegWrite_out(WB_regWrite),
314         .MemtoReg_out(WB_memtoReg),
315         .MEM_WB_Reg_Rd(WB_register)
316     );
317
318
319     assign WB_write_data = (WB_memtoReg == 0) ? WB_D_MEM_read_addr:WB_D_MEM_read_data;
320
321     Forward forwad(
322         .registerRsID(ID_EX_Reg_Rs),
323         .registerRtID(ID_EX_Reg_Rt),
324         .registerRsEX(EX_MEM_reg_Rs),

```

```

325     .registerRtEX(EX_MEM_reg_Rt),
326     .registerRdMEM(MEM_register),
327     .registerRdWB(WB_register),
328     .regWriteMEM(MEM_WB_regWrite),
329     .regWriteWB(WB_regWrite),
330     .forwardA(forward_A),
331     .forwardB(forward_B),
332     .forwardC(forward_C),
333     .forwardD(forward_D)
334 );
335
336
337
338 // MEM_WB_RegWrite
339 endmodule
340
341 // 3-to-1 MUX for forwarding
342
343 module Mux_32bit_3to1 (in00, in01, in10, mux_out, control);
344     input [31:0] in00, in01, in10;
345     output [31:0] mux_out;
346     input [1:0] control;
347     reg [31:0] mux_out;
348     always @(in00 or in01 or in10 or control)
349     begin
350         case(control)
351             2'b00:mux_out<=in00;
352             2'b01:mux_out<=in01;
353             2'b10:mux_out<=in10;
354             default: mux_out<=in00;
355         endcase
356     end
357 endmodule
358
359 module HazardDetection (
360     input          branchEqID,
361                   branchNeID,
362                   memReadEX,
363                   regWriteEX,
364                   memReadMEM,
365     input          [4:0] registerRsID,
366                   registerRtID,
367                   registerRtEX,
368                   registerRdEX,
369                   registerRdMEM,
370     output reg     stall,
371                   flush
372 );
373

```

```

374     initial begin
375         stall = 1'b0;
376         flush = 1'b0;
377     end
378
379     always @ ( * ) begin
380         if (memReadEX && registerRtEX && (registerRtEX == registerRsID || registerRtEX ==
registerRtID)) begin
381             stall = 1'b1;
382             flush = 1'b1;
383         end else if (branchEqID || branchNeID) begin
384             if (regWriteEX && registerRdEX && (registerRdEX == registerRsID ||
registerRdEX == registerRtID)) begin
385                 stall = 1'b1;
386                 flush = 1'b1;
387             end else if (memReadMEM && registerRdMEM && (registerRdMEM == registerRsID ||
registerRdMEM == registerRtID)) begin
388                 stall = 1'b1;
389                 flush = 1'b1;
390             end else begin
391                 stall = 1'b0;
392                 flush = 1'b0;
393             end
394         end else begin
395             stall = 1'b0;
396             flush = 1'b0;
397         end
398     end
399
400 endmodule

```

## 12. pipe\_test.v

```

1  module pipeline_tb;
2      integer i = 0;
3      // Inputs
4      reg clk;
5      // Instantiate the Unit Under Test (UUT)
6      Pipeline uut (
7          .clk(clk)
8      );
9      initial begin
10         // Initialize Inputs
11         clk = 0;
12         $dumpfile("pipeline.vcd");
13         $dumpvars(1, uut);
14         $display("Textual result of pipeline:");
15         $display("=====");
16         #700;

```



```

17     $stop;
18     end
19     always #10 begin
20         $display("Time: %d, CLK = %d, PC = 0x%H", i, clk, uut.PC_out_IF);
21         $display("$s0 = 0x%H, [$s1] = 0x%H, [$s2] = 0x%H",
uut.register.register_memory[16], uut.register.register_memory[17],
uut.register.register_memory[18]);
22         $display("$s3 = 0x%H, [$s4] = 0x%H, [$s5] = 0x%H",
uut.register.register_memory[19], uut.register.register_memory[20],
uut.register.register_memory[21]);
23         $display("$s6 = 0x%H, [$s7] = 0x%H, [$t0] = 0x%H",
uut.register.register_memory[22], uut.register.register_memory[23],
uut.register.register_memory[8]);
24         $display("$t1 = 0x%H, [$t2] = 0x%H, [$t3] = 0x%H",
uut.register.register_memory[9], uut.register.register_memory[10],
uut.register.register_memory[11]);
25         $display("$t4 = 0x%H, [$t5] = 0x%H, [$t6] = 0x%H",
uut.register.register_memory[12], uut.register.register_memory[13],
uut.register.register_memory[14]);
26         $display("$t7 = 0x%H, [$t8] = 0x%H, [$t9] = 0x%H",
uut.register.register_memory[15], uut.register.register_memory[24],
uut.register.register_memory[25]);
27         $display("=====");
28         clk = ~clk;
29         if (~clk) i = i + 1;
30     end
31 endmodule

```

### 13. register.v

```

1  module Registers(
2      input      clk,
3              regWrite,
4      input      [4:0]  read_register_1,  read_register_2,
5      input      [4:0]  write_register,
6      input      [31:0] write_data,
7      output     [31:0] read_data_1,read_data_2
8  );
9      parameter size = 32;          // 32-bit CPU, $0 - $31
10     reg [31:0] register_memory [0:size-1];
11     integer i;
12
13     initial begin
14         for (i = 0; i < size; i = i + 1)
15             register_memory[i] = 32'b0;
16     end
17     assign read_data_1 = register_memory[read_register_1];
18     assign read_data_2 = register_memory[read_register_2];
19     always @(negedge clk) begin

```

```

20     if (regWrite == 1)
21         register_memory[write_register] <= write_data;
22     end
23 endmodule // registers

```

## 14. sign\_extension.v

```

1 module sign_extension(
2     shortInput, longOutput
3 );
4     input  [15:0] shortInput;
5     output [31:0] longOutput;
6     // reg [31:0] longOutput;
7     assign longOutput[15:0] = shortInput[15:0];
8     assign longOutput[31:16] = shortInput[15]?16'b1111_1111_1111_1111:16'b0;
9 endmodule

```

## 15. state\_register.v

```

1 module IF_ID_Reg(
2     // Data input
3     input  [31:0]    PC_plus4_in,instruction_in,
4     // Control signal input
5     input          IF_ID_Write, IF_Flush, clk, reset,
6     // output
7     output [31:0]    PC_plus4_out,instruction_out
8 );
9     reg [31:0] PC_plus4_out, instruction_out;
10
11     initial begin
12         PC_plus4_out <= 32'b0;
13         instruction_out <= 32'b0;
14     end
15     always @(posedge clk or posedge reset)
16     begin if (reset == 1'b1|| IF_Flush == 1'b1) begin
17         PC_plus4_out <= 32'b0;
18         instruction_out <= 32'b0;
19     end else if(IF_ID_Write == 1'b1) begin
20         PC_plus4_out <= PC_plus4_in;
21         instruction_out <= instruction_in;
22     end
23     end
24 endmodule
25
26 // Since branch and jump are taken within ID stage, no need to keep Branch jump as well
27 // as the address to this register.
28 module ID_EX_Reg(
29     // *****INPUT*****
30     input          clk,reset,

```

```

30     // hazard control signal
31     input      flush,
32     // Control signal
33     input      RegDst,
34               MemtoReg,
35               // Branch,
36               MemRead,
37               MemWrite,
38               ALUSrc,
39               RegWrite,
40     input  [1:0]  ALUOp,
41     // input  [31:0]  PC_plus4_in,
42     // Read data
43     input  [31:0]  reg_read_data_1,reg_read_data_2,extended_imm,
44     // Register ID
45     input  [4:0]  IF_ID_Register_Rs,IF_ID_Register_Rt, IF_ID_Register_Rd,
46     // *****OUTPUT*****
47     output reg    out_RegDst,
48               out_MemRead,
49               out_MemtoReg,
50               out_MemWrite,
51               out_ALUSrc,
52               out_RegWrite,
53     output reg [31:0]  reg_read_data_1_out,reg_read_data_2_out, extended_imm_out,
54     // PC_plus4_out, jump_address_out,
55     output reg [4:0]
IF_ID_Register_Rs_out,IF_ID_Register_Rd_out,IF_ID_Register_Rt_out,
56     output reg [1:0]  ALUOp_out
57
58 );
59     // For ID_Flush == 1, which means that lw occurs and cannot be solved with
forwarding, set WB, MEM, EX control signal to 0.
60     initial begin
61         out_RegDst      = 1'b0;
62         // out_Jump      = 1'b0;
63         // out_Branch    = 1'b0;
64         out_MemRead     = 1'b0;
65         out_MemtoReg    = 1'b0;
66         out_MemWrite    = 1'b0;
67         out_ALUSrc     = 1'b0;
68         out_RegWrite    = 1'b0;
69         ALUOp_out       = 2'b0;
70         reg_read_data_1_out = 32'b0;
71         reg_read_data_2_out = 32'b0;
72         // extended_imm_out = 32'b0;
73         IF_ID_Register_Rd_out = 5'b0;
74         IF_ID_Register_Rs_out = 5'b0;
75         IF_ID_Register_Rt_out = 5'b0;
76         // funct_out     = 6'b0;

```

```

77     end
78
79     always @(posedge clk or posedge reset)
80     begin
81         if (reset == 1'b1) begin
82             out_RegDst      = 1'b0;
83             // out_Jump      = 1'b0;
84             // out_Branch    = 1'b0;
85             out_MemRead     = 1'b0;
86             out_MemtoReg    = 1'b0;
87             out_MemWrite    = 1'b0;
88             out_ALUSrc      = 1'b0;
89             out_RegWrite    = 1'b0;
90             ALUop_out       = 2'b0;
91             reg_read_data_1_out = 32'b0;
92             reg_read_data_2_out = 32'b0;
93             extended_imm_out  = 32'b0;
94             IF_ID_Register_Rd_out = 5'b0;
95             IF_ID_Register_Rs_out = 5'b0;
96             IF_ID_Register_Rt_out = 5'b0;
97         end else if(flush) begin
98             out_RegWrite = 1'b0;
99             out_MemtoReg = 1'b0;
100            // out_Branch    = 1'b0;
101            out_MemRead  = 1'b0;
102            out_MemWrite = 1'b0;
103            // out_Jump    = 1'b0;
104            out_ALUSrc   = 1'b0;
105            out_RegDst   = 1'b0;
106            ALUop_out    = 2'b0;
107        end else begin
108            out_RegWrite <= RegWrite;
109            out_MemtoReg <= MemtoReg;
110            // out_Branch = Branch;
111            out_MemRead <= MemRead;
112            out_MemWrite <= MemWrite;
113            // out_Jump = Jump;
114            out_RegDst <= RegDst;
115            out_ALUSrc <= ALUSrc;
116            ALUop_out <= ALUop;
117            // jump_addr_out = jump_address;
118            // PC_plus4_out = PC_plus4_in;
119            reg_read_data_1_out <= reg_read_data_1;
120            reg_read_data_2_out <= reg_read_data_2;
121            extended_imm_out <= extended_imm;
122            IF_ID_Register_Rs_out <= IF_ID_Register_Rs;
123            IF_ID_Register_Rt_out <= IF_ID_Register_Rt;
124            IF_ID_Register_Rd_out <= IF_ID_Register_Rd;
125            // funct_out = funct;

```

```

126         end
127     end
128 endmodule
129
130
131 module EX_MEM_Reg(
132     input      clk,reset,
133     // Hazard signal,
134     // Flush in ID/EX will keep the control signal here zero and do not need flush signal
135     any more. For this stage, just store information every posedge.
136     // Control Signal
137     RegWrite,
138     MemtoReg,
139     // Branch,
140     MemRead,
141     MemWrite,
142     // Jump,
143     output reg RegWrite_out,
144     MemtoReg_out,
145     // Branch_out,
146     MemRead_out,
147     MemWrite_out,
148     // Jump_out,
149     // Data
150     input      [31:0] ALU_result,reg_read_data_2,
151     output reg [31:0] ALU_result_out, reg_read_data_2_out,
152     input      [4:0] ID_EX_Regsiter_Rd,
153     output reg [4:0] EX_MEM_Regsiter_Rd_out
154     // No need for ALU zero
155     // input      ALU_zero,
156     // output reg  ALU_zero_out
157 );
158
159 initial begin
160     ALU_result_out = 32'b0;
161     reg_read_data_2_out = 32'b0;
162     EX_MEM_Regsiter_Rd_out = 5'b0;
163     // ALU_zero_out = 1'b0;
164     RegWrite_out = 1'b0;
165     MemtoReg_out = 1'b0;
166     MemRead_out = 1'b0;
167     MemWrite_out = 1'b0;
168
169 end
170
171 always @(posedge clk or posedge reset)
172 begin
173     if(reset == 1'b1)
174     begin
175         ALU_result_out = 32'b0;
176         reg_read_data_2_out = 32'b0;

```

```

174         EX_MEM_Regsiter_Rd_out = 5'b0;
175         RegWrite_out = 1'b0;
176         MemtoReg_out = 1'b0;
177         MemRead_out = 1'b0;
178         MemWrite_out = 1'b0;
179     end
180     else begin
181         MemRead_out = MemRead;
182         RegWrite_out = RegWrite;
183         MemWrite_out = MemWrite;
184         MemtoReg_out = MemtoReg;
185         ALU_result_out = ALU_result;
186         reg_read_data_2_out = reg_read_data_2;
187         EX_MEM_Regsiter_Rd_out = ID_EX_Regsiter_Rd;
188     end
189 end
190 endmodule
191
192 module MEM_WB_Reg(
193     // *****INPUT*****
194     input      RegWrite,MemtoReg,
195     input  [31:0] D_MEM_read_data_in,ALU_result,
196     input  [4:0]  EX_MEM_Reg_Rd,
197     input      clk,reset,
198     // *****OUTPUT*****
199     output reg   RegWrite_out,MemtoReg_out,
200     output reg  [31:0] D_MEM_read_data_out,D_MEM_read_addr_out,
201     output reg  [4:0]  MEM_WB_Reg_Rd
202 );
203     initial begin
204         RegWrite_out = 1'b0;
205         MemtoReg_out = 1'b0;
206         MEM_WB_Reg_Rd = 5'b0;
207         D_MEM_read_addr_out = 32'b0;
208         D_MEM_read_data_out = 32'b0;
209     end
210     always @(posedge clk)
211     begin
212         RegWrite_out <= RegWrite;
213         MemtoReg_out <= MemtoReg;
214         MEM_WB_Reg_Rd <= EX_MEM_Reg_Rd;
215         D_MEM_read_addr_out <= ALU_result;
216         D_MEM_read_data_out <= D_MEM_read_data_in;
217     end
218 endmodule

```

## 16. control.v

```

1  module Control (
2      input      [5:0]  op_code,
3      output reg  [1:0]  ALUOp,
4      output reg  RegDst,
5                      Jump,
6                      Ins_Beq,
7                      Ins_Bne,
8                      MemRead,
9                      MemtoReg,
10                     MemWrite,
11                     ALUSrc,
12                     RegWrite
13 );
14
15     initial begin
16         RegDst      = 1'b0;
17         Jump        = 1'b0;
18         Ins_Beq     = 1'b0;
19         Ins_Bne     = 1'b0;
20         MemtoReg    = 1'b0;
21         MemWrite    = 1'b0;
22         ALUSrc      = 1'b0;
23         RegWrite    = 1'b0;
24         ALUOp       = 2'b00;
25     end
26
27     always @ ( op_code ) begin
28         case (op_code)
29             6'b000000: begin // R-type
30                 RegDst      <= 1'b1;
31                 Jump        <= 1'b0;
32                 Ins_Beq     <= 1'b0;
33                 Ins_Bne     <= 1'b0;
34                 MemRead     <= 1'b0;
35                 MemtoReg    <= 1'b0;
36                 MemWrite    <= 1'b0;
37                 ALUSrc      <= 1'b0;
38                 RegWrite    <= 1'b1;
39                 ALUOp       <= 2'b10;
40             end
41             6'b000010: begin // j
42                 RegDst      <= 1'b1;
43                 Jump        <= 1'b1;
44                 Ins_Beq     <= 1'b0;
45                 Ins_Bne     <= 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         Ins_Beq     <= 1'b1;
57         Ins_Bne     <= 1'b0;
58         MemRead     <= 1'b0;
59         MemtoReg    <= 1'b0;
60         MemWrite    <= 1'b0;
61         ALUSrc      <= 1'b0;
62         RegWrite    <= 1'b0;
63         ALUOp       <= 2'b01;
64     end
65     6'b000100: begin // bne
66         RegDst      <= 1'b1;
67         Jump        <= 1'b0;
68         Ins_Beq     <= 1'b0;
69         Ins_Bne     <= 1'b1;
70         MemRead     <= 1'b0;
71         MemtoReg    <= 1'b0;
72         MemWrite    <= 1'b0;
73         ALUSrc      <= 1'b0;
74         RegWrite    <= 1'b0;
75         ALUOp       <= 2'b01;
76     end
77     6'b001000: begin // addi
78         RegDst      <= 1'b0;
79         Jump        <= 1'b0;
80         Ins_Beq     <= 1'b0;
81         Ins_Bne     <= 1'b0;
82         MemRead     <= 1'b0;
83         MemtoReg    <= 1'b0;
84         MemWrite    <= 1'b0;
85         ALUSrc      <= 1'b1;
86         RegWrite    <= 1'b1;
87         ALUOp       <= 2'b00;
88     end
89     6'b001100: begin // andi
90         RegDst      <= 1'b0;
91         Jump        <= 1'b0;
92         Ins_Beq     <= 1'b0;
93         Ins_Bne     <= 1'b0;
94         MemRead     <= 1'b0;
95         MemtoReg    <= 1'b0;
96         MemWrite    <= 1'b0;
97         ALUSrc      <= 1'b1;
98         RegWrite    <= 1'b1;

```



```
99         ALUOp         <= 2'b11;
100     end
101     6'b100011: begin // lw
102         RegDst         <= 1'b0;
103         Jump           <= 1'b0;
104         Ins_Beq        <= 1'b0;
105         Ins_Bne        <= 1'b0;
106         MemRead        <= 1'b1;
107         MemtoReg       <= 1'b1;
108         MemWrite       <= 1'b0;
109         ALUSrc         <= 1'b1;
110         RegWrite       <= 1'b1;
111         ALUOp         <= 2'b00;
112     end
113     6'b101011: begin // sw
114         RegDst         <= 1'b0;
115         Jump           <= 1'b0;
116         Ins_Beq        <= 1'b0;
117         Ins_Bne        <= 1'b0;
118         MemRead        <= 1'b0;
119         MemtoReg       <= 1'b0;
120         MemWrite       <= 1'b1;
121         ALUSrc         <= 1'b1;
122         RegWrite       <= 1'b0;
123         ALUOp         <= 2'b00;
124     end
125
126     default: ;
127 endcase
128 end
129 endmodule
```