

**Ve 370 Project 2**  
**Simulation and FPGA Implementation of a Single-cycle and  
Pipelined CPU Design**  
**Final Report**

**Project Duration**

October 17, 2013 – November 28, 2013

**Prepared by**

Qian Wang (5113709012)  
Canchao Duan (5113709185)  
Yanrong Li (5113709246)

*Undergraduate Student  
Ve370*

*UM-SJTU Joint Institute*

**Prepared for**

Dr. Gang Zheng  
*Instructor for Ve370, UM-SJTU Joint Institute*

Mr. Tianyuan Zhao  
*Teaching Assistant for Ve370, UM-SJTU Joint Institute*

Ms. Yue Zheng  
*Teaching Assistant for Ve370, UM-SJTU Joint Institute*

**Date of Submission**

November 28, 2013

# Content

<b>1. Objective</b>	<b>3</b>
<b>2. Background</b>	<b>3</b>
<b>3. Design</b>	<b>3</b>
<b>3.1 Single-cycle CPU</b>	<b>3</b>
3.1.1 Program Counter (PC)	4
3.1.2 Instruction Memory	4
3.1.3 Control Unit	4
3.1.4 Register File	5
3.1.5 Arithmetic Logic Center (ALU)	6
3.1.6 ALU Control	7
3.1.7 Data Memory	8
3.1.8 Structure of Single-cycle CPU	8
<b>3.2 Pipelined CPU</b>	<b>9</b>
3.2.1 Stage Registers	9
3.2.2 Forwarding Unit	11
3.2.3 Hazard Detection Unit	12
3.2.4 Structure of Pipelined CPU	13
<b>3.3 FPGA Board Implementation</b>	<b>13</b>
<b>4. Testing and Results</b>	<b>14</b>
4.1 Testing Instructions	14
4.2 Single-cycle CPU Software Simulation	14
4.3 Pipelined CPU Software Simulation	72
4.4 Hardware Implementation	130
<b>5. Discussions</b>	<b>131</b>
<b>6. Conclusion</b>	<b>132</b>
<b>Appendices</b>	<b>133</b>
Appendix A: Source Code of Single-cycle CPU	133
Appendix B: Single-cycle CPU Test Bench	141
Appendix C: Source Code of Pipelined CPU and Hardware Implementation Unit	142
Appendix D: Pipelined CPU Test Bench	160
Appendix E: Simulation Result for TA's Test Case with Single-cycle CPU	161
Appendix F: Simulation Result for TA's Test Case with Pipelined CPU	218

# 1. Objective

- To deepen understanding to single-cycle CPU and pipelined CPU.
- To practice building a single-cycle CPU and pipelined CPU which can implement a basic subset of MIPS instructions including “add”, “sub”, “and”, “or”, “addi”, “beq”, “slt”, “lw”, “sw” and “j” using Verilog with Xilinx ISE Design Suite.
- To simulate the behavior of the CPU using Isim.
- To implement the CPU on a Xilinx FPGA board.

## 2. Background

**CPU** A central processing unit (CPU) is the hardware in a computer that stores a part of the being-executed program and being-used data, and performs arithmetical, logical and I/O functions. The CPU execution is driven by a periodical clock, and the simplest one is called single-cycle CPU, which execute one instruction per clock cycle. However, this kind of CPU has low efficiency because the clock cycle is determined by the instruction that has the longest execution time. To enhance the performance, another kind of CPU is invented: pipelined CPU. In a pipelined CPU, the execution of a piece of instruction is divided into five stages, and each stage possesses one clock cycle. It seems the execution time per instruction is stretched. However, there are two differences: First, in the same clock cycle, different parts of CPU are actually executing stages from different instructions, so at most five instructions can be executed at the same time. Second, each clock cycle is much shorter than that of a single-cycle CPU, because the clock cycle is determined by the longest stage of one instruction, and it is only a small part of a whole complete instruction. Therefore, the performance of a pipelined CPU is much better than a single-cycle CPU.

**MIPS** MIPS is a reduced instruction set computer (RISC) instruction set architecture (ISA). It belongs to assembly languages (a kind of low-level programming languages where each instruction is corresponding to a piece of machine code instruction).

**Verilog** Verilog is a hardware description language used to program the programmable electronic systems. In Verilog, the basic function unit is called module, which can implement one or more specific function and can be treated as an element in the logical combination circuits. The hardware programming is quite different from software programming. It is just like connecting circuits without much care about the order of the instructions.

**FPGA board** A field-programmable gate array (FPGA) is an integrated circuit which can be controlled, or rearranged, by hardware programming to implement different functions. We can use Verilog or VHDL to write programs and burn them to the board, then the logic gate array will be programmed and the circuit will be able to execute the program.

In this project we will use Verilog to write a CPU program, both single-cycle and pipelined. The CPU program will be simulated with a piece of MIPS testing instructions stored in the Instruction Memory. Finally, we will burn the program into an FPGA board to implement it.

## 3. Design

### 3.1 Single-cycle CPU

### 3.1.1 Program Counter (PC)

The program counter counts the address of the instructions stored in the instruction memory. The PC will add 4 to itself automatically every time it fetches one instruction, except for instructions like jump or branch-on-equal which require the PC to change its value to the address of the target instruction.

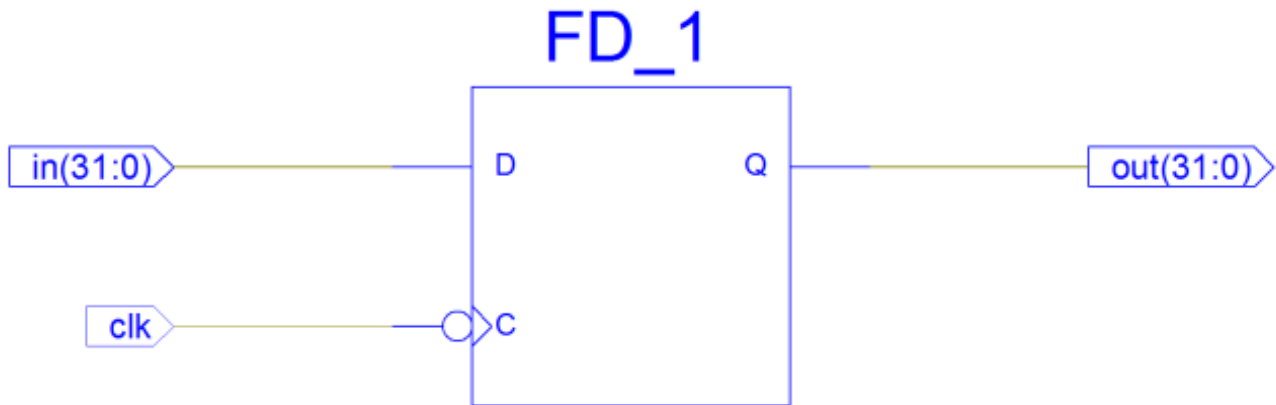


Fig. 1 RTL Schematic for Program Counter

### 3.1.2 Instruction Memory

The instruction memory is a 128\*32-bit array holding instruction codes. The input is the byte address, and the output is the target instruction.

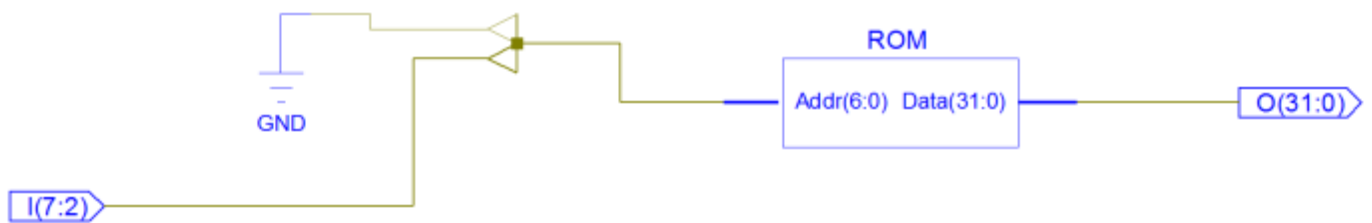


Fig. 2 RTL Schematic for Instruction Memory

### 3.1.3 Control Unit

The control unit produces control signals according to the specific instruction fetched from the instruction memory. These control signals will be output and command the elements to select or operate on data, thus implement the function of the instruction.

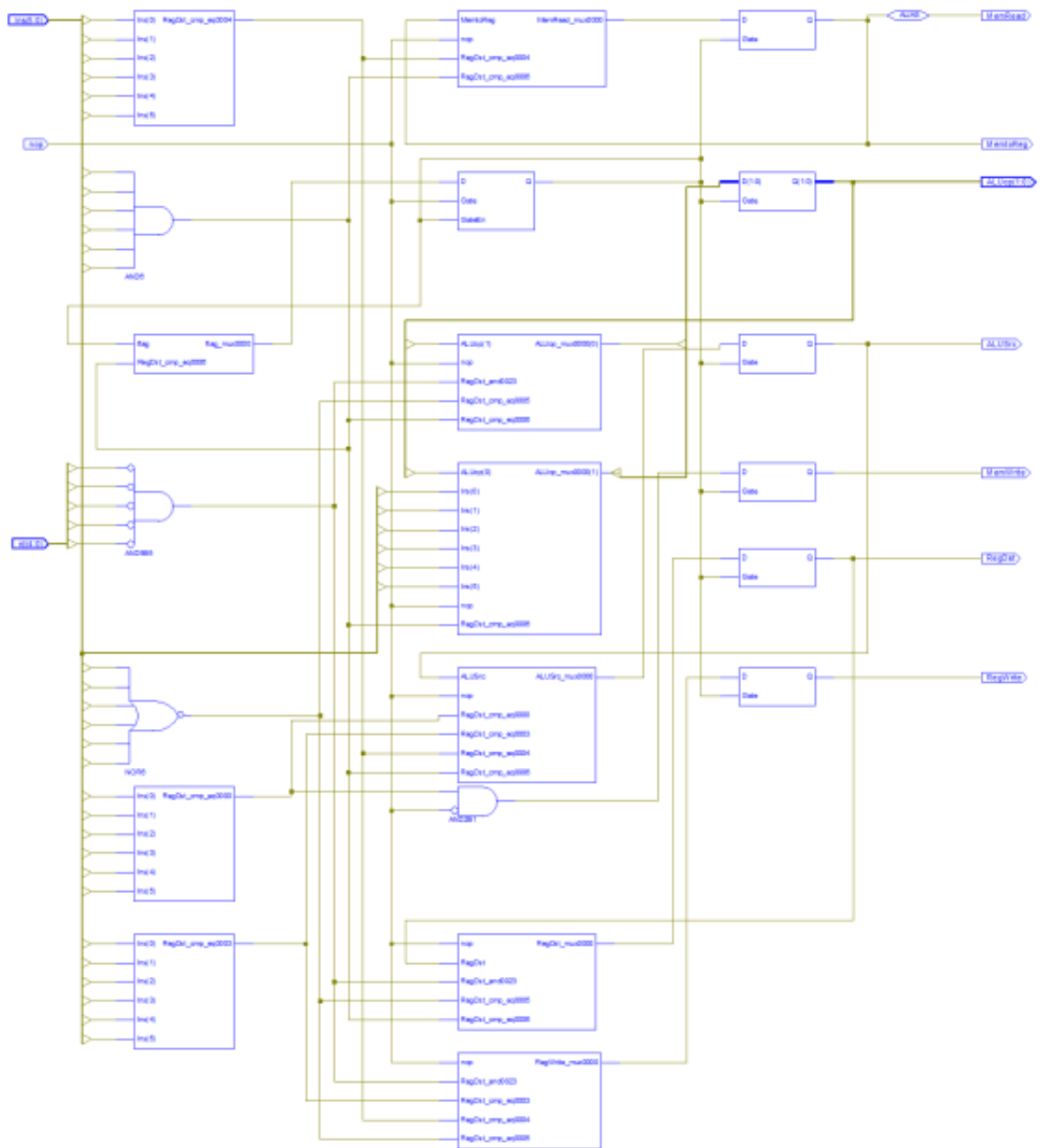


Fig. 3 RTL Schematic for Control Unit

### 3.1.4 Register File

The register file is implemented as a 32\*32 array, and each line is a register storing a 32-bit word. The data can be fetched, modified and overwritten. In single-cycle CPU, all these operations are done in one clock cycle.

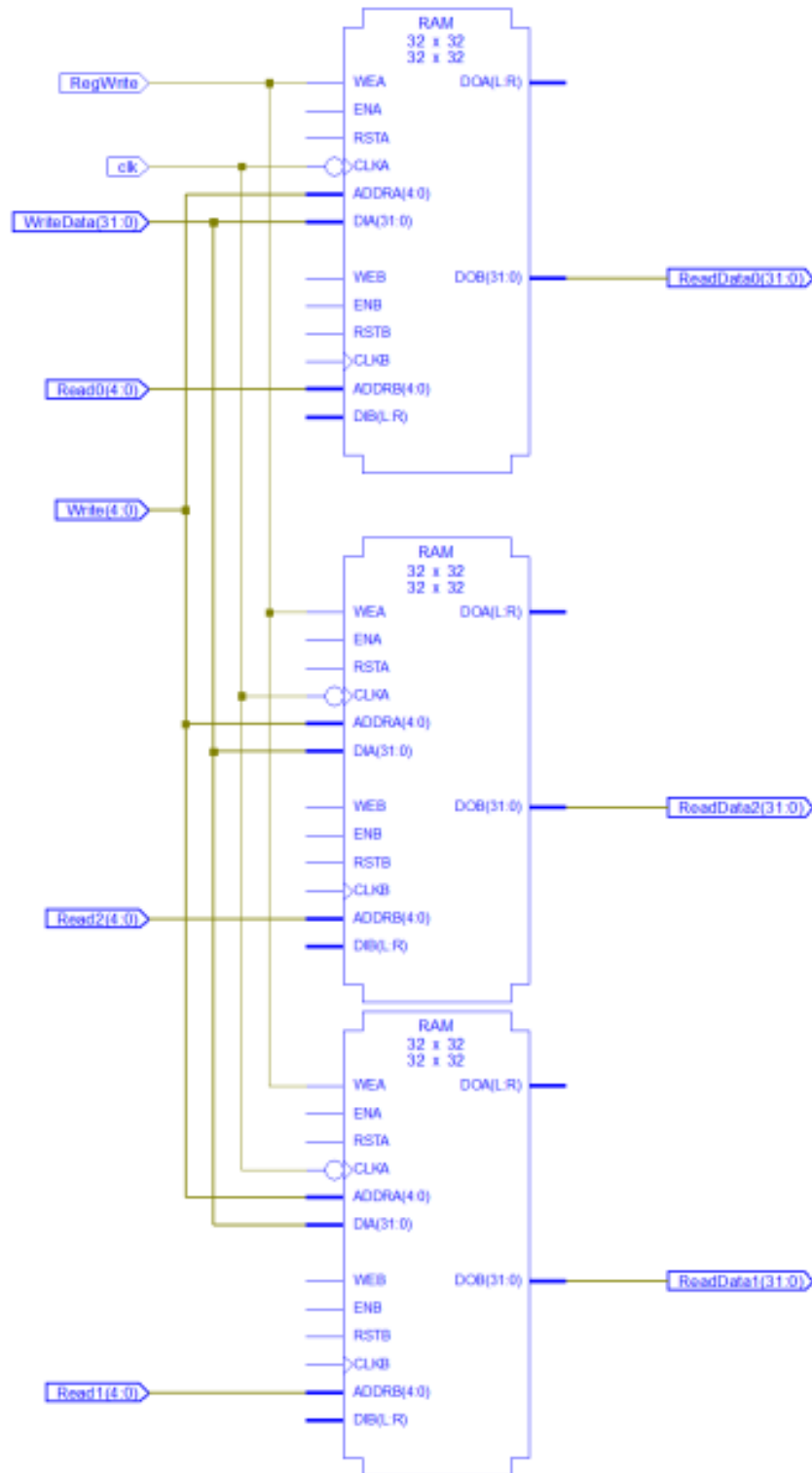


Fig. 4 RTL Schematic for Register File

### 3.1.5 Arithmetic Logic Center (ALU)

A combinational logic circuit is built to implement an ALU. This ALU contains the following basic operations: addition, subtraction, logic and, logic or, set-on-smaller-than. It takes a four-bit ALU control signal, one or two data as the input and does arithmetic or logical operations on the data to produce a result. The “zero” denotes whether two inputs are equal when they perform “sub” or “beq” instructions.

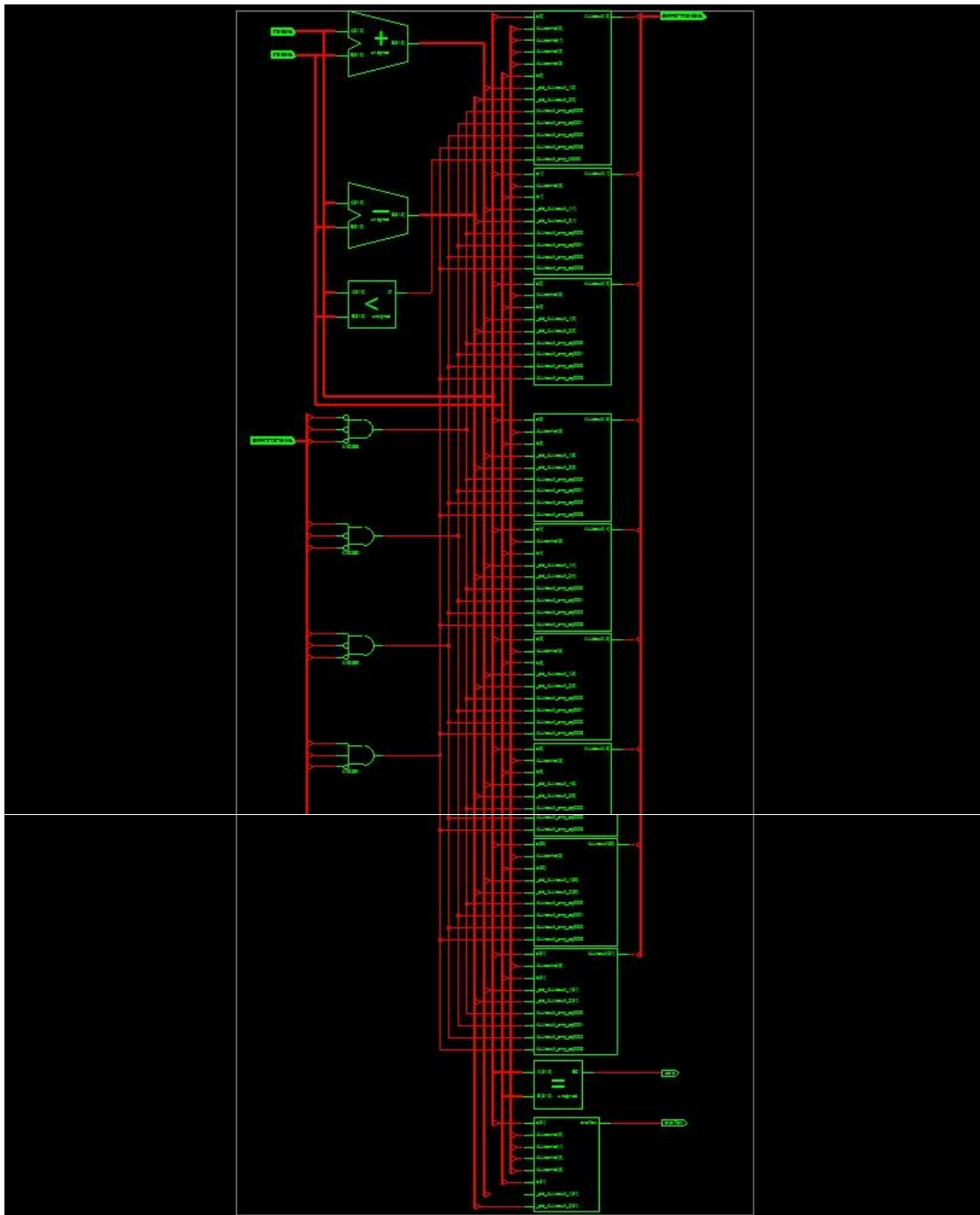


Fig. 5 Key Parts of RTL Schematic for ALU

### 3.1.6 ALU Control

The ALU shown above needs function selection, so that we can obtain the desired results from it. Therefore, we need to add an ALU control unit. This unit takes “funct” part of the instruction and the signal ALUop produced from the control unit as input, and produce a 4-bit ALU control signal. By this signal, the ALU will be able to select the correct arithmetic or logical operation.

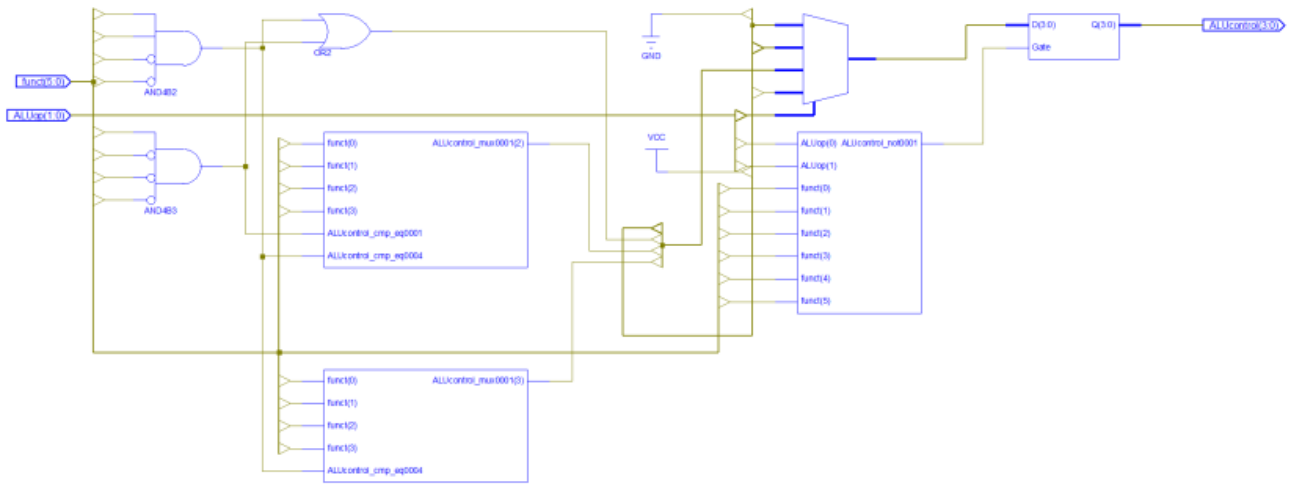


Fig. 6 RTL Schematic for ALU Control

### 3.1.7 Data Memory

Data Memory is a 64\*32 array holding data. The data can be stored and read. It takes data, addresses and control signals as inputs. If MemWrite == 1, the data will be written into the corresponding position in the memory. If MemRead == 1, the data memory will output a datum stored in the corresponding address. What's more, the writing operation is driven by the falling edge of the clock.

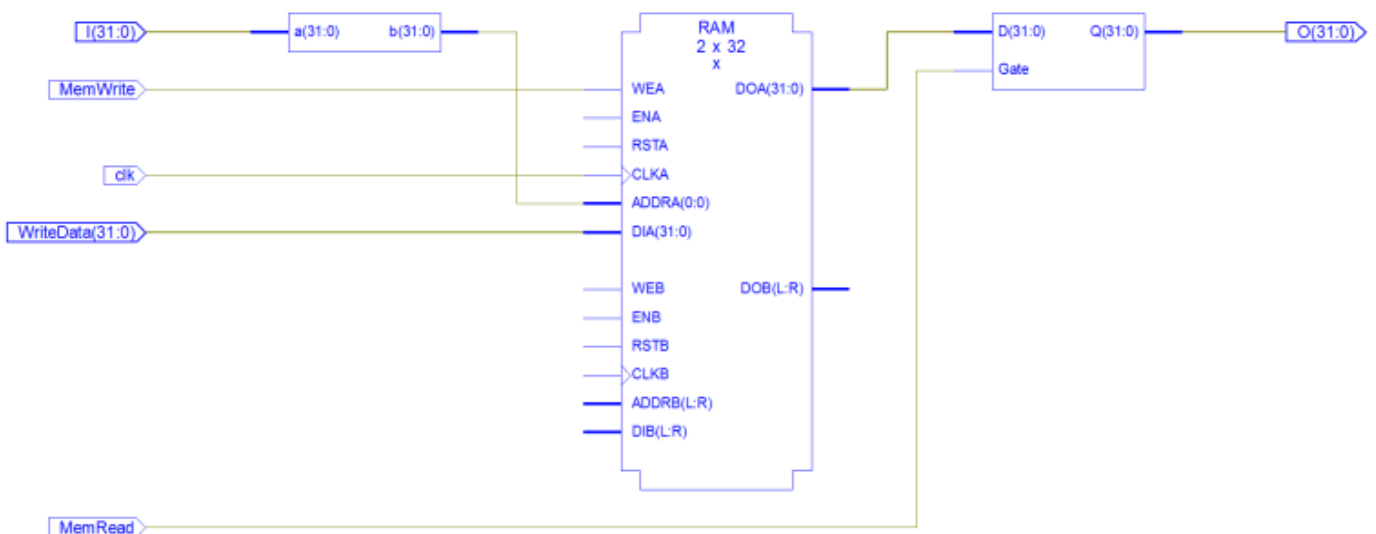


Fig. 7 RTL Schematic for Data Memory

### 3.1.8 Structure of Single-cycle CPU

The complete RTL schematic of a single-cycle CPU is shown below:



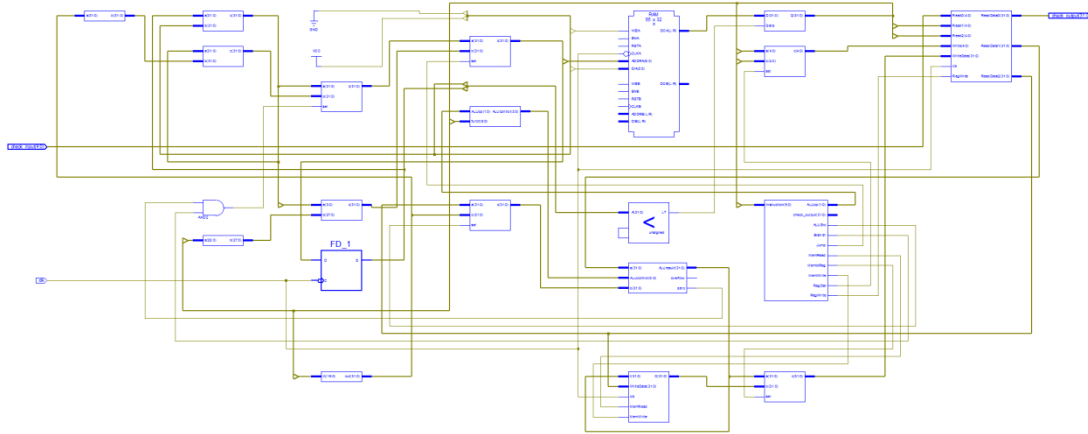


Fig. 7 RTL Schematic for Single-cycle CPU

### 3.2 Pipelined CPU

Except for some tiny difference in the components in single-cycle CPU, there are many new components in pipelined CPU design.

#### 3.2.1 Stage Registers

The stage registers are used to store the data passed between different stages, and they are controlled by the rising edge of the clock. At rising edge, the data can be written into the registers. Moreover, the signal “flush” can control whether to clear the content in the registers. If some undesired data are written, then the flush is set to 1 and the stage registers will be cleared.

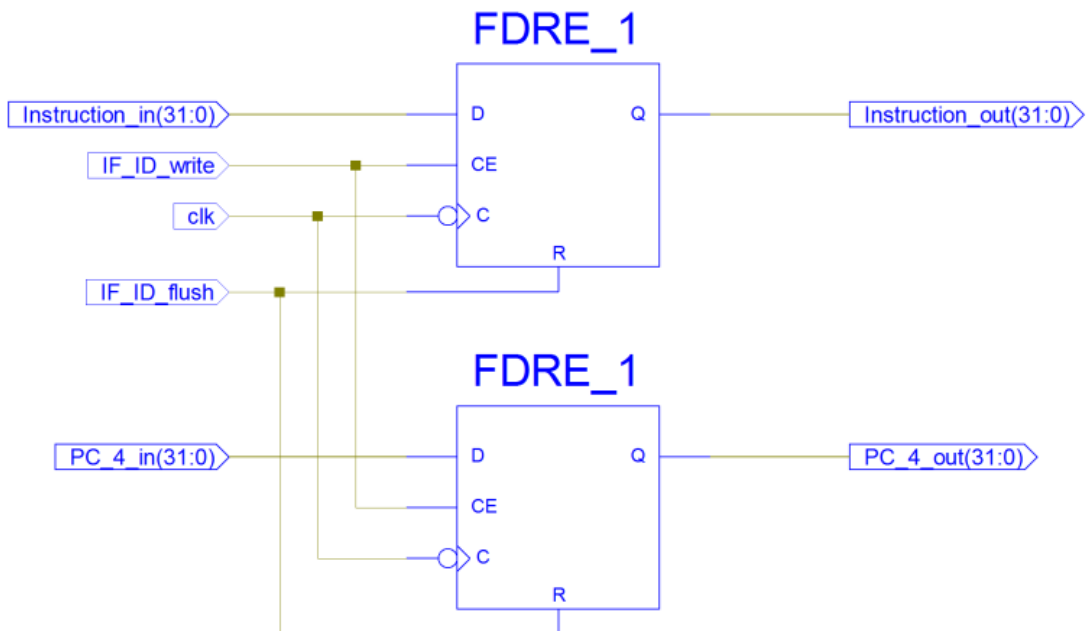


Fig. 8 RTL Schematic for IF-ID Register

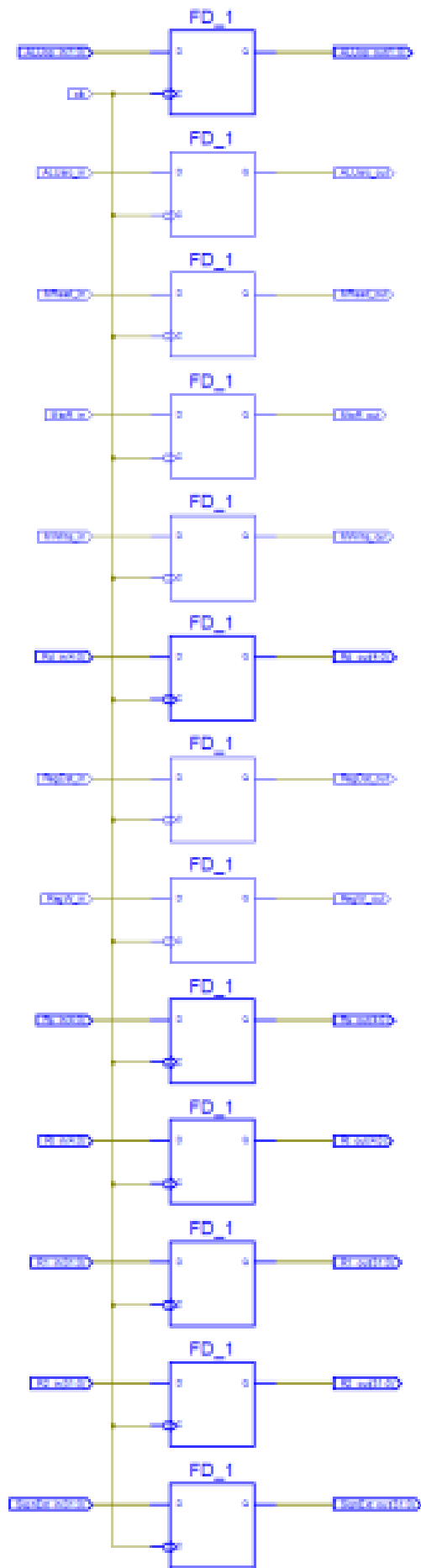


Fig. 9 RTL Schematic for ID-EX Register

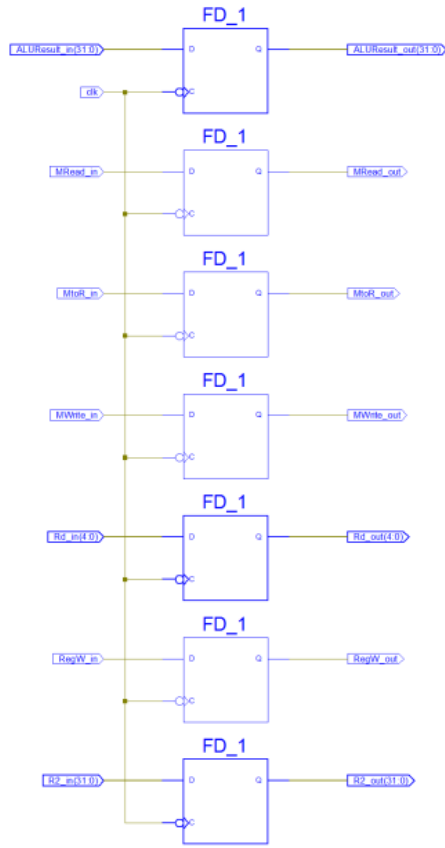


Fig. 10 RTL Schematic for EX-MEM Register

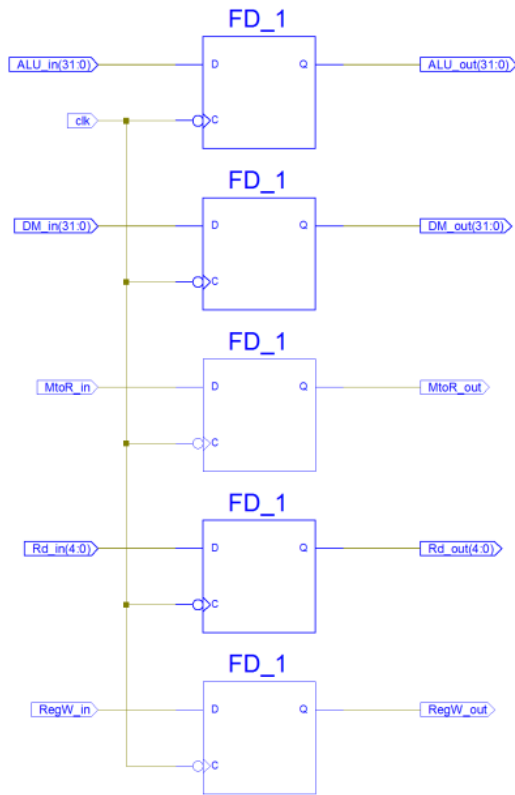


Fig. 11 RTL Schematic for MEM-WB Register

### 3.2.2 Forwarding Unit

Sometimes data needs to be used before it is written into the register file in a pipelined CPU, so we need to forward the result coming from ALU to the next instruction in advance. The forwarding unit is able to detect the forwarding condition and execute the forwarding.

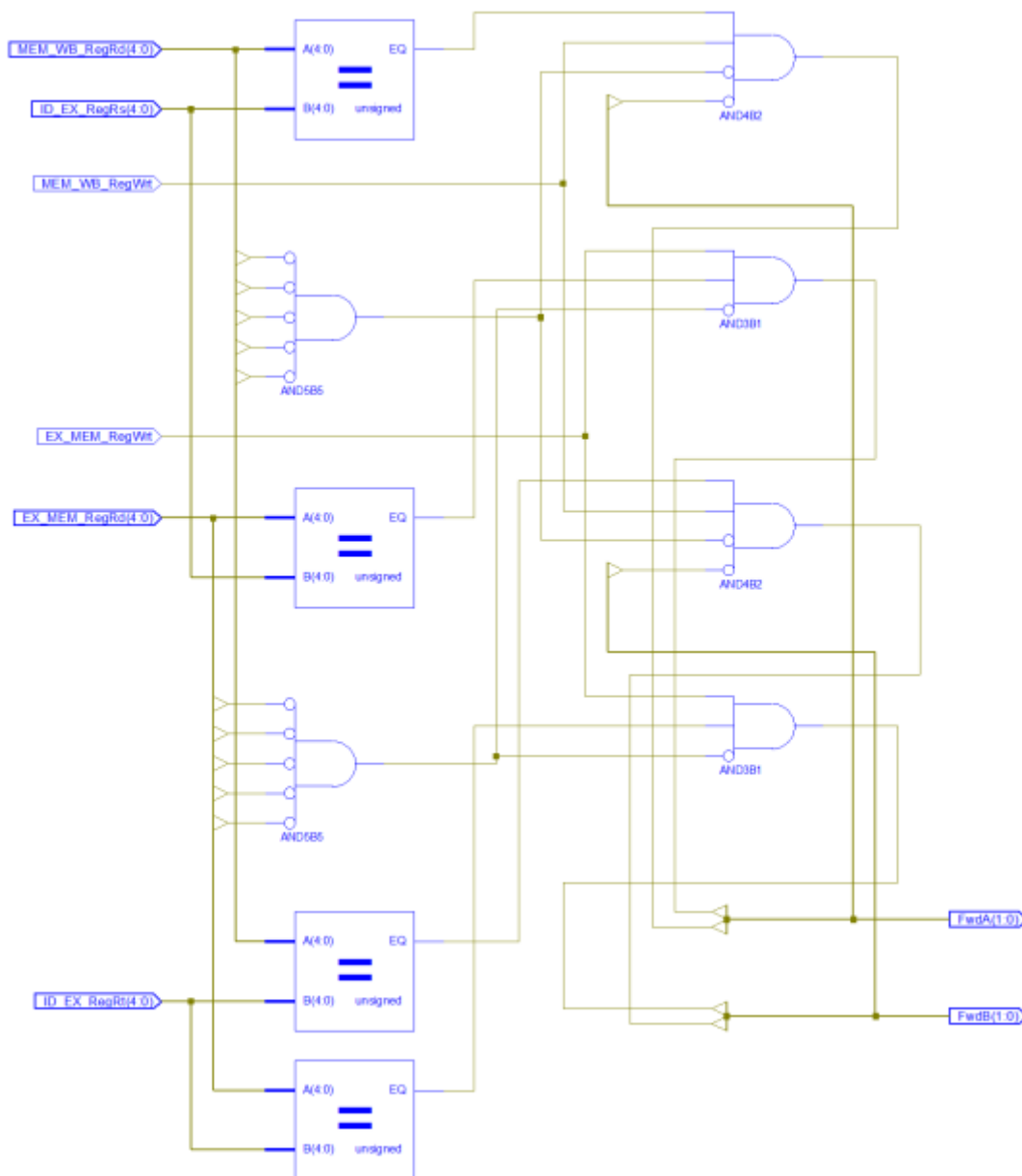


Fig. 12 RTL Schematic for Forwarding Unit

### 3.2.3 Hazard Detection Unit

This hazard detection unit can detect two kinds of hazard. One is load-use data hazard. Once it detects the load-use data hazard, it will stall the PC and IF/ID stage register by one clock cycle. Then the load-word instruction may be able to use the result forwarding from the ALU output. The other is control hazard caused by branch instructions like “beq”. We make the comparison part at the ID part. If the data we get from the register-file has not been updated yet, we need to let the instruction wait or add some new forwarding unit for it. We use the first way, inserting a nop, to solve the problem.

The hazard detection actually are the module to insert nop and decided next PC.

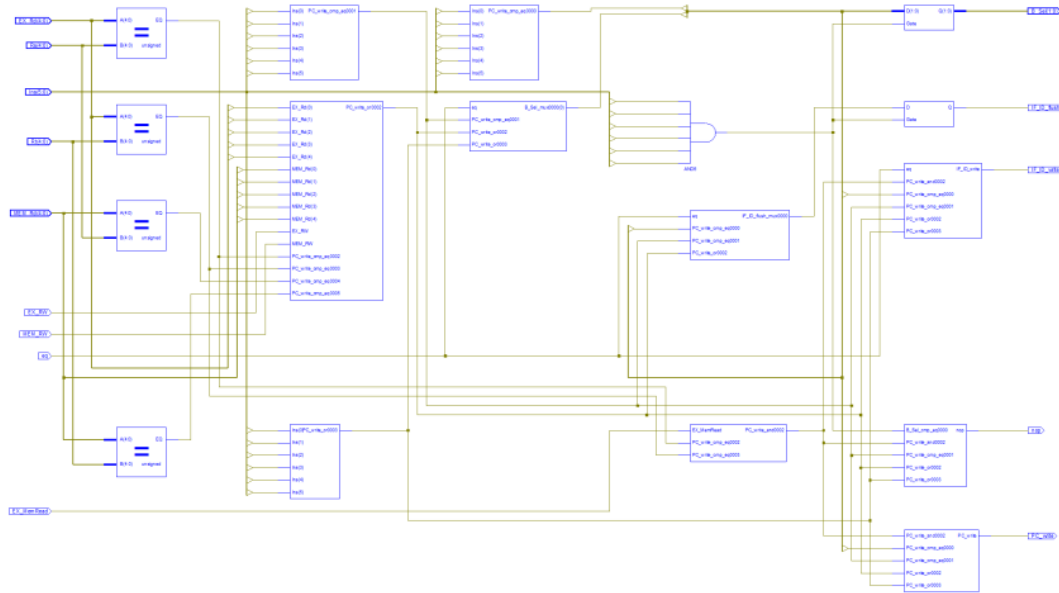


Fig. 13 RTL Schematic for Hazard Detection Unit

### 3.2.4 Structure of Pipelined CPU

The complete RTL schematic of a pipelined CPU is shown below:

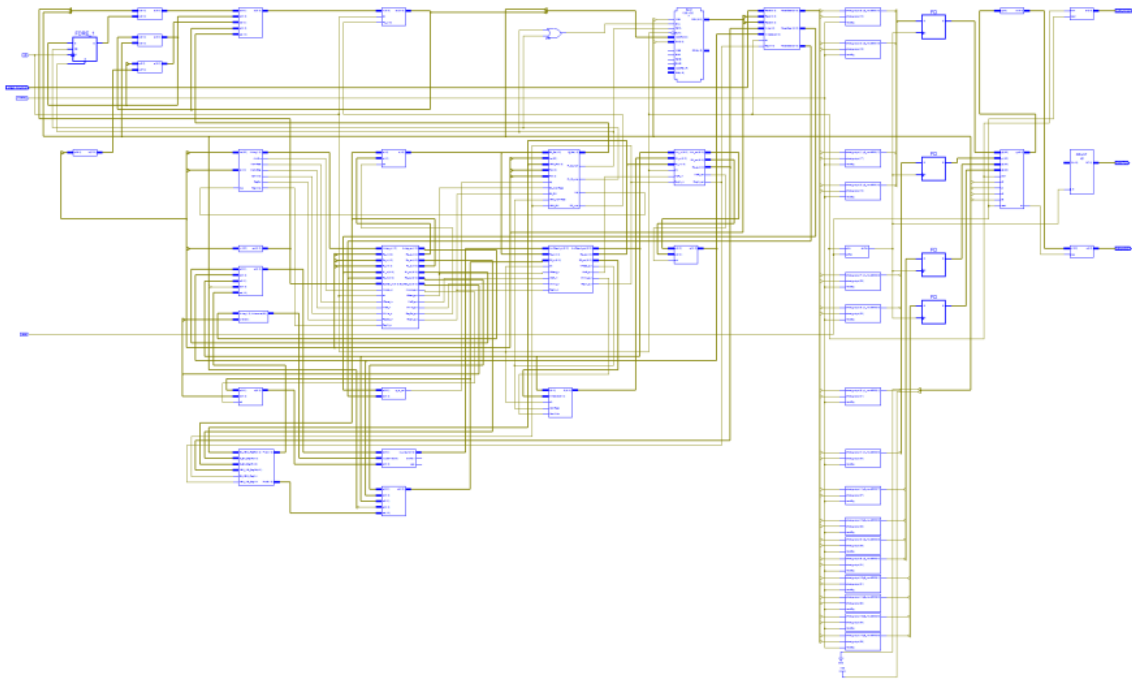


Fig. 14 RTL Schematic for Pipelined CPU

## 3.3 FPGA Board Implementation

In order to check whether we successfully implement the module with the help of an FPGA board, we use the four SSDs, six switches and eight LED lights on the FPGA board to show any of the values stored in the register file at any time. We set up a top module, and set CPU module and the display module as two sub-modules.

The four SSDs can be used to display either the most or the least significant two bytes of one particular variable in the register file. Five of the six switches will determine the position of this particular variable and the remaining one switch will determine which two bytes will be displayed. The eight LED lights will indicate how many clock cycles have already passed.

## 4. Testing and Results

### 4.1 Testing Instructions

```
    addi $s1, $0, 10;
    addi $s0, $0, 10;
    addi $s0, $s0, 10;
    add $s2, $s1, s0;
    addi $t0, $0, 1;
    add $t0, $s2, $t0;
    addi $t1, $0, 1;
    slt $t2, $t1, $t0;
    beq $t1, $t2, TAR;
    addi $s0, $0, 0;
    addi $s1, $0, 0;
    addi $s7, $0, 7;
    addi $s6, $0, 6;
TAR:  sw $s2, 4($0);
    lw $t3, 4($0);
    addi $t4, $t3, 5;
BACK: addi $t4, $t4, 5;
    sub $t5, $t3, t4;
    and $t6, $t3, $t5;
    or $t7, $t5, $t0;
    j TAR2;
    addi $s0, $0, 127;
TAR2: addi $s0, $0, 127;
    addi $s6, $0, 255;
    sw $s6, 4($0);
    lw $s7, 4($0);
    beq $s6, $s7, BACK;
```

A good set of testing instructions as shown above must cover all the instructions required. What is more, all kinds of hazard, including EX data hazard, MEM data hazard, load-use hazard and control hazard, must be perfectly dealt.

### 4.2 Single-cycle CPU Software Simulation

The clock cycle is 40ns for this software simulation. The result of the simulation is shown below. It can be found that this result follows the real result of the testing instructions.

Time: 0 [clk] = 1

































































































































```

[$s0] = 00000014 [$s1] = 0000000a [$s2] = 0000001e
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 000000ff [$s7] = 000000ff [$t0] = 0000001f
[$t1] = 00000001 [$t2] = 00000001 [$t3] = 0000001e
[$t4] = 0000008c [$t5] = ffffffff92 [$t6] = 00000012
[$t7] = ffffffff9f [$t8] = 00000000 [$t9] = 00000000
Time:      199 [clk] = 0
[$s0] = 00000014 [$s1] = 0000000a [$s2] = 0000001e
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 000000ff [$s7] = 000000ff [$t0] = 0000001f
[$t1] = 00000001 [$t2] = 00000001 [$t3] = 0000001e
[$t4] = 0000008c [$t5] = ffffffff92 [$t6] = 00000012
[$t7] = ffffffff9f [$t8] = 00000000 [$t9] = 00000000
Time:      200 [clk] = 1
[$s0] = 00000014 [$s1] = 0000000a [$s2] = 0000001e
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 000000ff [$s7] = 000000ff [$t0] = 0000001f
[$t1] = 00000001 [$t2] = 00000001 [$t3] = 0000001e
[$t4] = 0000008c [$t5] = ffffffff92 [$t6] = 00000012
[$t7] = ffffffff9f [$t8] = 00000000 [$t9] = 00000000

```

### 4.3 Pipelined CPU Software Simulation

The clock cycle is 40ns for this software simulation. The result of the simulation is shown below. It can be found that this result follows the real result of the testing instructions.

```

Time:      0 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      0 [clk] = 0
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      1 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      1 [clk] = 0

```



































































































































## 4.4 Hardware Implementation

The clock cycle is one second for the hardware implementation. The result of the implementation is shown below. It can be found that this result follows the real result of the testing instructions.

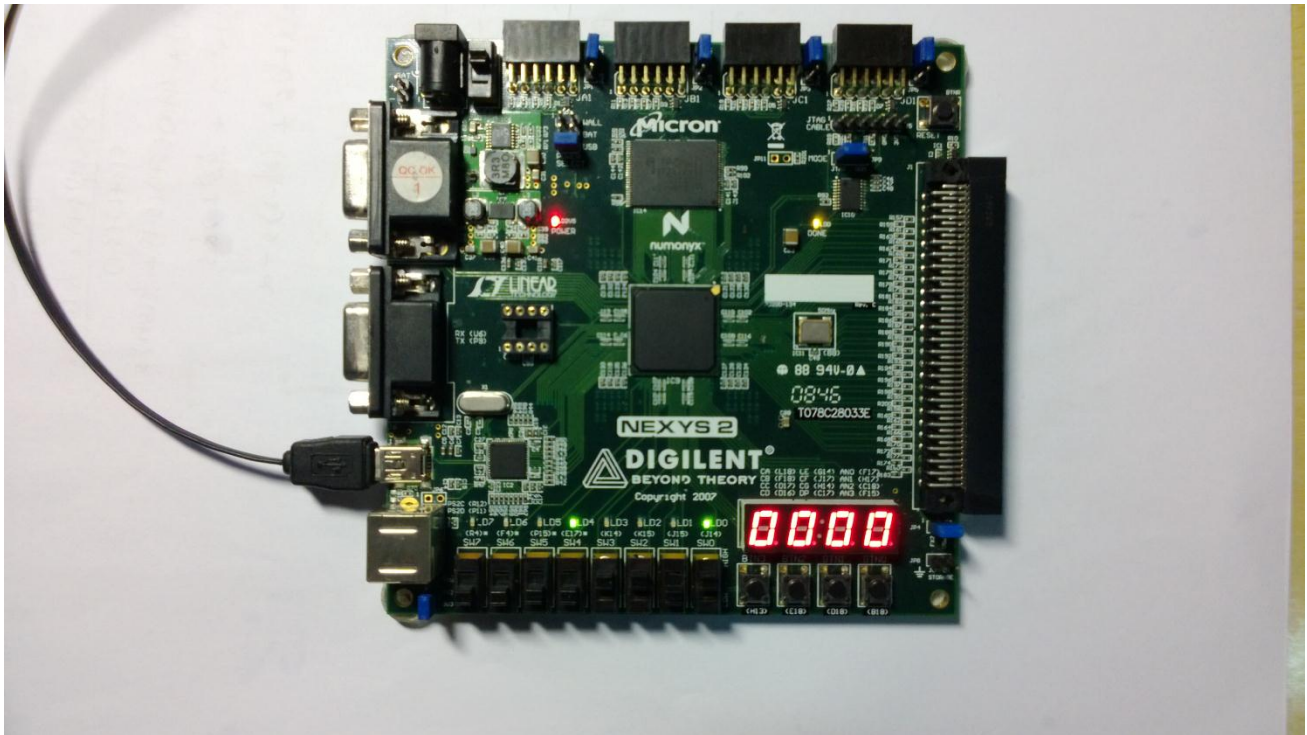


Fig. 15 Hardware Implementation

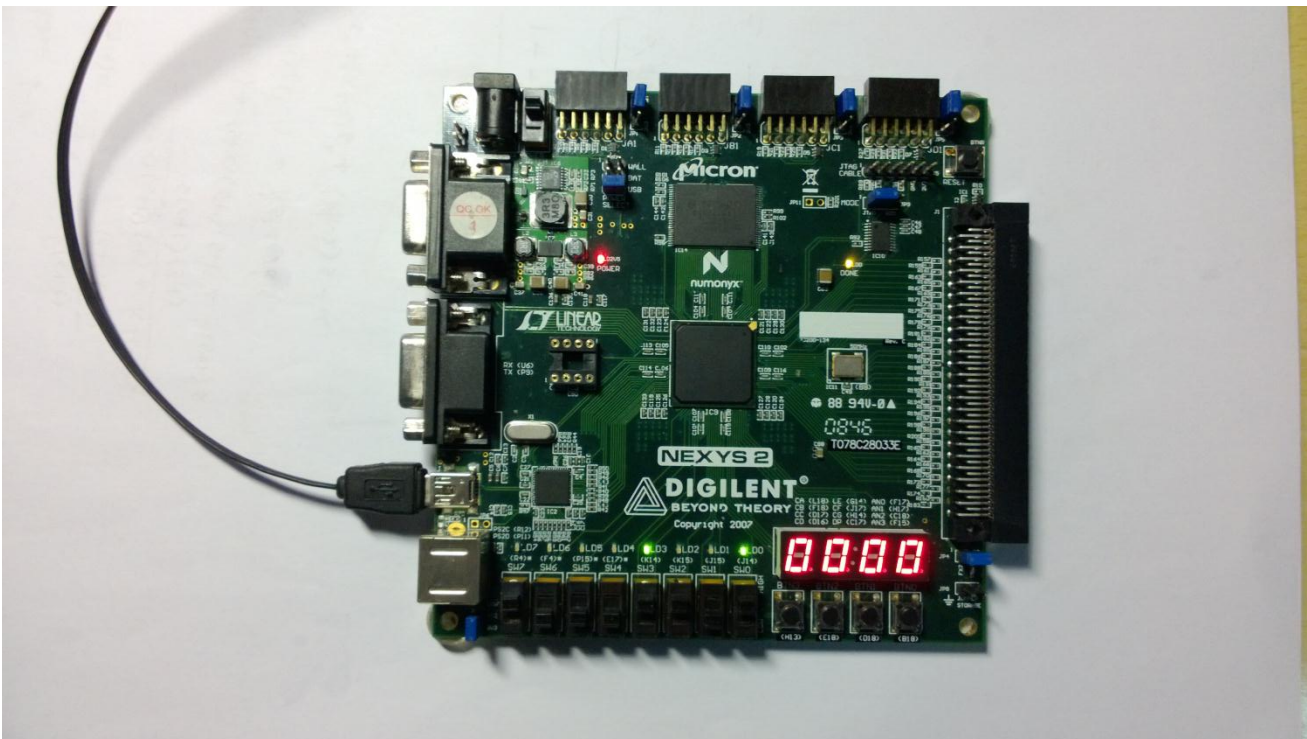


Fig. 16 Hardware Implementation

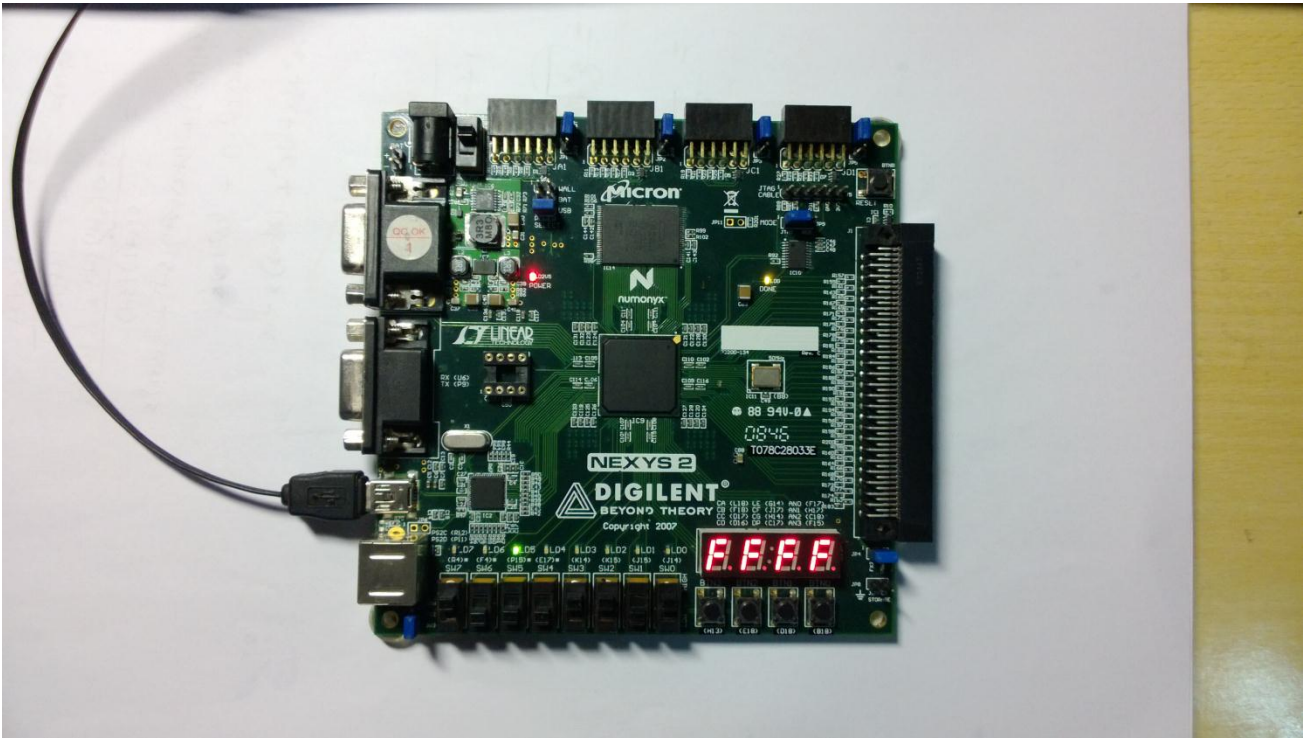


Fig. 17 Hardware Implementation

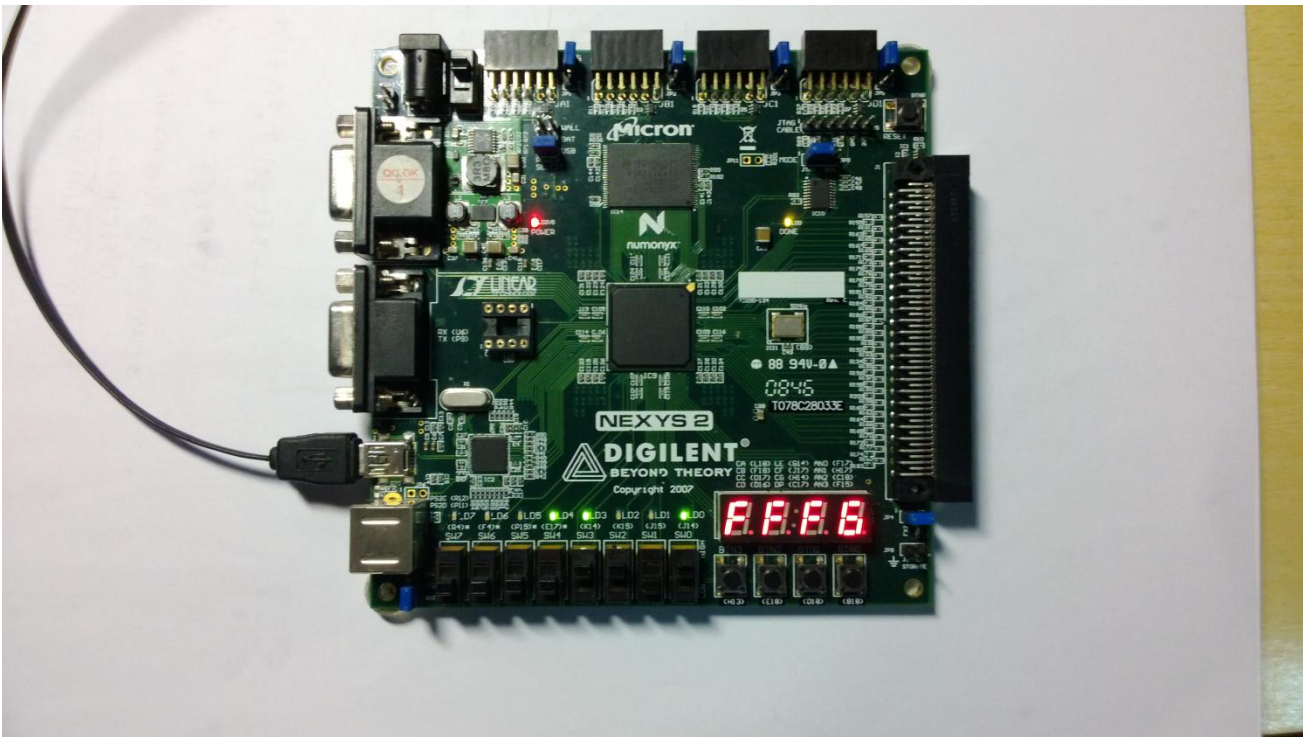


Fig. 18 Hardware Implementation

Fig. 13 and Fig. 14 mean that during the ninth second, the value stored in \$t5 is 0. Fig. 17 and Fig. 18 mean that this value will eventually be 0xFFFFFFFF6, which is -10.

## 5. Discussions

In this section we will mainly discuss about some strange problems we met during the project and the solutions to them.

The first problem was caused by an unexpected infinite loop. When we tested the single-cycle CPU, a

strange phenomenon appeared: the clock doesn't run and the simulation time remains 0 ns. We had initialized the clock as well as other inputs in the testbench, but it still didn't work. After checking most of the components, we suddenly found that when closing the Isim, it asked us whether to "save the change", which is abnormal if the simulation is finished even the clock doesn't run. So, we derive that the simulation process is actually still running. By the information shown in Console, we thought we are stuck at some infinite loop while the simulation is initializing the circuit. Since this circuit initialization actually contains not only the initial sentences in the module but also many other constructions, the scanning range was quite large. Finally, with the help of our TA Miss Zheng, we found that there is a piece of code with start "always" but no conditions, thus an infinite loop is caused. And it was possibly because we were about to change the conditions there but forgot to write after we deleted the previous ones.

The second problem was caused by the different "comprehension" to the program between software and hardware. After we successfully simulated the pipelined CPU with Isim, we burnt the program to the board, but the instructions output from the instruction memory was a total mess. Since the program counter was checked fine, we found that there is one sentence "O = Imemory[I[7:2]]" may cause the confusion to the hardware. So we tried to modified the expression as well as the structure of the initialized instruction memory. After several trials, we corrected it as "....." and the program successfully ran on the FPGA board.

The third problem was caused by some unknown data change, which we called "data shaking". When we simulate the load-word instruction, a strange phenomenon appeared: the instruction next to the load-word instruction was not executed, but the PC indeed counted it. What's more, if the instruction next to this one belongs to the same type, then both of them were not executed. But if the instruction next to this one is of another type, then it can run normally. So we thought the problem occurred at the control unit. We monitored all the variables and found a very strange thing: the "nop" signal was 0 all the time, but the program went into the opposite branch: the branch taken if nop = =1. After many trials and analysis, we raised one hypothesis: it is possible that the nop changed suddenly to 1 and then immediately 0 at the moment when the opcode of the load-word instruction came to the control unit, but after it changed back, the control signals didn't change back because we didn't include the "nop" signal into the "always @" conditions. Then we added "nop" as well as other possible affecting signals, and the program ran normally! Though we don't know why the signal shook, it definitely happened, and we checked all the similar potential problems. This is the last problem we fixed. After that all the tests can be done successfully.

## **6. Conclusion**

After days of analysis, writing and testing, we successfully finish all the tasks: simulation of single-cycle CPU and pipelined CPU, and implementation of pipelined CPU. We are satisfied about our results. We are very happy.



# Appendices

## Appendix A: Source Code of Single-cycle CPU

```
module single_cycle(clk,check_input,check_output);
    input  [4:0]  check_input;
    input                clk;
    output [31:0]  check_output;
    wire   [31:0]  check_output1;
    wire   [31:0]  w1,w2,w3,w4,w5,w6,w7,w8,w9,
                w10,w11,w13,w14,w15,w17,sw1;

    wire   [27:0]  sw;
    wire   [4:0]  w12;
    wire                I0,I1,I2,I3,I4,I6,I7,I8,I9,I10;
    wire   [1:0]  I5;
    wire   [3:0]  I11;
    wire                ww1,ww2,ww3;
    reg    [31:0]  four;

    initial four = 4;
    PC pc(clk,w1,w2);
    Instruction_memory IM (w2,w4);
    RegisterFile RF (clk,w4[25:21],w4[20:16],check_input,
        w12,w15,I8,w8,w9,check_output);
    DataMemory DM (clk,w13,w9,w14,I6,I3);
    control CL (check_output1,w4[31:26],I0,I1,I2,I3,I4,I5,I6,I7,I8);
    ALU_control AC (w4[5:0],I5,I11);
    ALU A1 (w8,w11,I9,w13,I11,ww1);
    mux_n_bit_2_to_1 #(5) m1 (w4[20:16],w4[15:11],w12,I0);
    mux_n_bit_2_to_1 #(32) m2 (w9,w10,w11,I7);
    mux_n_bit_2_to_1 #(32) m3 (w13,w14,w15,I4);
    mux_n_bit_2_to_1 #(32) m4 (w3,w5,w6,I10);
    mux_n_bit_2_to_1 #(32) m5 (w6,w7,w1,I1);
    shift_left_2_n_to_nplus2_bit sl1 (w4[25:0],sw);
    combine c1 (w3[31:28],sw,w7);
    Add ad1 (w2,four,w3);
    Add ad2 (w3,w17,w5);
    shift_left_2_n_bit sl (w10,w17);
    Sign_extend se (w4[15:0],w10);
    assign I10 = I2 & I9;
endmodule
```

```
module mux_n_bit_2_to_1(a,b,c,sel);
    parameter          N=1;
    input  [N-1:0]    a,b;
    input                sel;
```

```

output [N-1:0] c;
reg [N-1:0] c;
always @(a,b,sel)
begin
    if(sel==0)
        c=a;
    else if(sel==1)
        c=b;
end
endmodule

module combine(a,b,c);
input [3:0] a;
input [27:0] b;
output [31:0] c;
assign c[31:28]=a;
assign c[27:0]=b;
endmodule

module shift_left_2_n_to_nplus2_bit(a,b);
parameter N=26;
input [N-1:0] a;
output [N+1:0] b;
reg [N+1:0] b;
always @(a)
begin
    b[N+1:2] <=a [N-1:0];
    b[1:0] <= 0;
end
endmodule

module shift_right_2_n_bit(a,b);
parameter N=32;
input [N-1:0] a;
output [N-1:0] b;
reg [N-1:0] b;
always @(a)
begin
    b[N-3:0] <= a[N-1:2];
    b[N-1:N-2] <= 0;
end
endmodule

module shift_left_2_n_bit(a,b);
parameter N=32;
input [N-1:0] a;
output [N-1:0] b;
reg [N-1:0] b;
always @(a)

```

```

begin
    b[N-1:2] <= a[N-3:0];
    b[1:0] <= 0;
end
endmodule

```

```

module ALU(a,b,zero,ALUresult,ALUcontrol,overflow);
input  [31:0] a,b;
input  [3:0]  ALUcontrol;
output          zero,overflow;
output [31:0]  ALUresult;
reg          overflow,zero;
reg  [31:0]  ALUresult;
always @(ALUcontrol,a,b)
begin
    zero = (a == b);
    overflow = 0;
    case(ALUcontrol)
        4'b0000:
            ALUresult = a & b;
        4'b0001:
            ALUresult=a|b;
        4'b0010:
            begin
                ALUresult = a + b;
                overflow=(~a[31]&&~b[31]&&ALUresult[31])
                    || (a[31]&&b[31]&&~ALUresult[31]);
            end
        4'b0110:
            begin
                ALUresult = a - b;
                overflow=(a[31]&&~b[31]&&~ALUresult[31])
                    || (~a[31]&&b[31]&&ALUresult[31]);
            end
        4'b0111:
            ALUresult = (a < b);
        default
            ALUresult = 0;
    endcase
end
endmodule

```

```

module ALU_control(funcnt,ALUop,ALUcontrol);
input  [5:0]  funcnt;
input  [1:0]  ALUop;
output [3:0]  ALUcontrol;
reg  [3:0]  ALUcontrol;
always @(funcnt, ALUop)
    if(ALUop==2)

```

```

begin
    case (funct)
        6'b100000: ALUcontrol=4'b0010;
        6'b100010: ALUcontrol=4'b0110;
        6'b100100: ALUcontrol=4'b0000;
        6'b100101: ALUcontrol=4'b0001;
        6'b101010: ALUcontrol=4'b0111;
    endcase
end
else if (ALUop==0)
    ALUcontrol=4'b0010;
else if (ALUop==1)
    ALUcontrol=4'b0110;
else
    ALUcontrol=0;
endmodule

module control (check_output, Instruction, RegDst, Jump, Branch, MemRead,
MemtoReg, ALUop, MemWrite, ALUSrc, RegWrite);
input [5:0] Instruction;
output RegDst, Jump, Branch, MemRead,
MemtoReg, MemWrite, ALUSrc, RegWrite;
output [1:0] ALUop;
output [31:0] check_output;
reg [31:0] check_output;
reg RegDst, Jump, Branch, MemRead, MemtoReg,
MemWrite, ALUSrc, RegWrite;
reg [1:0] ALUop;
reg flag;

initial
begin
    flag <= 0;
    check_output <= 0;
end

always @(Instruction)
if (~flag)
case (Instruction)
6'b100011:
begin
    RegDst=0; Jump=0; Branch=0; MemRead=1; MemtoReg=1;
    ALUop=0; MemWrite=0; ALUSrc=1; RegWrite=1;
end//lw
6'b101011:
begin
    RegDst=0; Jump=0; Branch=0; MemRead=0; MemtoReg=0;
    ALUop=0; MemWrite=1; ALUSrc=1; RegWrite=0;
end//sw

```



```

        6'b000000:
            begin
                RegDst=1;Jump=0;Branch=0;MemRead=0;MemtoReg=0;
                ALUOp=2;MemWrite=0;ALUSrc=0;RegWrite=1;
            end//add,sub,or,alt
        6'b000100:
            begin
                RegDst=0;Jump=0;Branch=1;MemRead=0;MemtoReg=0;
                ALUOp=1;MemWrite=0;ALUSrc=0;RegWrite=0;
            end//beq
        6'b000010:
            begin
                RegDst=0;Jump=1;Branch=0;MemRead=0;MemtoReg=0;
                ALUOp=1;MemWrite=0;ALUSrc=0;RegWrite=0;
            end//j
        6'b001000:
            begin
                RegDst=0;Jump=0;Branch=0;MemRead=0;MemtoReg=0;
                ALUOp=0;MemWrite=0;ALUSrc=1;RegWrite=1;
            end//addi
        6'b111111:
            begin
                MemWrite=0; RegWrite=0; flag=1;check_output=1;
            end
    endcase
endmodule

```

```

module Sign_extend(in, out);
    input  [15:0] in;
    output [31:0] out;
    reg    [31:0] out;

    always @(in)
        begin
            out[15:0] <= in;
            if(in[15])
                begin
                    out[31:16] <= 16'hFFFF;
                end
            else if (~in[15])
                begin
                    out[31:16] <= 16'h0000;
                end
            end
        end
endmodule

```

```

module Add(a,b,c);
    input  [31:0] a,b;
    output [31:0] c;

```

```

    assign c = a + b;
endmodule

```

```

module PC(clk,in,out);
    input      clk;
    input  [31:0] in;
    output [31:0] out;
    reg   [31:0] out;

    initial out=0;
    always @(negedge clk) out=in;
endmodule

```

```

module DataMemory(clk, I,WriteData,O,MemWrite,MemRead);
    parameter      N = 64;
    input  [31:0] I,WriteData;
    input      MemWrite,MemRead, clk;
    output [31:0] O;
    reg   [31:0] O;
    reg   [31:0] im [N-1:0];
    wire  [31:0] INew;
    integer      i;

    initial
        for(i=0;i<N;i=i+1)
            im[i]=32'b0;

    always @ (negedge clk)
        if (MemWrite)
            if(INew < N)
                im[INew]=WriteData;

    always @ (MemRead, INew)
        if(MemRead==1'b1)
            if(INew < N)
                O=im[INew];

    shift_right_2_n_bit sr (I,INew);
endmodule

```

```

module RegisterFile(clk,Read1,Read2,Read0,Write,WriteData,RegWrite,
    ReadData1,ReadData2,ReadData0);
    input  [4:0]  Read1, Read2, Read0, Write;
    input      RegWrite,clk;
    input  [31:0] WriteData;
    output [31:0] ReadData1,ReadData2,ReadData0;
    reg   [31:0] ReadData1,ReadData2,ReadData0;
    reg   [31:0] r[31:0];
    integer      k;

```

```

initial
    for(k=0;k<32;k=k+1)
        r[k]=32'b0;

always @ (clk)
    $monitor("Clock Time: ",$time,"ns    [clk] = %h\n", clk,
        "$s0 = %h [$s1] = %h [$s2] = %h\n",r[16],r[17],r[18],
        "$s3 = %h [$s4] = %h [$s5] = %h\n",r[19],r[20],r[21],
        "$s6 = %h [$s7] = %h [$t0] = %h\n",r[22],r[23],r[8],
        "$t1 = %h [$t2] = %h [$t3] = %h\n",r[9],r[10],r[11],
        "$t4 = %h [$t5] = %h [$t6] = %h\n",r[12],r[13],r[14],
        "$t7 = %h [$t8] = %h [$t9] = %h",r[15],r[24],r[25]);

always @ (Read1, Read2, Read0, r[Read1], r[Read2], r[Read0])
    begin
        ReadData1=r[Read1];
        ReadData2=r[Read2];
        ReadData0=r[Read0];
    end
always@(negedge clk)
    if(RegWrite)
        r[Write]=WriteData;

endmodule

```

```

module Instruction_memory(I,O);
    input  [31:0] I;
    output [31:0] O;
    reg    [31:0] Imemory[64:0];
    reg    [31:0] O;
    wire   [31:0] INew;
    integer      k;

    initial
        begin
            for(k=0;k<65;k=k+1)
                Imemory[k] = 32'hFFFFFFFF;
            //You can give your own code block here.
            Imemory[0] = 32'b00100000000100010000000000001010;
            Imemory[1] = 32'b00100000000100000000000000001010;
            Imemory[2] = 32'b0010001000010000000000000000001010;
            Imemory[3] = 32'b00000010001100001001000000100000;
            Imemory[4] = 32'b00100000000010000000000000000001;
            Imemory[5] = 32'b00000010010010000100000000100000;
            Imemory[6] = 32'b00100000000010010000000000000001;
            Imemory[7] = 32'b00000001001010000101000000101010;
            // Imemory[7] = 32'b00000001000010010101000000101010;
            Imemory[8] = 32'b00010001001010100000000000000100;
        end
    endmodule

```

```
Imemory[9] = 32'b00100000001000000000000000000000;
Imemory[10] = 32'b00100000001000100000000000000000;
Imemory[11] = 32'b00100000001011100000000000000111;
Imemory[12] = 32'b00100000001011000000000000000110;
Imemory[13] = 32'b10101100000100100000000000000100;
Imemory[14] = 32'b10001100000010110000000000000100;
Imemory[15] = 32'b00100001011011000000000000000101;
Imemory[16] = 32'b00100001100011000000000000000101;
Imemory[17] = 32'b00000001011011000110100000100010;
Imemory[18] = 32'b00000001011011010111000000100100;
Imemory[19] = 32'b00000001101010000111100000100101;
Imemory[20] = 32'b000010000000000000000000000010110;
Imemory[21] = 32'b001000000001000000000000001111111;
Imemory[22] = 32'b00100000000101100000000011111111;
Imemory[23] = 32'b10101100000101100000000000000100;
Imemory[24] = 32'b10001100000101110000000000000100;
Imemory[25] = 32'b00010010110101111111111111110110;
```

**end**

```
shift_right_2_n_bit sr (I,INew);
```

```
always @(INew)
```

```
  if(INew<65)
```

```
    O=Imemory[INew];
```

```
endmodule
```

## Appendix B: Single-cycle CPU Test Bench

```
module SC_test;
    reg clk;
    reg [4:0] check_input;
    wire [31:0] check_output;
    single_cycle uut (
        .clk(clk),
        .check_input(check_input),
        .check_output(check_output)
    );
    initial clk = 1;
    always #20 clk = ~clk;
endmodule
```

## Appendix C: Source Code of Pipelined CPU and Hardware Implement Unit

```
`timescale 1ns / 1ps
module Pipeline(clk,check_input,mostSig,reset,cNum,anNum,light);
    input      [4:0]  check_input;
    input      clk,reset,mostSig;
    output     [6:0]  cNum;
    output     [3:0]  anNum;
    output     [7:0]  light;
    wire       [31:0] check_output,w14,w15,w16,w20,w21,w22,w23,
                w24,w26,w27,w29,w31;
    wire       [31:0] w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,
                ww1,ww2,ww3;
    wire       [27:0] sw1;
    wire       [4:0]  w17,w18,w19,w25,w28,w30;
    wire       [3:0]  I28;
    wire       [1:0]  I1,I6,I13,I25,I26;
    wire       [15:0] dNum;
    wire       [3:0]  qx;
    wire       dx,clkOut;
    wire       I0,I2,I3,I4,I5,I7,I8,I9,I10,I11,I12,I14,I15,I16,I17;
    wire       zero,overflow,I18,I19,I20,I21,I22,I23,I24,I27;
    reg        [31:0] four;
    reg        [3:0]  q0, q1, q2, q3;
    reg        d0, d1, d2, d3;
    reg        [7:0]  light;

    // Initialization
    initial
        begin
            four <= 4;
            light <= 0;
        end

    // Clock Divider
    divider DVD (clkOut, clk, reset);

    // PipeliLine CirCuit (clkOut changed to clk for software simulation)
    PL_PC pcl(clkOut,I0,w8,w3);
    Add add1(w3,four,w4);
    Instruction_memory InsM(w3,w5);
    IF_ID if_id(clkOut,w5,w4,I2,I3,w1,w11);
    shift_left_2_n_to_nplus2_bit sl1(w1[25:0],sw1);
    combine cmb(w4[31:28],sw1,w7);
    Sign_extend se(w1[15:0],w2);
    assign w31 = w2 << 2;
    Add add2(w31,w11,w6);
    mux_4_to_1 #(32)M1 (w6,w4,w7,w3,I1,w8);
```

```

hazard_detection_unit_and_branch hz(wl[31:26],wl[25:21],wl[20:16],
    I4,w19,w25,I15,I20,I18,I22,I0,I1,I2,I3,I27);
PL_control con(wl[15:11],wl[31:26],I27,I11,I10,I9,I8,I7,I6,I5);
equal eq(w9,w10,I4);
PL_RegisterFile RF (clkOut,wl[25:21],wl[20:16],check_input,w28,w29,
    I24,w9,w10,check_output);
ID_EX id_ex(clkOut,I11,I17,I10,I18,I9,I15,I8,I16,I7,I14,I6,I13,I5,
    I12,w9,w12,w10,w13,w2,w16,wl[20:16],w17,wl[15:11],
    w18,wl[25:21],w30);
mux_4_to_1 #(32)M2 (w12,w29,w22,ww1,I26,w14);
mux_4_to_1 #(32)M3 (w13,w29,w22,ww2,I25,w15);
mux_n_bit_2_to_1 #(5) M4 (w17,w18,w19,I14);
mux_n_bit_2_to_1 #(32) M6(w15,w16,w20,I12);
ALU_control ALU_C(w16[5:0],I13,I28);
ALU AA(w14,w20,zero,w21,I28,overflow);
forwarding_unit fu(w30, w17, w25, I22, w28, I24,I26,I25);
EX_MEM ex_mem (clkOut,I17,I21,I18,I22,I15,I20,I16,I19,w21,
    w22,w15,w23,w19,w25);
PL_DataMemory dm(clkOut,w22,w23,w24,I19,I20);
MEM_WB mem_wb (clkOut,I21,I23,I22,I24,w24,w26,w22,w27,w25,w28);
mux_n_bit_2_to_1 #(32) M5 (w27,w26,w29,I23);

```

```
// Display Unit
```

```

an_counter ANCN1 (anNum,clk, reset);
select SLCT1 (qx,dx, q0,q1,q2,q3, d0, d1, d2,d3,clk,reset);
decoder_4_16 DCD1 (dNum, qx);
SSD_4_sign SSD (cNum, dNum, dx);

```

```
always @ (posedge clkOut)
```

```
begin
```

```
light <= light + 1;
```

```
if (mostSig)
```

```
begin
```

```
q0 <= check_output[19:16]; q1 <= check_output[23:20];
```

```
q2 <= check_output[27:24]; q3 <= check_output[31:28];
```

```
d0 <= 1; d1 <= 1; d2 <= 1; d3 <= 1;
```

```
end
```

```
else if (~mostSig)
```

```
begin
```

```
q0 <= check_output[3:0]; q1 <= check_output[7:4];
```

```
q2 <= check_output[11:8]; q3 <= check_output[15:12];
```

```
d0 <= 1; d1 <= 1; d2 <= 1; d3 <= 1;
```

```
end
```

```
end
```

```
endmodule
```

```
module PL_PC(clk,PC_write,in,out);
```

```
input clk,PC_write;
```

```

input  [31:0] in;
output [31:0] out;
reg    [31:0] out;

initial out <= 0;
always@(negedge clk)
    if(PC_write)
        out=in;
endmodule

module Add(a,b,c);
input  [31:0] a,b;
output [31:0] c;

assign c = a + b;
endmodule

module PL_RegisterFile(clk,Read1,Read2,Read0,Write,WriteData,
    RegWrite,ReadData1,ReadData2,ReadData0);
input  [4:0]  Read1, Read2, Read0, Write;
input          RegWrite,clk;
input  [31:0] WriteData;
output [31:0] ReadData1,ReadData2,ReadData0;
reg    [31:0] ReadData1,ReadData2,ReadData0;
reg    [31:0] r [31:0];
integer k;

initial
    for(k = 0; k < 32; k = k+1)
        r[k]=32'b0;

always @ (clk)
begin
    $monitor("Clock Time: ",$time,"ns    [clk] = %h\n", clk,
        "[${s0}] = %h [${s1}] = %h [${s2}] = %h\n",r[16],r[17],r[18],
        "[${s3}] = %h [${s4}] = %h [${s5}] = %h\n",r[19],r[20],r[21],
        "[${s6}] = %h [${s7}] = %h [${t0}] = %h\n",r[22],r[23],r[8],
        "[${t1}] = %h [${t2}] = %h [${t3}] = %h\n",r[9],r[10],r[11],
        "[${t4}] = %h [${t5}] = %h [${t6}] = %h\n",r[12],r[13],r[14],
        "[${t7}] = %h [${t8}] = %h [${t9}] = %h",r[15],r[24],r[25]);
end

always@(Read1, Read2, Read0, r[Read1], r[Read2], r[Read0])
begin
    ReadData1=r[Read1];
    ReadData2=r[Read2];
    ReadData0=r[Read0];
end

```



```

always@(posedge clk)
    if(RegWrite==1'b1)
        r[Write]=WriteData;
endmodule

```

```

module Instruction_memory(I,O);
    input  [31:0] I;
    output [31:0] O;
    reg    [31:0] Imemory [127:0];
    reg    [31:0] O;
    wire   [31:0] INew;
    reg    [6:0]  II;
    integer      k;

    initial
        begin
            II=0;
            for(k = 0; k < 128; k = k + 1)
                begin
                    Imemory[k] = 32'hFFFFFFFF;
                end

            //You can give your own code block here.
            Imemory[0] = 32'b00100000000100010000000000001010;
            Imemory[1] = 32'b00100000000100000000000000001010;
            Imemory[2] = 32'b00100010000100000000000000001010;
            Imemory[3] = 32'b00000010001100001001000000100000;
            Imemory[4] = 32'b00100000000010000000000000000001;
            Imemory[5] = 32'b00000010010010000100000000100000;
            Imemory[6] = 32'b00100000000010010000000000000001;
            Imemory[7] = 32'b00000001001010000101000000101010;
            // Imemory[7] = 32'b0000000100001001010101000000101010;
            Imemory[8] = 32'b000100010010101010000000000000100;
            Imemory[9] = 32'b00100000000100000000000000000000;
            Imemory[10] = 32'b00100000000100010000000000000000;
            Imemory[11] = 32'b00100000000101110000000000000111;
            Imemory[12] = 32'b00100000000101100000000000000110;
            Imemory[13] = 32'b10101100000100100000000000000100;
            Imemory[14] = 32'b10001100000010110000000000000100;
            Imemory[15] = 32'b00100001011011000000000000000101;
            Imemory[16] = 32'b00100001100011000000000000000101;
            Imemory[17] = 32'b00000001011011000110100000100010;
            Imemory[18] = 32'b00000001011011010111000000100100;
            Imemory[19] = 32'b00000001101010000111100000100101;
            Imemory[20] = 32'b00001000000000000000000000010110;
            Imemory[21] = 32'b00100000000100000000000001111111;
            Imemory[22] = 32'b00100000000101100000000001111111;
            Imemory[23] = 32'b10101100000101100000000000000100;
            Imemory[24] = 32'b10001100000101110000000000000100;

```

```

        Imemory[25] = 32'b000100101101011111111111111110110;
    end

    always @(I)
    begin
        II = I[7:2];
        O = Imemory[II];
    end
endmodule

module shift_right_2_n_bit(a,b);
    parameter          N=32;
    input      [N-1:0] a;
    output     [N-1:0] b;
    reg        [N-1:0] b;

    always @(a)
    begin
        b[N-3:0]<=a[N-1:2];
        b[N-1:N-2]<=0;
    end
endmodule

module IF_ID(clk,Instruction_in,PC_4_in,IF_ID_write,IF_ID_flush,
    Instruction_out,PC_4_out);
    input  [31:0] Instruction_in,PC_4_in;
    input          clk,IF_ID_write,IF_ID_flush;
    output [31:0] Instruction_out, PC_4_out;
    reg    [31:0] Instruction_out, PC_4_out;

    initial
    begin
        Instruction_out=0;
        PC_4_out=0;
    end

    always @(negedge clk)
    begin
        if(IF_ID_write)
            begin
                Instruction_out <= Instruction_in;
                PC_4_out <= PC_4_in;
            end
        if(IF_ID_flush)
            begin
                Instruction_out <= 0;
                PC_4_out <= 0;
            end
    end
end

```

```
endmodule
```

```
module shift_left_2_n_to_nplus2_bit(a,b);
```

```
    parameter          N=26;
```

```
    input   [N-1:0] a;
```

```
    output  [N+1:0] b;
```

```
    reg     [N+1:0] b;
```

```
    always @(a)
```

```
        begin
```

```
            b[N+1:2] <= a[N-1:0];
```

```
            b[1:0] <= 0;
```

```
        end
```

```
endmodule
```

```
module combine(a,b,c);
```

```
    input   [3:0]  a;
```

```
    input   [27:0] b;
```

```
    output  [31:0] c;
```

```
    assign c[31:28] = a;
```

```
    assign c[27:0] = b;
```

```
endmodule
```

```
module Sign_extend(in, out);
```

```
    input   [15:0] in;
```

```
    output  [31:0] out;
```

```
    reg     [31:0] out;
```

```
    always @(in)
```

```
        begin
```

```
            out[15:0] <= in;
```

```
            if(in[15])
```

```
                out[31:16] <= 16'hFFFF;
```

```
            else if (~in[15])
```

```
                out[31:16] <= 16'h0000;
```

```
        end
```

```
endmodule
```

```
module mux_4_to_1(a,b,c,d,sel,o);
```

```
    parameter          N=32;
```

```
    input   [N-1:0] a,b,c,d;
```

```
    input   [1:0]  sel;
```

```
    output  [N-1:0] o;
```

```
    wire    [N-1:0] w0,w1;
```

```
    mux_n_bit_2_to_1 #(N) m1(a,b,w0,sel[0]);
```

```
    mux_n_bit_2_to_1 #(N) m2(c,d,w1,sel[0]);
```

```
    mux_n_bit_2_to_1 #(N) m3(w0,w1,o,sel[1]);
```

```
endmodule
```

```

module mux_n_bit_2_to_1(a,b,c,sel);
  parameter      N=1;
  input   [N-1:0] a,b;
  input      sel;
  output  [N-1:0] c;
  reg     [N-1:0] c;

  always @(a,b,sel)
    if(~sel)
      c=a;
    else if(sel)
      c=b;
endmodule

```

```

module equal(a,b,e_or_not);
  parameter      N=32;
  input   [N-1:0] a,b;
  output      e_or_not;

  assign e_or_not=(a==b);
endmodule

```

```

module ALU_control(funcnt,ALUop, ALUcontrol);
  input   [5:0]      funcnt;
  input   [1:0]      ALUop;
  output  [3:0]      ALUcontrol;
  reg     [3:0]      ALUcontrol;

  always @(funcnt, ALUop)
    if(ALUop==2)
      begin
        case(funcnt)
          6'b100000: ALUcontrol=4'b0010;
          6'b100010: ALUcontrol=4'b0110;
          6'b100100: ALUcontrol=4'b0000;
          6'b100101: ALUcontrol=4'b0001;
          6'b101010: ALUcontrol=4'b0111;
        endcase
      end
    else if(ALUop==0)
      ALUcontrol=4'b0010;
    else if(ALUop==1)
      ALUcontrol=4'b0110;
    else
      ALUcontrol=0;
endmodule

```

```

module ALU(a,b,zero,ALUresult,ALUcontrol,overflow);

```

```

input  [31:0] a,b;
input  [3:0]  ALUcontrol;
output          zero,overflow;
output [31:0]  ALUresult;
reg           overflow,zero;
reg  [31:0]  ALUresult;
always @(ALUcontrol,a,b)
  begin
    zero = (a==b);
    overflow = 0;
    case (ALUcontrol)
      4'b0000:
        ALUresult=a&b;
      4'b0001:
        ALUresult=a|b;
      4'b0010:
        begin
          ALUresult=a+b;
          overflow=(~a[31]&&~b[31]&&ALUresult[31]) ||
            (a[31]&&b[31]&&~ALUresult[31]);
        end
      4'b0110:
        begin
          ALUresult=a-b;
          overflow=(a[31]&&~b[31]&&~ALUresult[31]) ||
            (~a[31]&&b[31]&&ALUresult[31]);
        end
      4'b0111:
        begin
          ALUresult=(a<b);
        end
      default
        ALUresult=0;
    endcase
  end
endmodule

module PL_control(rd,Ins,nop,MemtoReg,RegWrite,MemRead,MemWrite,
  RegDst,ALUop,ALUSrc);
input  [5:0]  Ins;
input  [4:0]  rd;
input          nop;
output          MemtoReg,RegWrite,MemRead,MemWrite,RegDst,ALUSrc;
output [1:0]  ALUop;
reg           MemtoReg,RegWrite,MemRead,MemWrite,RegDst,ALUSrc;
reg  [1:0]  ALUop;
reg          flag;

initial

```



```

        end//j
        6'b001000:
        begin
            RegDst=0;MemRead=0;MemtoReg=0;ALUop=0;
            MemWrite=0;ALUSrc=1;RegWrite=1;
        end//addi
        6'b111111:
        begin
            MemWrite=0; RegWrite=0; flag=1;
        end
        default:
        begin
            RegDst=0;MemRead=0;MemtoReg=0;ALUop=0;
            MemWrite=0;ALUSrc=0;RegWrite=0;
        end
    endcase
end
else if(nop)
begin
    MemtoReg=0;
    RegWrite=0;
    MemRead=0;
    MemWrite=0;
    RegDst=0;
    ALUop=0;
    ALUSrc=0;
end
end
endmodule

```

```

module ID_EX(clk,MtoR_in,MtoR_out,RegW_in,RegW_out,MRead_in,MRead_out,
MWrite_in,MWrite_out,RegDst_in,RegDst_out,ALUop_in,ALUop_out,
ALUSrc_in,ALUSrc_out,R1_in,R1_out,R2_in,R2_out,SignExt_in,
SignExt_out,Rt_in,Rt_out,Rd_in,Rd_out,Rs_in,Rs_out);
input          clk,MtoR_in,RegW_in,MRead_in,MWrite_in,
              RegDst_in,ALUSrc_in;
input  [1:0]  ALUop_in;
input  [31:0] R1_in,R2_in,SignExt_in;
input  [4:0]  Rt_in, Rd_in,Rs_in;
output          MtoR_out,RegW_out,MRead_out,MWrite_out,
              RegDst_out,ALUSrc_out;
output  [1:0]  ALUop_out;
output  [31:0] R1_out,R2_out,SignExt_out;
output  [4:0]  Rt_out,Rd_out,Rs_out;
reg          MtoR_out,RegW_out,MRead_out,MWrite_out,
              RegDst_out,ALUSrc_out;
reg  [1:0]  ALUop_out;
reg  [31:0] R1_out,R2_out,SignExt_out;
reg  [4:0]  Rt_out,Rd_out,Rs_out;

```

```

initial
begin
    MtoR_out=0;
    RegW_out=0;
    MRead_out=0;
    MWrite_out=0;
    RegDst_out=0;
    ALUsrc_out=0;
    ALUop_out=0;
    R1_out=0;
    R2_out=0;
    SignExt_out=0;
    Rt_out=0;
    Rd_out=0;
    Rs_out=0;
end

always @(negedge clk)
begin
    MtoR_out=MtoR_in;
    RegW_out=RegW_in;
    MRead_out=MRead_in;
    MWrite_out=MWrite_in;
    RegDst_out=RegDst_in;
    ALUsrc_out=ALUsrc_in;
    ALUop_out=ALUop_in;
    R1_out=R1_in;
    R2_out=R2_in;
    SignExt_out=SignExt_in;
    Rt_out=Rt_in;
    Rd_out=Rd_in;
    Rs_out=Rs_in;
end
endmodule

module EX_MEM(clk,MtoR_in,MtoR_out,RegW_in,RegW_out,MRead_in,MRead_out,
MWrite_in,MWrite_out,ALUResult_in,ALUResult_out,
R2_in,R2_out,Rd_in,Rd_out);
input          clk,MtoR_in,RegW_in,MRead_in,MWrite_in;
input  [31:0]  ALUResult_in,R2_in;
input  [4:0]   Rd_in;
output        MtoR_out, RegW_out, MRead_out, MWrite_out;
output  [31:0] ALUResult_out,R2_out;
output  [4:0]  Rd_out;
reg       MtoR_out, RegW_out, MRead_out, MWrite_out;
reg  [31:0] ALUResult_out,R2_out;
reg  [4:0]  Rd_out;

```



```

initial
begin
    MtoR_out=0;
    RegW_out=0;
    MRead_out=0;
    MWrite_out=0;
    ALUResult_out=0;
    R2_out=0;
    Rd_out=0;
end

always @(negedge clk)
begin
    MtoR_out=MtoR_in;
    RegW_out=RegW_in;
    MRead_out=MRead_in;
    MWrite_out=MWrite_in;
    ALUResult_out=ALUResult_in;
    R2_out=R2_in;
    Rd_out=Rd_in;
end
endmodule

module MEM_WB(clk,MtoR_in,MtoR_out,RegW_in,RegW_out,DM_in,
    DM_out,ALU_in,ALU_out,Rd_in,Rd_out);
input          clk,MtoR_in,RegW_in;
input  [31:0]  DM_in,ALU_in;
input  [4:0]   Rd_in;
output        MtoR_out,RegW_out;
output  [31:0] DM_out,ALU_out;
output  [4:0]  Rd_out;
reg      MtoR_out,RegW_out;
reg  [31:0] DM_out,ALU_out;
reg  [4:0]  Rd_out;

initial
begin
    MtoR_out=0;
    RegW_out=0;
    DM_out=0;
    ALU_out=0;
    Rd_out=0;
end

always @(negedge clk)
begin
    MtoR_out=MtoR_in;
    RegW_out=RegW_in;
    DM_out=DM_in;

```

```

        ALU_out=ALU_in;
        Rd_out=Rd_in;
    end
endmodule

```

```

module PL_DataMemory(clk,I,WriteData,O,MemWrite,MemRead);
    parameter          N = 64;
    input              [31:0] I,WriteData;
    input              MemWrite,MemRead, clk;
    output            [31:0] O;
    reg                [31:0] O;
    reg                [31:0] im [N-1:0];
    wire              INew;
    integer           i;

    initial
        for(i=0;i<N;i=i+1)
            im[i]=32'b0;
    always @ (posedge clk)
        if (MemWrite)
            if(INew < N)
                im[INew]=WriteData;

    always @ (MemRead,INew)
        if(MemRead)
            if(INew < N)
                O=im[INew];

    shift_right_2_n_bit sr (I,INew);
endmodule

```

```

module forwarding_unit(ID_EX_RegRs,ID_EX_RegRt,EX_MEM_RegRd,
    EX_MEM_RegWrt,MEM_WB_RegRd,MEM_WB_RegWrt,FwdA,FwdB);
    input    [4:0]    ID_EX_RegRs,ID_EX_RegRt,EX_MEM_RegRd,MEM_WB_RegRd;
    input    EX_MEM_RegWrt,MEM_WB_RegWrt;
    output   [1:0]    FwdA, FwdB;
    reg      [1:0]    FwdA, FwdB;

    initial
        begin
            FwdA=0;
            FwdB=0;
        end

    always @ (ID_EX_RegRs,ID_EX_RegRt,EX_MEM_RegRd,
        EX_MEM_RegWrt,MEM_WB_RegRd,MEM_WB_RegWrt)
        begin
            if (EX_MEM_RegWrt & (EX_MEM_RegRd != 0) &
                (EX_MEM_RegRd == ID_EX_RegRs))

```

```

        FwdA = 2;
    else if (MEM_WB_RegWrt & (MEM_WB_RegRd != 0) &
        (MEM_WB_RegRd == ID_EX_RegRs))
        FwdA = 1;
    else
        FwdA = 0;

    if (EX_MEM_RegWrt & (EX_MEM_RegRd != 0) &
        (EX_MEM_RegRd == ID_EX_RegRt))
        FwdB = 2;
    else if (MEM_WB_RegWrt & (MEM_WB_RegRd != 0) &
        (MEM_WB_RegRd == ID_EX_RegRt))
        FwdB = 1;
    else
        FwdB = 0;
    end
endmodule

module hazard_detection_unit_and_branch(Ins,Rs,Rt,eq,EX_Rd,MEM_Rd,
EX_MemRead,MEM_MemRead,EX_RW,MEM_RW,PC_write,B_Sel,
IF_ID_write,IF_ID_flush,nop);
input  [5:0]  Ins;
input  [4:0]  Rs, Rt, EX_Rd, MEM_Rd;
input      EX_MemRead,MEM_MemRead,eq,EX_RW,MEM_RW;
output      nop,PC_write,IF_ID_write,IF_ID_flush;
output [1:0]  B_Sel;
reg        nop,PC_write,IF_ID_write,IF_ID_flush;
reg        [1:0]  B_Sel;

initial
    begin
        nop=0;
        PC_write=1;
        IF_ID_write=1;
        IF_ID_flush=0;
        B_Sel=1;
    end

always@(Ins,Rs,Rt,EX_Rd,MEM_Rd,EX_MemRead,MEM_MemRead,EX_RW,MEM_RW,eq)
    if (Ins == 6'b111111)
        begin
            PC_write = 0;
            IF_ID_write=0;
            nop = 1;
        end
    else if(Ins==6'b000010)
        begin
            B_Sel=2;
            PC_write=1;
        end
endmodule

```

```

IF_ID_flush=1;
IF_ID_write=1;
nop=0;
end
else if(Ins==6'b000100)
begin
if(((EX_RW==1'b1) && (EX_Rd!=0) && (EX_Rd==Rs || EX_Rd==Rt)) ||
((MEM_RW==1'b1) && (MEM_Rd!=0) && (MEM_Rd==Rs || MEM_Rd==Rt)))
begin
B_Sel=1;
PC_write=0;
IF_ID_write=0;
IF_ID_flush=0;
nop=1;
end
else if(eq==1'b1)
begin
B_Sel=0;
IF_ID_flush=1;
IF_ID_write=0;
PC_write=1;
nop=0;
end
else
begin
B_Sel=1;
IF_ID_flush=0;
IF_ID_write=1;
PC_write=1;
nop=0;
end
end
else if(EX_MemRead==1'b1 && (EX_Rd==Rs || EX_Rd==Rt))
begin
B_Sel=1;
PC_write=0;
IF_ID_write=0;
IF_ID_flush=0;
nop=1;
end
else
begin
B_Sel=1;
PC_write=1;
IF_ID_write=1;
IF_ID_flush=0;
nop=0;
end
end
endmodule

```

```

module an_counter(an, clkIn, reset);
    output [3:0] an;
    input reset, clkIn;
    reg [3:0] an;
    reg [31:0] c;

    always @ (posedge clkIn or posedge reset)
        if (reset)
            begin
                an <= 4'b1110;
                c <= 0;
            end
        else if (c == 32'h0003D090)
            begin
                c <= 0;
                if (~an[0])
                    an <= 4'b1101;
                else if (~an[1])
                    an <= 4'b1011;
                else if (~an[2])
                    an <= 4'b0111;
                else if (~an[3])
                    an <= 4'b1110;
            end
        else
            c <= c + 1;
    endmodule

```

```

module divider(clkOut, clkIn, clkRst);
    output clkOut;
    input clkIn, clkRst;
    reg [31:0] c;
    reg clkOut;

    always @ (posedge clkRst or posedge clkIn)
        if (clkRst)
            begin
                clkOut <= 1;
                c <= 0;
            end
        else if (c == 32'h02FAF07F)
            begin
                clkOut <= 1;
                c <= 0;
            end
        else if (c < 32'h017D783F)
            begin
                clkOut <= 1;
            end

```

```

        c <= c + 1;
    end
else
    begin
        clkOut <= 0;
        c <= c + 1;
    end
endmodule

```

```

module select(qx, dx, q0, q1, q2, q3, d0, d1, d2, d3, clkIn, reset);
    input  [3:0]  q0, q1, q2, q3;
    input                d0, d1, d2, d3, clkIn, reset;
    output [3:0]  qx;
    output                dx;
    reg  [1:0]  flag;
    reg  [3:0]  qx;
    reg  [31:0] c;
    reg                dx;

    always @ (posedge clkIn or posedge reset)
        if (reset)
            begin
                flag <= 0;
                c <= 0;
                qx <= q0;
            end
        else if (c == 32'h0003D090)
            begin
                flag <= flag + 1;
                c <= 0;
                if (flag == 3)
                    begin
                        qx <= q0;
                        dx <= d0;
                    end
                else if (flag == 0)
                    begin
                        qx <= q1;
                        dx <= d1;
                    end
                else if (flag == 1)
                    begin
                        qx <= q2;
                        dx <= d2;
                    end
                else if (flag == 2)
                    begin
                        qx <= q3;
                        dx <= d3;
                    end
            end

```

```

        end
    end
else
    c <= c + 1;
endmodule

```

```

module SSD_4_sign(c, d, disp);
    output [6:0] c;
    input [15:0] d;
    input disp;

    assign c[0] = (d[1] | d[4] | d[11] | d[13]) | (~disp);
    assign c[1] = (d[5] | d[6] | d[11] | d[12] | d[14] | d[15]) | (~disp);
    assign c[2] = (d[2] | d[12] | d[14] | d[15]) | (~disp);
    assign c[3] = (d[1] | d[4] | d[7] | d[10] | d[15]) | (~disp);
    assign c[4] = (d[1] | d[3] | d[4] | d[5] | d[7] | d[9]) | (~disp);
    assign c[5] = (d[1] | d[2] | d[3] | d[7] | d[13]) | (~disp);
    assign c[6] = (d[0] | d[1] | d[7] | d[12]) | (~disp);
endmodule

```

```

module decoder_4_16(d, q);
    output[15:0] d;
    input [3:0] q;

    assign d[0] = (~q[3] & (~q[2] & (~q[1] & (~q[0]));
    assign d[1] = (~q[3] & (~q[2] & (~q[1] & (q[0]));
    assign d[2] = (~q[3] & (~q[2] & (q[1] & (~q[0]));
    assign d[3] = (~q[3] & (~q[2] & (q[1] & (q[0]));
    assign d[4] = (~q[3] & (q[2] & (~q[1] & (~q[0]));
    assign d[5] = (~q[3] & (q[2] & (~q[1] & (q[0]));
    assign d[6] = (~q[3] & (q[2] & (q[1] & (~q[0]));
    assign d[7] = (~q[3] & (q[2] & (q[1] & (q[0]));
    assign d[8] = (q[3] & (~q[2] & (~q[1] & (~q[0]));
    assign d[9] = (q[3] & (~q[2] & (~q[1] & (q[0]));
    assign d[10] = (q[3] & (~q[2] & (q[1] & (~q[0]));
    assign d[11] = (q[3] & (~q[2] & (q[1] & (q[0]));
    assign d[12] = (q[3] & (q[2] & (~q[1] & (~q[0]));
    assign d[13] = (q[3] & (q[2] & (~q[1] & (q[0]));
    assign d[14] = (q[3] & (q[2] & (q[1] & (~q[0]));
    assign d[15] = (q[3] & (q[2] & (q[1] & (q[0]));
endmodule

```

## Appendix D: Pipelined CPU Test Bench

```
module Pipeline_test;
    reg clk,reset;
    reg [4:0] check_input;
    wire [6:0] cNum;
    wire [3:0] anNum;
    wire [7:0] light;
    Pipeline uut (
        .clk(clk),
        .check_input(check_input),
        .reset(reset),
        .cNum(cNum),
        .anNum(anNum),
        .light(light)
    );
    initial begin
        clk = 1;
        reset = 0;
    end
    always #20 clk = ~clk;
endmodule
```



## Appendix E: Simulation Result for TA's Test Case with Single-cycle CPU

```
Time:          0 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          0 [clk] = 0
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          1 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          1 [clk] = 0
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          2 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          2 [clk] = 0
[$s0] = 00000057 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          3 [clk] = 1
[$s0] = 00000057 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
```































































































































```
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      199 [clk] = 1
[$s0] = 00000057 [$s1] = ffffffff9 [$s2] = 00000020
[$s3] = 00000037 [$s4] = 00000001 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      199 [clk] = 0
[$s0] = 00000057 [$s1] = ffffffff9 [$s2] = 00000020
[$s3] = 00000037 [$s4] = 00000001 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      200 [clk] = 1
[$s0] = 00000057 [$s1] = ffffffff9 [$s2] = 00000020
[$s3] = 00000037 [$s4] = 00000001 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
```

## Appendix F: Simulation Result for TA's Test Case with Pipelined CPU

```
Time:          0 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          0 [clk] = 0
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          1 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          1 [clk] = 0
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          2 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          2 [clk] = 0
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
[$t1] = 00000000 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:          3 [clk] = 1
[$s0] = 00000000 [$s1] = 00000000 [$s2] = 00000000
[$s3] = 00000000 [$s4] = 00000000 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000000
```































































































































```
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      199 [clk] = 1
[$s0] = 00000057 [$s1] = ffffffff9 [$s2] = 00000020
[$s3] = 00000037 [$s4] = 00000001 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      199 [clk] = 0
[$s0] = 00000057 [$s1] = ffffffff9 [$s2] = 00000020
[$s3] = 00000037 [$s4] = 00000001 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
Time:      200 [clk] = 1
[$s0] = 00000057 [$s1] = ffffffff9 [$s2] = 00000020
[$s3] = 00000037 [$s4] = 00000001 [$s5] = 00000000
[$s6] = 00000000 [$s7] = 00000000 [$t0] = 00000020
[$t1] = 00000037 [$t2] = 00000000 [$t3] = 00000000
[$t4] = 00000000 [$t5] = 00000000 [$t6] = 00000000
[$t7] = 00000000 [$t8] = 00000000 [$t9] = 00000000
```