VE370 Project 2 Report

Team Member:
Pan Qiying 517370910039
Richard Murray 516370990002
Sun Yiwen 517370910213

1. Objective

- To model both single cycle and pipeline implementation of MIPS computer in Verilog that supports a subset of MIPS instruction set including `lw`, `sw`, `add`, `addi`, `sub`, `and`, `andi`, `or`, `slt`, `beq`, `bne`, and `j`.
- To implement forwarding in the pipelined structure to handle data and control hazards including the data hazards involving branch instructions.
- To demonstrate the pipeline processor on the FPGA board.

2. Top Level Diagram

2.1 Top Level Diagram for single cycle implementation

The top level diagram for single cycle implementation is shown in Figure 1.

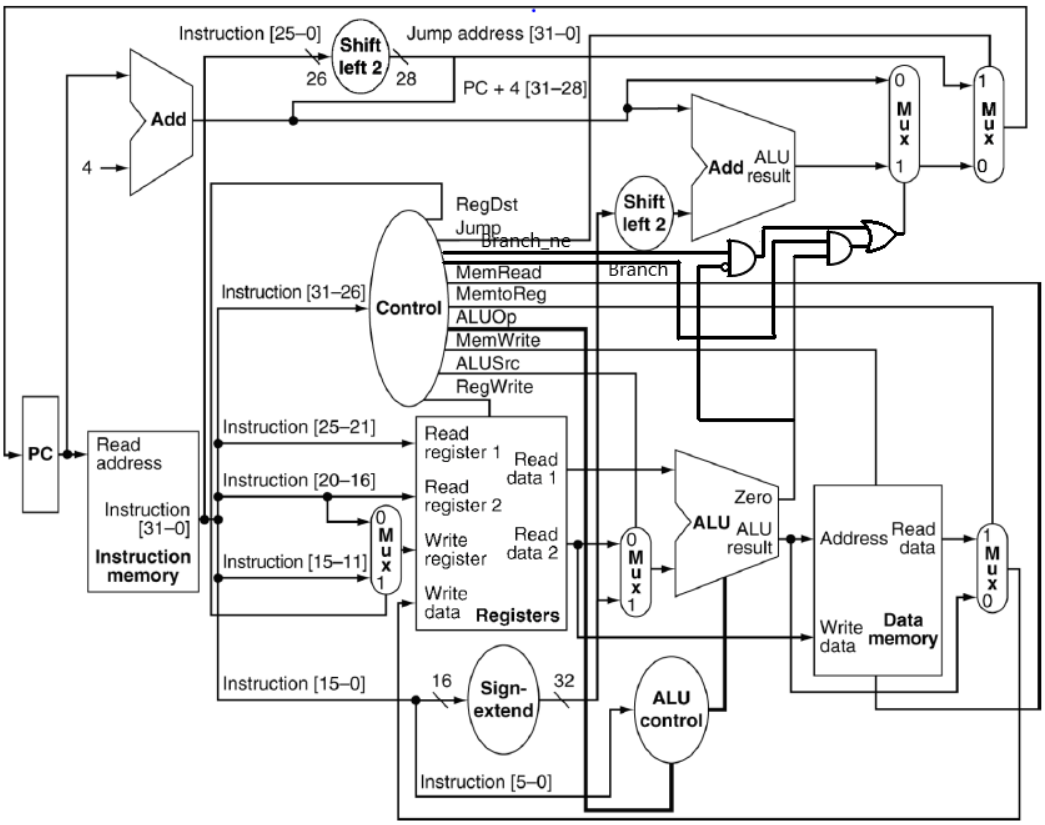

Figure 1: Top Level Design for single cycle processor

We also generate the RTL design for the single cycle processsor with Vivado.

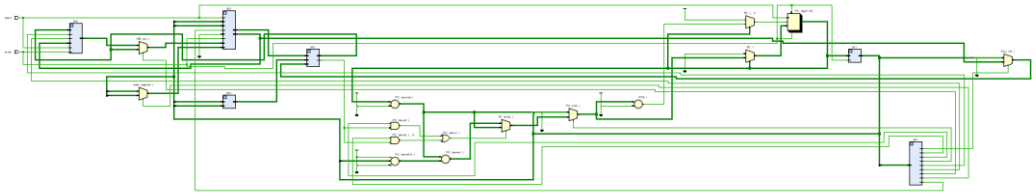

Figure 2: RTL design for single cycle processor

2.2 Top Level Diagram for pipeline implementation

The top level diagram for pipeline processor omits some components to support `bne` and `j` instruction and some blocks to cope with control and data hazards. We utilize "assume branch not taken" method to resolve control hazards.

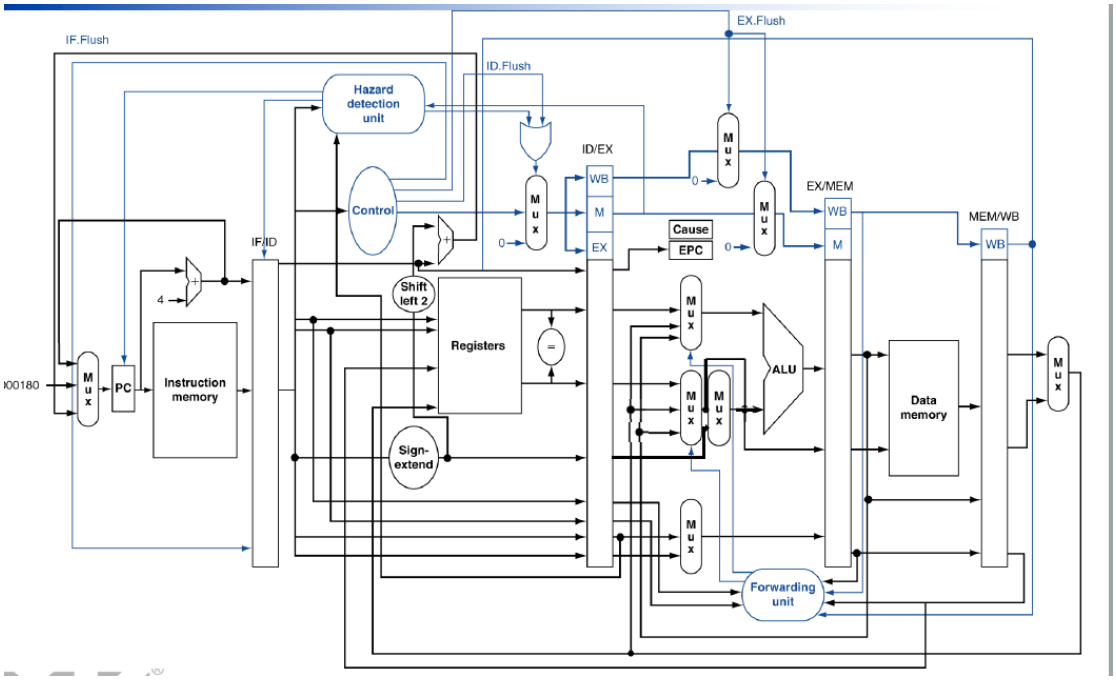

Figure 3: Top level design for pipeline processor

We also generate the RTL design for the pipeline processsor with Vivado.

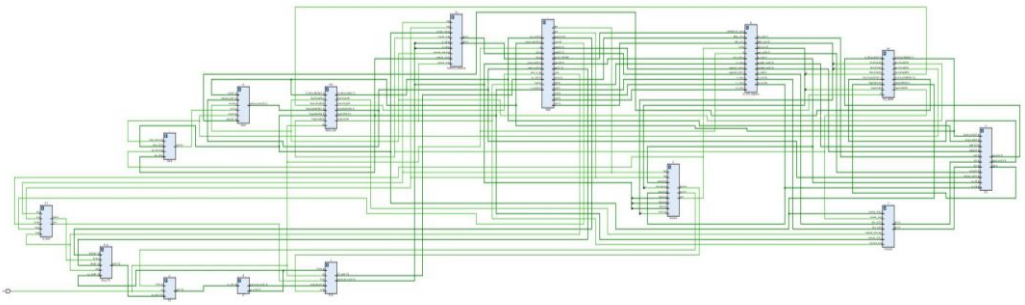

Figure 4: RTL design for the pipeline processor

3. Components Design

3.1 PC register

The PC takes an address and a control signal as the input, if PCWrite is true then the output address will be set as the address of the PC. Otherwise the output address will be set to zero and the output address is also initialized to be zero.

```
module PC(PCWrite, pc_prev, pc, clk);
    input PCWrite,clk;
    input [31:0] pc_prev;
    output reg [31:0] pc;

    initial
```

```
      pc=32'b0;

    always @(posedge clk) begin
        if (PCWrite)
            pc = pc_prev;

    end
endmodule
```

3.2 Instruction memory

The instruction memory takes the PC address as an input and outputs the code of the next instruction. We are given a list of instructions here which have been placed in an array in a 32-bit register.

```
module InstructionMem(pc, instr);
    input [31:0] pc;
    output [31:0] instr;
    reg [31:0] memory [30:0];
    reg[31:0] instr;

    initial begin
   memory[0] = 32'b00100000000010000000000000100000; //addi
$t0, $zero, 0x20
    memory[1] = 32'b00100000000010010000000000110111;
//addi $t1, $zero, 0x37
    memory[2] = 32'b00000001000010011000000000100100; //and
$s0, $t0, $t1
    memory[3] = 32'b00000001000010011000000000100101; //or
$s0, $t0, $t1
    memory[4] = 32'b10101100000100000000000000000100; //sw
$s0, 4($zero)
    memory[5] = 32'b10101100000010000000000000001000; //sw
$t0, 8($zero)
    memory[6] = 32'b00000001000010011000100000100000; //add
$s1, $t0, $t1
    memory[7] = 32'b00000001000010011001000000100010; //sub
$s2, $t0, $t1
    memory[8] = 32'b00010010001100100000000000001001; //beq
$s1, $s2, error0
    memory[9] = 32'b10001100000100010000000000000100; //lw
$s1, 4($zero)
    memory[10]= 32'b00110010001100100000000001001000;
//andi $s2, $s1, 0x48
    memory[11] =32'b00010010001100100000000000001001;
//beq $s1, $s2, error1
```

```
    memory[12] =32'b10001100000100110000000000001000; //lw
$s3, 8($zero)
    memory[13] =32'b00010010000100110000000000001010;
//beq $s0, $s3, error2
    memory[14] =32'b00000010010100011010000000101010;
//slt $s4, $s2, $s1 (Last)
    memory[15] =32'b00010010100000000000000000001111;
//beq $s4, $0, EXIT
    memory[16] =32'b00000010001000001001000000100000;
//add $s2, $s1, $0
    memory[17] =32'b00001000000000000000000000001110; //j
Last
    memory[18] =32'b00100000000010000000000000000000;
//addi $t0, $0, 0(error0)
    memory[19] =32'b00100000000010010000000000000000;
//addi $t1, $0, 0
    memory[20] =32'b00001000000000000000000000011111; //j
EXIT
    memory[21] =32'b00100000000010000000000000000001;
//addi $t0, $0, 1(error1)
    memory[22] =32'b00100000000010010000000000000001;
//addi $t1, $0, 1
    memory[23] =32'b00001000000000000000000000011111; //j
EXIT
    memory[24] =32'b00100000000010000000000000000010;
//addi $t0, $0, 2(error2)
    memory[25] =32'b00100000000010010000000000000010;
//addi $t1, $0, 2
    memory[26] =32'b00001000000000000000000000011111; //j
EXIT
    memory[27] =32'b00100000000010000000000000000011;
//addi $t0, $0, 3(error3)
    memory[28] =32'b00100000000010010000000000000011;
//addi $t1, $0, 3
    memory[29] =32'b00001000000000000000000000011111; //j
EXIT
    end

    always @(pc) begin
         if (pc>>2>29) begin
             $display("=================");
             $stop;
         end
         instr=memory[pc>>2];
```

```
    end
endmodule
```

3.3 IF/ID pipeline register

The IF/ID pipeline register loads the new PC and instruction from IF stage to ID stage. A write signal controls whether to load the new inputs and a flush signal clears all the outputs to zero when flush is true.

```
module IFID(write, flush, PC, instruction, PC_out,
instruction_out,clk);
  input write, flush,clk;
  input [31:0] PC,instruction;
  output [31:0] PC_out,instruction_out;
  reg [31:0] PC_out,instruction_out;

  initial
  begin
    PC_out=32'b0;
    instruction_out=32'b0;
  end

  always @(posedge clk) begin
    if ({flush,write}==2'b11)
       instruction_out=32'b0;
    else if ({flush,write}==2'b01)
       instruction_out=instruction;
    if (write==1)
       PC_out=PC;
  end
endmodule
```

3.4 Register

The register holds 32 32-bit numbers with assigned 5-bit address. It always reads data out while it writes data only when the regWrite signal is true.

```
module
register(read1,read2,write,w_data,data1,data2,regWrite,cl
k,registersel, FPGAout);
input [4:0] read1,read2,write, registersel;
input [31:0] w_data;
input regWrite,clk;
output [31:0] data1,data2,FPGAout;
reg[31:0] data1,data2;
reg[31:0] register[31:0], FPGAout;
initial
```

```
begin
       register[0]=32'b0;
       register[1]=32'b0;
       register[2]=32'b0;
       register[3]=32'b0;
       register[4]=32'b0;
       register[5]=32'b0;
       register[6]=32'b0;
       register[7]=32'b0;
       register[8]=32'b0;
       register[9]=32'b0;
       register[10]=32'b0;
       register[11]=32'b0;
       register[12]=32'b0;
       register[13]=32'b0;
       register[14]=32'b0;
       register[15]=32'b0;
       register[16]=32'b0;
       register[17]=32'b0;
       register[18]=32'b0;
       register[19]=32'b0;
       register[20]=32'b0;
       register[21]=32'b0;
       register[22]=32'b0;
       register[23]=32'b0;
       register[24]=32'b0;
       register[25]=32'b0;
       register[26]=32'b0;
       register[27]=32'b0;
       register[28]=32'b0;
       register[29]=32'b0;
       register[30]=32'b0;
       register[31]=32'b0;
end


always @(read1,read2,write,w_data,regWrite)
begin
  data1=register[read1];
  data2=register[read2];
end
always@(*) begin
FPGAout=register[registersel];
end
```

```
always@(negedge clk)
begin
  if (regWrite==1)
  register[write]=w_data;
end


always @(posedge clk)
begin
  $display("[$s0]=%h,[$s1]=%h,[$s2]=%h",register[16],regi
ster[17],register[18]);
  $display("[$s3]=%h,[$s4]=%h,[$s5]=%h",register[19],regi
ster[20],register[21]);
  $display("[$s6]=%h,[$s7]=%h,[$t0]=%h",register[22],regi
ster[23],register[8]);
  $display("[$t1]=%h,[$t2]=%h,[$t3]=%h",register[9],regis
ter[10],register[11]);
  $display("[$t4]=%h,[$t5]=%h,[$t6]=%h",register[12],regi
ster[13],register[14]);
  $display("[$t7]=%h,[$t8]=%h,[$t9]=%h",register[15],regi
ster[24],register[25]);
end
endmodule
```

3.5 Sign extension component

The sign extension component extends a 16-bit number to a 32-bit number based on its sign.

module signextension(imme,imme_out);

```
module signextension(imme,imme_out);
input [15:0] imme;
output [31:0] imme_out;

reg [31:0] imme_out;
always @(imme)
  imme_out={{16{imme[15]}},imme[15:0]};
endmodule
```

3.6 Control component

The control component takes the opcode (the first 6 bits of a instruction code) as the input and outputs all the needed control signals for the given instruction.

```
module control(opcode,wb,m,ex,jump,beq,bne,funct);
input [5:0] opcode,funct;
```

```
output jump,beq,bne;
output [1:0] wb,m;
output [3:0] ex;

reg jump,beq,bne;
reg [1:0] wb,m;
reg [3:0] ex;

initial begin
wb=2'b00;m=2'b00;ex=4'b0100;jump=0;beq=0;bne=0;
end

always @(opcode,funct)begin
   case (opcode)
      6'b100011:begin
wb=2'b11;m=2'b10;ex=4'b0001;jump=0;beq=0;bne=0; end
      6'b101011:begin
wb=2'b00;m=2'b01;ex=4'b0001;jump=0;beq=0;bne=0; end
      6'b001000:begin
wb=2'b10;m=2'b00;ex=4'b0001;jump=0;beq=0;bne=0; end
      6'b001100:begin
wb=2'b10;m=2'b00;ex=4'b0111;jump=0;beq=0;bne=0; end
      6'b000100:begin
wb=2'b00;m=2'b00;ex=4'b0010;jump=0;beq=1;bne=0; end
      6'b000101:begin
wb=2'b00;m=2'b00;ex=4'b0010;jump=0;beq=0;bne=1; end
      6'b000010:begin
wb=2'b00;m=2'b00;ex=4'b0000;jump=1;beq=0;bne=0; end
      6'b000000:begin
wb=2'b10;m=2'b00;ex=4'b1100;jump=0;beq=0;bne=0; end
      default: begin
wb=2'b00;m=2'b00;ex=4'b0100;jump=0;beq=0;bne=0;end
   endcase
   if ({opcode,funct}==12'b0)
      begin
wb=2'b00;m=2'b00;ex=4'b0100;jump=0;beq=0;bne=0;end
end
endmodule
```

3.7 ID/EX pipeline register

```
module ID_EX_register(EX_out,MEM_out,WB_out,regdata1_out,
regdata2_out,imm_out,rs_out,rt1_out,rd_out,EX_in,MEM_in,W
B_in,regdata1_in,rt2_out,regdata2_in,imm_in,rs_in,rt1_in,
rd_in,rt2_in,clk,bubble);
```

```
input [1:0]WB_in,MEM_in;
input [3:0]EX_in;
input clk,bubble;
input [31:0]regdata1_in,regdata2_in,imm_in;
input [4:0]rs_in,rt1_in,rd_in,rt2_in;
output [1:0]WB_out,MEM_out;
output [3:0]EX_out;
output [31:0]regdata1_out,regdata2_out,imm_out;
output [4:0]rs_out,rt1_out,rd_out,rt2_out;
reg [1:0]WB_out,MEM_out;
reg [3:0]EX_out;
reg [31:0]regdata1_out,regdata2_out,imm_out;
reg [4:0]rs_out,rt1_out,rd_out,rt2_out;
reg [133:0]ID_EX;

initial begin
   WB_out=2'b0;
   MEM_out=2'b0;
   EX_out=4'b0;
   regdata1_out=32'b0;
   regdata2_out=32'b0;
   imm_out=32'b0;
   rs_out=5'b0;
   rt1_out=5'b0;
   rd_out=5'b0;
   rt2_out=5'b0;
   ID_EX=134'b0;
end

always @(posedge clk)begin
    if (!bubble) begin
        ID_EX[133:132]=WB_in;
        ID_EX[130:127]=EX_in;
        ID_EX[126:125]=MEM_in;
    end else begin
        ID_EX[133:132]=2'b0;
        ID_EX[130:127]=4'b0;
        ID_EX[126:125]=2'b0;
    end
        ID_EX[122:118]=rt2_in;
        ID_EX[110:79]=regdata1_in;
        ID_EX[78:47]=regdata2_in;
        ID_EX[46:15]=imm_in;
        ID_EX[14:10]=rs_in;
```

```
            ID_EX[9:5]=rt1_in;
            ID_EX[4:0]=rd_in;
        WB_out=ID_EX[133:132];
        EX_out=ID_EX[130:127];
        MEM_out=ID_EX[126:125];
        rt2_out=ID_EX[122:118];
        regdata1_out=ID_EX[110:79];
        regdata2_out=ID_EX[78:47];
        imm_out=ID_EX[46:15];
        rs_out=ID_EX[14:10];
        rt1_out=ID_EX[9:5];
        rd_out=ID_EX[4:0];
    end
endmodule
```

3.8 ALU

```
module ALU(Result,A,B,ALUControl);
    parameter n=32;
    input [n-1:0]A;
    input [n-1:0]B;
    input [3:0]ALUControl;
    output [n-1:0]Result;
    reg [n-1:0]Result;

    initial begin
       Result=32'b0;
    end

    always @(A,B,ALUControl) begin
        case(ALUControl)
            4'b0010: Result=A+B;
            4'b0110: Result=A+(~B)+32'b1;
            4'b0000: Result=A&B;
            4'b0001: Result=A|B;
            4'b0111: Result=(A+(~B)+32'b1)>>31;
            default Result=32'b0;
        endcase
    end

endmodule
```

3.9 EX/MEM pipeline register

The EX/MEM pipeline registers takes all the outputs and control signals from the EX

stage as inputs and outputs them into the MEM stage.

```
module EX_MEM(MemReadEX, MemtoRegEX, MemWriteEX, RegWriteEX,
ALUResultEX, MuxForwardEX, RegDstEX, MemReadMEM,
MemtoRegMEM, MemWriteMEM, RegWriteMEM, ALUResultMEM,
MuxForwardMEM, RegDstMEM,clk);
    input MemReadEX, MemtoRegEX, MemWriteEX,
RegWriteEX,clk;
    input [31:0] ALUResultEX, MuxForwardEX;
    input [4:0] RegDstEX;
    output reg MemReadMEM, MemtoRegMEM, MemWriteMEM,
RegWriteMEM;
    output reg [31:0] ALUResultMEM, MuxForwardMEM;
    output reg [4:0]RegDstMEM;

    initial begin
        MemReadMEM<=1'b0;
        MemtoRegMEM<=1'b0;
        MemWriteMEM<=1'b0;
        RegWriteMEM<=1'b0;
    end

    always @(posedge clk) begin
        MemReadMEM<=MemReadEX;
        MemtoRegMEM<=MemtoRegEX;
        MemWriteMEM<=MemWriteEX;
        RegWriteMEM<=RegWriteEX;
        ALUResultMEM<=ALUResultEX;
        MuxForwardMEM<=MuxForwardEX;
        RegDstMEM<=RegDstEX;
    end
endmodule
```

3.10 Data memory

The data memory takes the output from the EX/MEM pipeline registers as inputs. An array of size 32 is initialized to zero and stored in a 32-bit register. When MemWrite is true, the data will be written into the next memory block. When MemRead is true, it will output the next data meomory.

```
module DataMem(MemWrite, MemRead, WriteData, ALU_address,
ReadData, clk);
    input MemWrite, MemRead, clk;
    input [31:0] WriteData, ALU_address;
    output [31:0] ReadData;
    reg [31:0] DataMemory [31:0];
    wire [31:0] i;
```

```
    assign i = ALU_address >> 2;
    initial begin
       DataMemory[0]=32'b0;
       DataMemory[1]=32'b0;
       DataMemory[2]=32'b0;
       DataMemory[3]=32'b0;
       DataMemory[4]=32'b0;
       DataMemory[5]=32'b0;
       DataMemory[6]=32'b0;
       DataMemory[7]=32'b0;
       DataMemory[8]=32'b0;
       DataMemory[9]=32'b0;
       DataMemory[10]=32'b0;
       DataMemory[11]=32'b0;
       DataMemory[12]=32'b0;
       DataMemory[13]=32'b0;
       DataMemory[14]=32'b0;
       DataMemory[15]=32'b0;
       DataMemory[16]=32'b0;
       DataMemory[17]=32'b0;
       DataMemory[18]=32'b0;
       DataMemory[19]=32'b0;
       DataMemory[20]=32'b0;
       DataMemory[21]=32'b0;
       DataMemory[22]=32'b0;
       DataMemory[23]=32'b0;
       DataMemory[24]=32'b0;
       DataMemory[25]=32'b0;
       DataMemory[26]=32'b0;
       DataMemory[27]=32'b0;
       DataMemory[28]=32'b0;
       DataMemory[29]=32'b0;
       DataMemory[30]=32'b0;
       DataMemory[31]=32'b0;
    end


    always@(posedge clk) begin
        if (MemWrite==1'b1) DataMemory[i] = WriteData;
        else DataMemory[i] = DataMemory[i];
    end
    assign ReadData = (MemRead==1'b1)? DataMemory[i]:32'b0;
endmodule
```

3.11 MEM/WB pipeline register

The MEM/WB pipeline register takes all the outputs and control signals from the MEM stage as inputs and outputs them into the WB stage.

```
module MEM_WB(MemtoRegMEM, RegWriteMEM, ReadDataMEM,
ALUResultMEM, RegDstMEM,
MemtoRegWB, RegWriteWB, ReadDataWB, ALUResultWB,
RegDstWB,MemReadMem,MemReadWB,clk);
    input MemtoRegMEM, RegWriteMEM,MemReadMem,clk;
    input [31:0] ReadDataMEM, ALUResultMEM;
    input [4:0] RegDstMEM;
    output reg MemtoRegWB, RegWriteWB,MemReadWB;
    output reg [31:0] ReadDataWB, ALUResultWB;
    output reg [4:0] RegDstWB;

    initial begin
    MemtoRegWB<=1'b0;
    RegWriteWB<=1'b0;
    end

    always @(posedge clk) begin
        MemtoRegWB<=MemtoRegMEM;
        RegWriteWB<=RegWriteMEM;
        ReadDataWB<=ReadDataMEM;
        ALUResultWB<=ALUResultMEM;
        RegDstWB<=RegDstMEM;
        MemReadWB<=MemReadMem;
    end
endmodule
```

4. Data Hazard Handling

4.1 Forwarding unit

The forwarding unit resolve the data hazards involving R-type instructions. For example, if the previous instruction is `add $1, $1, $0` and the current is `add $2, $1, $0`, it is a typical type of data hazard which forwarding unit deals with. A forwarding unit detects the related hazards and links the ALU output of the previous instruction directly with the ALU input of the current instruction to eliminate the hazard by sending signals to the muxes which choose the ALU inputs.

```
module
forward(memwb_write,memwb_rd,idex_rs,idex_rt,exmem_write,
exmem_rd,fa,fb,memwb_memread);
input memwb_write,exmem_write,memwb_memread;
input [4:0] memwb_rd,idex_rs,idex_rt,exmem_rd;
output [1:0] fa,fb;
reg [1:0] fa,fb;
```

```
initial begin
 fa=2'b00;
 fb=2'b00;
end


always @(memwb_write,memwb_rd,idex_rs,idex_rt,exmem_write,
exmem_rd,memwb_memread)begin
   fa=2'b00;
   fb=2'b00;
   if (exmem_rd!=5'b0)
     if ({exmem_write,exmem_rd}=={1'b1,idex_rs})
       fa=2'b10;
   if (fa!=2'b10)
     if (memwb_rd!=5'b0)
       if ({memwb_write,memwb_rd}=={1 'b1,idex_rs})
        if (memwb_memread==0)
         fa=2'b01;
        else
         fa=2'b11;
   if (exmem_rd!=5'b0)
       if ({exmem_write,exmem_rd}=={1'b1,idex_rt})
          fb=2'b10;
   if (fb!=2'b10)
      if (memwb_rd!=5'b0)
        if ({memwb_write,memwb_rd}=={1'b1,idex_rt})
         if (memwb_memread==0)
           fb=2'b01;
         else
           fb=2'b11;
end
endmodule
```

To deal with the data hazards in the branch instructions, a new forwarding unit is introduced before the comparator of the register read data.

```
module
forward_compare(memwb_write,memwb_rd,id_rt,id_rs,exmem_wr
ite,exmem_rd,fa,fb,memwb_memread,beq,bne);
input memwb_write,exmem_write,memwb_memread,beq,bne;
input [4:0] memwb_rd,id_rt,id_rs,exmem_rd;
output [1:0] fa,fb;
reg [1:0] fa,fb;
```

```
reg branch;
initial begin
  fa=2'b00;
  fb=2'b00;
end


always
@(memwb_write,exmem_write,memwb_rd,id_rt,id_rs,exmem_rd,
memwb_memread,beq,bne)
begin
  fa=2'b00;
  fb=2'b00;
  branch=beq||bne;
  if (exmem_rd!=5'b0)
     if ({exmem_write,exmem_rd,branch}=={1'b1,id_rs,1'b1})
        fa=2'b10;
  if (fa!=2'b10)
     if (memwb_rd!=5'b0)
        if
({memwb_write,memwb_rd,branch}=={1'b1,id_rs,1'b1})
          if (memwb_memread!=1)
            fa=2'b01;
          else
            fa=2'b11;
   if (exmem_rd!=5'b0)
      if
({exmem_write,exmem_rd,branch}=={1'b1,id_rt,1'b1})
                    fb=2'b10 ;
   if (fb!=2'b10)
       if (memwb_rd!=5'b0)
           if
({memwb_write,memwb_rd,branch}=={1'b1,id_rt,1'b1})
             if (memwb_memread!=1)
                 fb=2'b01;
             else
                 fb=2'b11;
end
endmodule
```

4.2 Hazard detection unit

This hazard detection unit is useful when a stall is needed.

```
module
hazard(ifidwrite,stall,memread,idexrt,ifidrt,ifidrd,
```

```
pcwrite,bne,beq,idexrd,ifidrs,idexregwrite,settle,stall_f
inish,exmemrt,exmemread,exmemregwrite);
input memread,bne,beq,idexregwrite,stall_finish,exmemread,
exmemregwrite;
input [4:0] idexrt,ifidrt,ifidrd,idexrd,ifidrs,exmemrt;
output ifidwrite,stall,pcwrite,settle;


reg ifidwrite,stall,pcwrite,stall2,settle;


initial
begin
  ifidwrite=1;
  stall=0;
  pcwrite=1;
  stall2=0;
  settle=0;
end
always
@(memread,idexrt,ifidrt,ifidrd,bne,beq,idexrd,idexregwrit
e,
ifidrs,stall_finish,exmemrt,exmemread,exmemregwrite)
begin

     stall=0;
     stall2=0;


  stall=stall || (memread &&(idexrt!=5'b0)
&&((idexrt==ifidrt)||(idexrt==ifidrs))&&idexregwrite);
  stall=stall || (exmemread && (exmemrt!=5'b0) &&
((exmemrt==ifidrt)||(exmemrt==ifidrs))&&(bne||beq)&&exmem
regwrite);
  stall=stall ||
((idexrd!=5'b0)&&((idexrd==ifidrs)||(idexrd==ifidrt))&&(b
eq||bne)&&idexregwrite);
  ifidwrite=!stall;
  pcwrite=!stall;


end
endmodule
```

5. Control Hazard Handling

5.1 Flush unit

Flush is a way to clear the previous instructions which shouldn't be processed. When a branch or a jump instruction is executed, a flush is needed. Since we use "assume branch not taken" method, we only need to flush the instruction executed in IF stage. Notice that if a bubble is to take place, the processor won't flush anymore.

```
module
if_flush(jump,beq,bne,equal,flush,branch,bubble,stall_fin
ish,settle);
input jump,beq,bne,equal,bubble,settle;
output flush,branch,stall_finish;


reg flush,branch,stall_finish;


initial
   begin
      flush=0;
      branch=0;
      stall_finish=0;
   end


always @(jump,beq,bne,equal,bubble,settle)
begin
  case ({jump,beq,bne,equal,bubble})
    5'b10000:begin flush=1;branch=0;end
    5'b10010:begin flush=1;branch=0;end
    5'b01010:begin flush=1;branch=1;end
    5'b00100:begin flush=1;branch=1;end
    default:begin flush=0;branch=0;end
  endcase
end
endmodule
```

6. SSD and Top Module Design

6.1 SSD

The SSD outputs the number and letter (value of the selected register) onto the SSD of the FPGA board. The register is selected through a 5-bit input and the output between PC or register is selected through a 1-bit input.

```
module Demo(clk,clk2, pc_switch,an, ca, register_sw);
input clk,clk2, pc_switch;
```

```
output [3:0] an;
output reg [6:0] ca;
wire clk1kHz;
reg [4:0] an1, an2, an3, an4;
reg [6:0] ca1, ca2, ca3, ca4;
reg [31:0] tempreg [0:31];
wire [31:0] PC,REG;
input [4:0] register_sw;
top pipeline(clk2, PC, register_sw, REG);


clkdiv_1Hz clkdiv1k(clk,clk1kHz);
RingCounter count(an, clk1kHz);


always@(*) begin
 if (pc_switch==1'b1) begin
  an1<=PC[3:0];
  an2<=PC[7:4];
  an3<=PC[11:8];
  an4<=PC[15:12];
 end
 else begin
  an1<=REG[3:0];
  an2<=REG[7:4];
  an3<=REG[11:8];
  an4<=REG[15:12];
 end
end

always@(an1) begin
 case (an1)
 4'b0000: ca1=7'b1000000;
 4'b0001: ca1=7'b1111001;
 4'b0010: ca1=7'b0100100;
 4'b0011: ca1=7'b0110000;
 4'b0100: ca1=7'b0011001;
 4'b0101: ca1=7'b0010010;
 4'b0110: ca1=7'b0000010;
 4'b0111: ca1=7'b1111000;
 4'b1000: ca1=7'b0000000;
 4'b1001: ca1=7'b0010000;
 4'b1010: ca1=7'b0001000;
 4'b1011: ca1=7'b0000011;
```

```
 4'b1100: ca1=7'b1000110;
 4'b1101: ca1=7'b0100001;
 4'b1110: ca1=7'b0000100;
 4'b1111: ca1=7'b0001110;
 default ca1=7'b0000000;
 endcase
end

always@(an2) begin
  case (an2)
  4'b0000: ca2=7'b1000000;
  4'b0001: ca2=7'b1111001;
  4'b0010: ca2=7'b0100100;
  4'b0011: ca2=7'b0110000;
  4'b0100: ca2=7'b0011001;
  4'b0101: ca2=7'b0010010;
  4'b0110: ca2=7'b0000010;
  4'b0111: ca2=7'b1111000;
  4'b1000: ca2=7'b0000000;
  4'b1001: ca2=7'b0010000;
  4'b1010: ca2=7'b0001000;
  4'b1011: ca2=7'b0000011;
  4'b1100: ca2=7'b1000110;
  4'b1101: ca2=7'b0100001;
  4'b1110: ca2=7'b0000100;
  4'b1111: ca2=7'b0001110;
  default ca2=7'b0000000;
  endcase
end


always@(an3) begin
 case (an3)
 4'b0000: ca3=7'b1000000;
 4'b0001: ca3=7'b1111001;
 4'b0010: ca3=7'b0100100;
 4'b0011: ca3=7'b0110000;
 4'b0100: ca3=7'b0011001;
 4'b0101: ca3=7'b0010010;
 4'b0110: ca3=7'b0000010;
 4'b0111: ca3=7'b1111000;
 4'b1000: ca3=7'b0000000;
 4'b1001: ca3=7'b0010000;
 4'b1010: ca3=7'b0001000;
```

```
 4'b1011: ca3=7'b0000011;
 4'b1100: ca3=7'b1000110;
 4'b1101: ca3=7'b0100001;
 4'b1110: ca3=7'b0000100;
 4'b1111: ca3=7'b0001110;
 default ca3=7'b0000000;
 endcase
 end

always@(an4) begin
  case (an4)
  4'b0000: ca4=7'b1000000;
  4'b0001: ca4=7'b1111001;
  4'b0010: ca4=7'b0100100;
  4'b0011: ca4=7'b0110000;
  4'b0100: ca4=7'b0011001;
  4'b0101: ca4=7'b0010010;
  4'b0110: ca4=7'b0000010;
  4'b0111: ca4=7'b1111000;
  4'b1000: ca4=7'b0000000;
  4'b1001: ca4=7'b0010000;
  4'b1010: ca4=7'b0001000;
  4'b1011: ca4=7'b0000011;
  4'b1100: ca4=7'b1000110;
  4'b1101: ca4=7'b0100001;
  4'b1110: ca4=7'b0000100;
  4'b1111: ca4=7'b0001110;
  default ca4=7'b0000000;
  endcase
 end

always@(an) begin
 case(an)
 4'b1110:ca=ca1;
 4'b1101:ca=ca2;
 4'b1011:ca=ca3;
 4'b0111:ca=ca4;
 endcase
end
endmodule
```

6.2 Top module design

6.2.1 Top module design for single cycle processor

```
module SingleCycle(clock,reset);
    input clock,reset;
    reg [31:0]PC;
    wire [31:0]PC_next;
    wire [31:0]PC_normal;
    wire [31:0]PC_temp;
    wire [31:0]PC_jump;
    wire [31:0]PC_branch;
    wire [31:0]instruction;
    wire [4:0]write_register;
    wire RegDst;
    wire [4:0]ins25_21;
    wire [4:0]ins20_16;
    wire [4:0]ins15_11;
    wire [31:0]write_data;
    wire [31:0]read_data1;
    wire [31:0]read_data2;
    wire RegWrite;
    wire [5:0]opcode,funct;
    wire [15:0]imm;
    wire [3:0]ALUControl;
    wire [31:0]ALU_in2;
    wire [31:0]ALU_out;
    wire Zero;
    wire ALUSrc;
    wire MemWrite;
    wire MemRead;
    wire [31:0]DataMem_ReadData;
    wire MemtoReg;
    wire [31:0]DM_out;
    wire [31:0]RA;
    wire Branch;
    wire Branch_ne;
    wire Jump;

    //PC generation
    always @(posedge reset or posedge clock) begin
        if (reset) PC=-4;
        else if (PC==-4) PC=0;
        else if (PC_next<168) PC=PC_next;
    end
```

```
    //Instruction Fetch
    InstructionMem M1 (instruction, PC, reset);
    //Instruction Decode and Register Read/Write
    assign ins25_21=instruction[25:21];
    assign ins20_16=instruction[20:16];
    assign ins15_11=instruction[15:11];
    assign write_register=(RegDst)? ins15_11:ins20_16;
    RegisterFile M2
(ins25_21,ins20_16,write_register,write_data,read_data1,r
ead_data2,RegWrite,clock,reset,regnum,regdata);
    //Instruction Decode and Execution
    assign opcode=instruction[31:26];
    assign funct=instruction[5:0];
    assign imm=instruction[15:0];
    ALU_Control M3 (ALUControl,opcode,funct);
    assign ALU_in2=(ALUSrc)? imm:read_data2;
    ALU M4 (ALU_out,Zero,read_data1,ALU_in2,ALUControl);
    //Access Memory Operand
    DataMem M5
(MemWrite,MemRead,read_data2,ALU_out,DataMem_ReadData,res
et,clock);
    assign DM_out=(MemtoReg)? DataMem_ReadData:ALU_out;
    assign write_data=DM_out;
    //PC selection
    assign PC_normal=PC+4;
    assign
PC_jump={PC_normal[31:28],instruction[25:0],2'b00};
    assign RA={{16{instruction[15]}},instruction[15:0]};
    assign PC_branch=PC_normal+(RA<<2);
    assign PC_temp=((Branch_ne&&(~Zero))||(Branch&&Zero))?
PC_branch:PC_normal;
    assign PC_next=(Jump)? PC_jump:PC_temp;
    //Control Unit
    ControlUnit M7
(opcode,RegDst,Jump,Branch,Branch_ne,MemRead,MemWrite,Mem
toReg,ALUSrc,RegWrite);
endmodule
```

6.2.2 Top module design for pipeline processor

The four pipeline registers are shifters. The PC register is a state register. The components in the four stage are the combinational logic.

```
module top(clk, PC, registersel, FPGAout);
  input clk;
  input [4:0] registersel;
```

```
  output [31:0] PC, FPGAout;
  wire
pcwrite,branch,jump,ifidwrite,flush,bubble,compare,beq,bn
e,
memread_wb,memsrc,stall_finish,settle;
  wire [31:0]
pc_in,pc_out,pc_plus_4_if,bradd,jadd,instr_if,pc_id,
instr_id,memout_wb,aluout_wb,data1_id,data2_id,imme_id,pc
_ex,aluout_ex,aluout_mem,memout_mem,memin_mem,data2_ex,da
ta1_ex,data2_ex_in,imme_ex;
  wire [1:0]
ex_mem,wb_wb,id_wb,id_mem,ex_wb,mem_wb,fa,fb,mem_mem,
fa_id,fb_id;//stage_signal
  wire [4:0]
ex_rt2,id_rd,id_rt1,id_rt2,id_rs,w_addr_wb,ex_rs,
ex_rt1,w_addr_mem,w_addr_ex,ex_rd;
  wire [3:0] id_ex,ex_ex;
  reg [5:0] cycle;


  initial begin
    cycle=6'b0;
    $monitor("Time:",$time,", CLK=%d,
PC=%h",cycle,pc_out);
  end
  always @(posedge clk)
  begin
    cycle=cycle+6'b1;
  end

  hazard
h(ifidwrite,bubble,ex_mem[1],ex_rt2,instr_id[20:16],
instr_id[15:11],pcwrite,bne,beq,ex_rd,instr_id[25:21],ex_
wb[1],settle,stall_finish,w_addr_mem,mem_mem[1],mem_wb[1]
);
  PC p(pcwrite,pc_in,pc_out,clk);
  Mux_PC m_p(branch,jump,pc_plus_4_if,bradd,jadd,pc_in);
  if_flush if_f(jump,beq,bne,compare,flush,branch,bubble,
stall_finish,settle);
  forward_compare fc(wb_wb[1],w_addr_wb,instr_id[20:16],
instr_id[25:21],mem_wb[1],w_addr_mem,fa_id,fb_id,mem_mem[
1],beq,bne);
  forward
f(wb_wb[1],w_addr_wb,ex_rs,ex_rt1,mem_wb[1],w_addr_mem,
```

```
fa,fb,memread_wb);
  lwsw
l(memsrc,w_addr_wb,w_addr_mem,memread_wb,mem_mem[0]);
   MEM_WB
mw(mem_wb[0],mem_wb[1],memout_mem,aluout_mem,w_addr_mem,
wb_wb[0],wb_wb[1],memout_wb,aluout_wb,w_addr_wb,mem_mem[1
],memread_wb,clk);
    EX_MEM
em(ex_mem[1],ex_wb[0],ex_mem[0],ex_wb[1],aluout_ex,
data2_ex,w_addr_ex,mem_mem[1],mem_wb[0],mem_mem[0],mem_wb
[1],aluout_mem,memin_mem,w_addr_mem,clk);
    ID_EX_register
ie(ex_ex,ex_mem,ex_wb,data1_ex,data2_ex_in,
imme_ex,ex_rs,ex_rt1,ex_rd,id_ex,id_mem,id_wb,data1_id,ex
_rt2,data2_id,imme_id,id_rs,id_rt1,id_rd,id_rt2,clk,bubbl
e);
    IFID ii(ifidwrite,flush,pc_plus_4_if,instr_if,pc_id,
instr_id,clk);
  assign PC = pc_out;
  IF iff(pc_out,pc_plus_4_if,instr_if);
  idwb
i(pc_id,instr_id,bradd,jadd,memout_wb,aluout_wb,wb_wb[1],
id_wb,id_ex,id_mem,data1_id,data2_id,imme_id,id_rd,id_rt1
,id_rt2,id_rs,jump,beq,bne,pc_ex,wb_wb[0],w_addr_wb,compa
re,fa_id,fb_id,aluout_mem,clk, registersel, FPGAout);
  EX
e(data1_ex,data2_ex_in,aluout_wb,aluout_mem,memout_wb,fa,
fb,
aluout_ex,imme_ex,ex_ex,data2_ex,ex_rd,ex_rt2,w_addr_ex);
  mem m(memsrc,aluout_mem,memin_mem,memout_wb,memout_mem,
mem_mem[1],mem_mem[0],clk);

endmodule
```

7. Simulation Scenario

```
    memory[0] = 32'b00100000000010000000000000100000;
//addi $t0, $zero, 0x20
    memory[1] = 32'b00100000000010010000000000110111;
//addi $t1, $zero, 0x37
    memory[2] = 32'b00000001000010011000000000100100; //and
$s0, $t0, $t1
```

```
    memory[3] = 32'b00000001000010011000000000100101; //or
$s0, $t0, $t1
    memory[4] = 32'b10101100000100000000000000000100; //sw
$s0, 4($zero)
    memory[5] = 32'b10101100000010000000000000001000; //sw
$t0, 8($zero)
    memory[6] = 32'b00000001000010011000100000100000; //add
$s1, $t0, $t1
    memory[7] = 32'b00000001000010011001000000100010; //sub
$s2, $t0, $t1
    memory[8] = 32'b00010010001100100000000000001001; //beq
$s1, $s2, error0 nop
    memory[9] = 32'b10001100000100010000000000000100; //lw
$s1, 4($zero)
    memory[10]= 32'b00110010001100100000000001001000;
//andi $s2, $s1, 0x48 nop
    memory[11] =32'b00010010001100100000000000001001;
//beq $s1, $s2, error1
    memory[12] =32'b10001100000100110000000000001000; //lw
$s3, 8($zero)
    memory[13] =32'b00010010000100110000000000001010;
//beq $s0, $s3, error2 nop nop
    memory[14] =32'b00000010010100011010000000101010;
//slt $s4, $s2, $s1 (Last)
    memory[15] =32'b00010010100000000000000000001111;
//beq $s4, $0, EXIT
    memory[16] =32'b00000010001000001001000000100000;
//add $s2, $s1, $0
    memory[17] =32'b00001000000000000000000000001110; //j
Last
    memory[18] =32'b00100000000010000000000000000000;
//addi $t0, $0, 0(error0)
    memory[19] =32'b00100000000010010000000000000000;
//addi $t1, $0, 0
    memory[20] =32'b00001000000000000000000000011111; //j
EXIT
    memory[21] =32'b00100000000010000000000000000001;
//addi $t0, $0, 1(error1)
    memory[22] =32'b00100000000010010000000000000001;
//addi $t1, $0, 1
    memory[23] =32'b00001000000000000000000000011111; //j
EXIT
    memory[24] =32'b00100000000010000000000000000010;
//addi $t0, $0, 2(error2)
```

```
    memory[25] =32'b00100000000010010000000000000010;
//addi $t1, $0, 2
    memory[26] =32'b00001000000000000000000000011111; //j
EXIT
    memory[27] =32'b00100000000010000000000000000011;
//addi $t0, $0, 3(error3)
    memory[28] =32'b00100000000010010000000000000011;
//addi $t1, $0, 3
    memory[29] =32'b00001000000000000000000000011111; //j
EXIT
```

8. Textual Result

8.1 Textual result for the single cycle processor

The full textual result is listed in the Appendix and it satisfies expectation. We only analyze some of the result to show its accuracy.

From CLK=1 to CLK=5, the processor successfully implements the instructions "addi", "and" and "or". Values are loaded into corresponding registers correctly.

```
Time:                  10000, CLK =   1, PC = 0x00000000
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                  20000, CLK =   2, PC = 0x00000004
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                  30000, CLK =   3, PC = 0x00000008
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                  40000, CLK =   4, PC = 0x0000000c
[$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
```

```
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                 50000, CLK =   5, PC = 0x00000010
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

In CLK=9, the value of $s2 changes to 0xfffffe9, which shows that the processor sucessfully implements the instruction "sub".

```
Time:                 90000, CLK =   9, PC = 0x00000020
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xfffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

In CLK=11, the value of $s1 changes to 0x00000037, which shows that both "sw" and "lw" are implemented correctly.

```
Time:                110000, CLK =  11, PC = 0x00000028
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xfffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

In CLK=16, the value of $s4 changes to 0x00000001 while it changes to 0x00000000 in CLK=20. This shows that the instruction "slt" is successfully implemented.

```
Time:                160000, CLK =  16, PC = 0x0000003c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                200000, CLK =  20, PC = 0x0000003c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

In CLK=13, PC doesn't branch to label "error0". In CLK=19, it jumps to lable "Last". This shows that insturction "beq" and "j" are correctly implemented.

```
Time:                130000, CLK =  13, PC = 0x00000030
```

```
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               190000, CLK =  19, PC = 0x00000038
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

In CLK=21,t simulation ends when a branch to exit instruction is executated.

```
Time:               210000, CLK =  21, PC = 0x0000007c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

8.2 Textual resut for the pipeline processor

The full textual result is listed in the Appendix and it satisfies expectation. We only analyze some of the result to show its accuracy.

From CLK=0 to CLK=5, the processor implememts the first two instructions.

```
Time:                    0, CLK= 0, PC=00000000
[$s0]=00000000,[$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:                   50, CLK= 1, PC=00000004
[$s0]=00000000,[$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:                  150, CLK= 2, PC=00000008
[$s0]=00000000,[$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:                   250, CLK= 3, PC=0000000c
[$s0]=00000000,[$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:                   350, CLK= 4, PC=00000010
[$s0]=00000000,[$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:                   450, CLK= 5, PC=00000014
[$s0]=00000000,[$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
```

In CLK=6, the value of `$s0` changes to 20 which shows that the data hazard is resolved by using forwarding unit.

```
Time:                   550, CLK= 6, PC=00000018
[$s0]=00000020,[$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
```

From CLK=9 to CLK=10, there is a data hazard involving `beq` instruction. The result shows that a bubble is inserted and after the bubble, the register successfully stores the right value.

```
Time:                   850, CLK= 9, PC=00000024
[$s0]=00000037,[$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:                   950, CLK=10, PC=00000024
[$s0]=00000037,[$s1]=00000057,[$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
```

From CLK=12 to CLK=13, a bubble is inserted because of the load-use data hazard.

```
Time:                 1150, CLK=12, PC=0000002c
[$s0]=00000037,[$s1]=00000057,[$s2]=fffffe9
[$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:                 1250, CLK=13, PC=0000002c
[$s0]=00000037,[$s1]=00000057,[$s2]=fffffe9
[$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
```

From CLK=16 to CLK=18, two bubbles are inserted because of the load-branch data hazard.

```
Time:                 1550, CLK=16, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$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:                 1650, CLK=17, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$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:                 1750, CLK=18, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000000,[$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
```

The sudden change of PC value proves the correct executation of `jump` instruction.

```
Time:                  2250, CLK=23, PC=00000048
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000001,[$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:                  2350, CLK=24, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000001,[$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
```

The simulation ends when a branch to exit instruction is executated.

```
Time:                  2650, CLK=27, PC=00000040
[$s0]=00000037,[$s1]=00000037,[$s2]=00000037
[$s3]=00000020,[$s4]=00000001,[$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
================================================
```

9. Conclusion and Discussion

In this project, we built a single cycle processor and a pipeline processor, the simulation of which conforms to the ideal result. A single cycle processor is less complex due to the absence of hazard but is more time-consuming than a pipeline processor. It is up to the designer to choose which to implement.

For the implementation of the pipeline processor we had to implement it onto a FPGA board. In order to do that we needed to implement the SSD for the FPGA board and we faced a lot of problems trying to match the results to our texual results. Many of the errors were related to the clock and connecting the correct inputs and outputs, but in the end we managed to solve all the errors and matched the textual result.

Overall, we have divided the work up evenly and worked well as a team. Each of us was able to get a deeper understanding in both single cycle and pipeline processors. This project not only taught us about implementing the different processors, but also gave us an opportunity to overcome obstacles as a team and in the end we believe this project was a huge success.

References

Patterson, D. A. (2013). ***Computer organization and design*****. San Francisco: Elsevier Science & Technology.****Appendix A Source Code**

A.1 Source code for single cycle processor

```
module SingleCycle(clock,reset);
    input clock,reset;
    reg [31:0]PC;
    wire [31:0]PC_next;
    wire [31:0]PC_normal;
    wire [31:0]PC_temp;
    wire [31:0]PC_jump;
    wire [31:0]PC_branch;
    wire [31:0]instruction;
    wire [4:0]write_register;
    wire RegDst;
    wire [4:0]ins25_21;
    wire [4:0]ins20_16;
    wire [4:0]ins15_11;
    wire [31:0]write_data;
    wire [31:0]read_data1;
    wire [31:0]read_data2;
    wire RegWrite;
    wire [5:0]opcode,funct;
    wire [15:0]imm;
    wire [3:0]ALUControl;
    wire [31:0]ALU_in2;
    wire [31:0]ALU_out;
    wire Zero;
    wire ALUSrc;
    wire MemWrite;
    wire MemRead;
    wire [31:0]DataMem_ReadData;
    wire MemtoReg;
    wire [31:0]DM_out;
    wire [31:0]RA;
    wire Branch;
    wire Branch_ne;
    wire Jump;

    //PC generation
    always @(posedge reset or posedge clock) begin
        if (reset) PC=-4;
```

```
        else if (PC==-4) PC=0;
        else if (PC_next<168) PC=PC_next;
    end
    //Instruction Fetch
    InstructionMem M1 (instruction, PC, reset);
    //Instruction Decode and Register Read/Write
    assign ins25_21=instruction[25:21];
    assign ins20_16=instruction[20:16];
    assign ins15_11=instruction[15:11];
    assign write_register=(RegDst)? ins15_11:ins20_16;
    RegisterFile M2
(ins25_21,ins20_16,write_register,write_data,read_data1,r
ead_data2,RegWrite,clock,reset,regnum,regdata);
    //Instruction Decode and Execution
    assign opcode=instruction[31:26];
    assign funct=instruction[5:0];
    assign imm=instruction[15:0];
    ALU_Control M3 (ALUControl,opcode,funct);
    assign ALU_in2=(ALUSrc)? imm:read_data2;
    ALU M4 (ALU_out,Zero,read_data1,ALU_in2,ALUControl);
    //Access Memory Operand
    DataMem M5
(MemWrite,MemRead,read_data2,ALU_out,DataMem_ReadData,res
et,clock);
    assign DM_out=(MemtoReg)? DataMem_ReadData:ALU_out;
    assign write_data=DM_out;
    //PC selection
    assign PC_normal=PC+4;
    assign
PC_jump={PC_normal[31:28],instruction[25:0],2'b00};
    assign RA={{16{instruction[15]}},instruction[15:0]};
    assign PC_branch=PC_normal+(RA<<2);
    assign PC_temp=((Branch_ne&&(~Zero))||(Branch&&Zero))?
PC_branch:PC_normal;
    assign PC_next=(Jump)? PC_jump:PC_temp;
    //Control Unit
    ControlUnit M7
(opcode,RegDst,Jump,Branch,Branch_ne,MemRead,MemWrite,Mem
toReg,ALUSrc,RegWrite);
endmodule
module
ControlUnit(opcode,RegDst,Jump,Branch,Branch_ne,MemRead,
MemWrite,MemtoReg,ALUSrc,RegWrite);
    input [5:0]opcode;
```

```
    output
RegDst,Jump,Branch,Branch_ne,MemRead,MemWrite,MemtoReg,AL
USrc,RegWrite;

    assign RegDst=(opcode==6'h00);
    assign Jump=(opcode==6'h02);
    assign Branch=(opcode==6'h04);
    assign Branch_ne=(opcode==6'h05);
    assign MemRead=(opcode==6'h23);
    assign MemWrite=(opcode==6'h2b);
    assign MemtoReg=(opcode==6'h23);
    assign
ALUSrc=(opcode==6'h08||opcode==6'h0c||opcode==6'h23||opco
de==6'h2b);
    assign
RegWrite=(opcode==6'h00||opcode==6'h08||opcode==6'h0c||op
code==6'h23);
endmodule
module
DataMem(MemWrite,MemRead,WriteData,ALU_address,ReadData,
reset,clock);
    input MemWrite, MemRead;
    input [31:0] WriteData, ALU_address;
    input clock,reset;
    output [31:0] ReadData;
    reg [31:0] DataMemory [0:31];

    assign ReadData=(reset|~MemRead)?
0:DataMemory[ALU_address>>2];

    always @(negedge clock or posedge reset) begin
        if (reset) begin
            for (integer i=0; i<32; i=i+1) DataMemory[i] =
32'b0;
        end
        else if (MemWrite) DataMemory[ALU_address>>2] =
WriteData;
    end
endmodule
module
RegisterFile(read1,read2,write,w_data,data1,data2,regWrit
e,
clock,reset,regnum,regdata);
    input [4:0] read1,read2,write;
```

```
input [31:0] w_data;
input regWrite;
input clock,reset;
input [4:0]regnum;
output [31:0] data1,data2;
output [31:0]regdata;
reg[31:0] register[31:0];

assign data1=(reset)? 0:register[read1];
assign data2=(reset)? 0:register[read2];
assign regdata=(reset)? 0:register[regnum];
always @(posedge clock or posedge reset) begin
    if (reset) begin
       register[0]=32'b0;
       register[1]=32'b0;
       register[2]=32'b0;
       register[3]=32'b0;
       register[4]=32'b0;
       register[5]=32'b0;
       register[6]=32'b0;
       register[7]=32'b0;
       register[8]=32'b0;
       register[9]=32'b0;
       register[10]=32'b0;
       register[11]=32'b0;
       register[12]=32'b0;
       register[13]=32'b0;
       register[14]=32'b0;
       register[15]=32'b0;
       register[16]=32'b0;
       register[17]=32'b0;
       register[18]=32'b0;
       register[19]=32'b0;
       register[20]=32'b0;
       register[21]=32'b0;
       register[22]=32'b0;
       register[23]=32'b0;
       register[24]=32'b0;
       register[25]=32'b0;
       register[26]=32'b0;
       register[27]=32'b0;
       register[28]=32'b0;
       register[29]=32'b0;
       register[30]=32'b0;
```

```
           register[31]=32'b0;
        end
        else if (regWrite && write!=0)
register[write]=w_data;
    end
endmodule
module InstructionMem(instr, pc, reset);
    input [31:0] pc;
    input reset;
    output [31:0] instr;
    reg [31:0] memory [0:41];

    assign instr=(reset||pc==-4)? 0:memory[pc>>2];
    always @(reset) begin
        if (reset) begin
            memory[0] =
32'b00100000000010000000000000100000; //addi $t0, $zero,
0x20
            memory[1] =
32'b00100000000010010000000000110111; //addi $t1, $zero,
0x37
            memory[2] =
32'b00000001000010011000000000100100; //and $s0, $t0, $t1
            memory[3] =
32'b00000001000010011000000000100101; //or $s0, $t0, $t1
            memory[4] =
32'b10101100000100000000000000000100; //sw $s0, 4($zero)
            memory[5] =
32'b10101100000100000000000000001000; //sw $t0, 8($zero)
            memory[6] =
32'b00000001000010011000100000100000; //add $s1, $t0, $t1
            memory[7] =
32'b00000001000010011001000000100010; //sub $s2, $t0, $t1
            memory[8] =
32'b00010010001100100000000000001001; //beq $s1, $s2,
error0 nop
            memory[9] =
32'b10001100000100010000000000000100; //lw $s1, 4($zero)
            memory[10]=
32'b00110010001100100000000001001000; //andi $s2, $s1, 0x48
nop
            memory[11]
=32'b00010010001100100000000000001001; //beq $s1, $s2,
error1
```

```
            memory[12]
=32'b10001100000100110000000000001000; //lw $s3, 8($zero)
            memory[13]
=32'b00010010000100110000000000001010; //beq $s0, $s3,
error2 nop nop
            memory[14]
=32'b00000010010100011010000000101010; //slt $s4, $s2, $s1
(Last)
            memory[15]
=32'b00010010100000000000000000001111; //beq $s4, $0, EXIT
            memory[16]
=32'b00000010001000001001000000100000; //add $s2, $s1, $0
            memory[17]
=32'b00001000000000000000000000001110; //j Last
            memory[18]
=32'b00100000000010000000000000000000; //addi $t0, $0,
0(error0)
            memory[19]
=32'b00100000000010010000000000000000; //addi $t1, $0, 0
            memory[20]
=32'b00001000000000000000000000011111; //j EXIT
            memory[21]
=32'b00100000000010000000000000000001; //addi $t0, $0,
1(error1)
            memory[22]
=32'b00100000000010010000000000000001; //addi $t1, $0, 1
            memory[23]
=32'b00001000000000000000000000011111; //j EXIT
            memory[24]
=32'b00100000000010000000000000000010; //addi $t0, $0,
2(error2)
            memory[25]
=32'b00100000000010010000000000000010; //addi $t1, $0, 2
            memory[26]
=32'b00001000000000000000000000011111; //j EXIT
            memory[27]
=32'b00100000000010000000000000000011; //addi $t0, $0,
3(error3)
            memory[28]
=32'b00100000000010010000000000000011; //addi $t1, $0, 3
            memory[29]
=32'b00001000000000000000000000011111; //j EXIT
        end
    end
```

```
endmodule
module ALU(Result,Zero,A,B,ALUControl);
    input [31:0]A;
    input [31:0]B;
    input [3:0]ALUControl;
    output [31:0]Result;
    output Zero;
    reg [31:0]Result;
    wire Zero;

    always @(A,B,ALUControl) begin
        case(ALUControl)
            4'b0010: Result=A+B;
            4'b0110: Result=A-B;
            4'b0000: Result=A&B;
            4'b0001: Result=A|B;
            4'b0111: Result=A<B;
            default
Result=32'b11111111111111111111111111111111;
        endcase
    end
    assign Zero=(Result==0);
endmodule
module ALU_Control(ALUControl,opcode,funct);
    input [5:0]opcode;
    input [5:0]funct;
    output [3:0]ALUControl;
    reg [3:0]ALUControl;

    always @(funct,opcode) begin
        case(opcode)
            6'h2b: ALUControl<=4'b0010;  //sw
            6'h23: ALUControl<=4'b0010;  //lw
            6'h04: ALUControl<=4'b0110;  //beq
            6'h05: ALUControl<=4'b0110;  //bne
            6'h08: ALUControl<=4'b0010;  //addi
            6'h0c: ALUControl<=4'b0000;  //andi
            6'h00:
                case(funct)
                    6'h20: ALUControl=4'b0010;  //add
                    6'h22: ALUControl=4'b0110;  //sub
                    6'h24: ALUControl=4'b0000;  //and
                    6'h25: ALUControl=4'b0001;  //or
                    6'h2a: ALUControl=4'b0111;  //slt
```

```
                    default ALUControl=4'b0010;
                endcase
            default ALUControl=4'b0010;
        endcase
    end
endmodule
```

A.2 Source code for pipeline processor

```
module PC(PCWrite, pc_prev, pc, clk);
    input PCWrite,clk;
    input [31:0] pc_prev;
    output reg [31:0] pc;

    initial
      pc=32'b0;

    always @(posedge clk) begin
        if (PCWrite)
            pc = pc_prev;

    end
endmodule
module InstructionMem(pc, instr);
    input [31:0] pc;
    output [31:0] instr;
    reg [31:0] memory [30:0];
    reg[31:0] instr;

    initial begin
   memory[0] = 32'b00100000000010000000000000100000; //addi
$t0, $zero, 0x20
    memory[1] = 32'b00100000000010010000000000110111;
//addi $t1, $zero, 0x37
    memory[2] = 32'b00000001000010011000000000100100; //and
$s0, $t0, $t1
    memory[3] = 32'b00000001000010011000000000100101; //or
$s0, $t0, $t1
    memory[4] = 32'b10101100000100000000000000000100; //sw
$s0, 4($zero)
    memory[5] = 32'b10101100000010000000000000001000; //sw
$t0, 8($zero)
    memory[6] = 32'b00000001000010011000100000100000; //add
$s1, $t0, $t1
    memory[7] = 32'b00000001000010011001000000100010; //sub
```

```
$s2, $t0, $t1
    memory[8] = 32'b00010010001100100000000000001001; //beq
$s1, $s2, error0
    memory[9] = 32'b10001100000100010000000000000100; //lw
$s1, 4($zero)
    memory[10]= 32'b00110010001100100000000001001000;
//andi $s2, $s1, 0x48
    memory[11] =32'b00010010001100100000000000001001;
//beq $s1, $s2, error1
    memory[12] =32'b10001100000100110000000000001000; //lw
$s3, 8($zero)
    memory[13] =32'b00010010000100110000000000001010;
//beq $s0, $s3, error2
    memory[14] =32'b00000010010100011010000000101010;
//slt $s4, $s2, $s1 (Last)
    memory[15] =32'b00010010100000000000000000001111;
//beq $s4, $0, EXIT
    memory[16] =32'b00000010001000001001000000100000;
//add $s2, $s1, $0
    memory[17] =32'b00001000000000000000000000001110; //j
Last
    memory[18] =32'b00100000000010000000000000000000;
//addi $t0, $0, 0(error0)
    memory[19] =32'b00100000000010010000000000000000;
//addi $t1, $0, 0
    memory[20] =32'b00001000000000000000000000011111; //j
EXIT
    memory[21] =32'b00100000000010000000000000000001;
//addi $t0, $0, 1(error1)
    memory[22] =32'b00100000000010010000000000000001;
//addi $t1, $0, 1
    memory[23] =32'b00001000000000000000000000011111; //j
EXIT
    memory[24] =32'b00100000000010000000000000000010;
//addi $t0, $0, 2(error2)
    memory[25] =32'b00100000000010010000000000000010;
//addi $t1, $0, 2
    memory[26] =32'b00001000000000000000000000011111; //j
EXIT
    memory[27] =32'b00100000000010000000000000000011;
//addi $t0, $0, 3(error3)
    memory[28] =32'b00100000000010010000000000000011;
//addi $t1, $0, 3
    memory[29] =32'b00001000000000000000000000011111; //j
```

```
EXIT
    end

    always @(pc) begin
         if (pc>>2>29) begin
             $display("=================");
             $stop;
         end
         instr=memory[pc>>2];
    end
endmodule
module
register(read1,read2,write,w_data,data1,data2,regWrite,
clk,registersel, FPGAout);
input [4:0] read1,read2,write, registersel;
input [31:0] w_data;
input regWrite,clk;
output [31:0] data1,data2,FPGAout;
reg[31:0] data1,data2;
reg[31:0] register[31:0], FPGAout;
initial
begin
       register[0]=32'b0;
       register[1]=32'b0;
       register[2]=32'b0;
       register[3]=32'b0;
       register[4]=32'b0;
       register[5]=32'b0;
       register[6]=32'b0;
       register[7]=32'b0;
       register[8]=32'b0;
       register[9]=32'b0;
       register[10]=32'b0;
       register[11]=32'b0;
       register[12]=32'b0;
       register[13]=32'b0;
       register[14]=32'b0;
       register[15]=32'b0;
       register[16]=32'b0;
       register[17]=32'b0;
       register[18]=32'b0;
       register[19]=32'b0;
       register[20]=32'b0;
       register[21]=32'b0;
```

```
        register[22]=32'b0;
        register[23]=32'b0;
        register[24]=32'b0;
        register[25]=32'b0;
        register[26]=32'b0;
        register[27]=32'b0;
        register[28]=32'b0;
        register[29]=32'b0;
        register[30]=32'b0;
        register[31]=32'b0;
end


always @(read1,read2,write,w_data,regWrite)
begin
  data1=register[read1];
  data2=register[read2];
end
always@(*) begin
FPGAout=register[registersel];
end


always@(negedge clk)
begin
  if (regWrite==1)
  register[write]=w_data;
end


always @(posedge clk)
begin
  $display("[$s0]=%h,[$s1]=%h,[$s2]=%h",register[16],regi
ster[17],register[18]);
  $display("[$s3]=%h,[$s4]=%h,[$s5]=%h",register[19],regi
ster[20],register[21]);
  $display("[$s6]=%h,[$s7]=%h,[$t0]=%h",register[22],regi
ster[23],register[8]);
  $display("[$t1]=%h,[$t2]=%h,[$t3]=%h",register[9],regis
ter[10],register[11]);
  $display("[$t4]=%h,[$t5]=%h,[$t6]=%h",register[12],regi
ster[13],register[14]);
  $display("[$t7]=%h,[$t8]=%h,[$t9]=%h",register[15],regi
ster[24],register[25]);
```

```
end
endmodule
module sim;
  parameter half_period=50;
  reg clock;
  top UUT(clock);

  initial begin
    $display("==============================");
  end

  initial begin
    #0 clock=0;
  end

  always #half_period clock=~clock;

  initial
    #9200 $stop;
endmodule

module idwb(pc_in,instruction,branch,jadd,mem_out,alu_out,
regwrite,wb,ex,mem,data1,data2,imme,rd,rt1,rt2,rs,jump,be
q,bne,pc_out,mem_to_reg,w_addr,compare,fa,fb,aluout_mem,c
lk,registersel, FPGAout);
input [31:0] pc_in,instruction,mem_out,alu_out,aluout_mem;
input [1:0] fa,fb;
input regwrite,mem_to_reg,clk;
input [4:0] w_addr, registersel;
output [1:0] wb,mem;
output [3:0] ex;
output [31:0] data1,data2,branch,pc_out,imme,jadd, FPGAout;
output [4:0] rd,rt1,rt2,rs;
output jump,beq,bne,compare;


wire [31:0] imme_mul_4,write,d1,d2;


reg [4:0] rd,rt1,rt2,rs;
reg [31:0] pc_out,jadd;


initial begin
```

```
  pc_out=32'b0;
  jadd=32'b0;
  rd=5'b0;
  rt1=5'b0;
  rt2=5'b0;
  rs=5'b0;
end


control c(instruction[31:26],wb,mem,ex,jump,beq,bne,
instruction[5:0]);
signextension s(instruction[15:0],imme);
shift2 sh(imme,imme_mul_4);
adder a(imme_mul_4,pc_in,branch);
mux_2_1_32bit m1(write,alu_out,mem_out,mem_to_reg);
register
r(instruction[25:21],instruction[20:16],w_addr,write,d1,d
2,regwrite,clk,registersel, FPGAout);
mux_4_1_32bit m2(data1,d1,alu_out,aluout_mem,mem_out,fa);
mux_4_1_32bit m3(data2,d2,alu_out,aluout_mem,mem_out,fb);
equal eq(data1,data2,compare);




always @(pc_in,imme)
begin
  pc_out=pc_in;
  jadd={pc_in[31:28],instruction[25:0],2'b0};
  rd=instruction[15:11];
  rt1=instruction[20:16];
  rt2=instruction[20:16];
  rs=instruction[25:21];
end


endmodule


module
forward(memwb_write,memwb_rd,idex_rs,idex_rt,exmem_write,
exmem_rd,fa,fb,memwb_memread);
input memwb_write,exmem_write,memwb_memread;
```

```
input [4:0] memwb_rd,idex_rs,idex_rt,exmem_rd;
output [1:0] fa,fb;
reg [1:0] fa,fb;


initial begin
 fa=2'b00;
 fb=2'b00;
end


always @(memwb_write,memwb_rd,idex_rs,idex_rt,exmem_write,
exmem_rd,memwb_memread)begin
   fa=2'b00;
   fb=2'b00;
   if (exmem_rd!=5'b0)
     if ({exmem_write,exmem_rd}=={1'b1,idex_rs})
       fa=2'b10;
   if (fa!=2'b10)
     if (memwb_rd!=5'b0)
       if ({memwb_write,memwb_rd}=={1 'b1,idex_rs})
        if (memwb_memread==0)
         fa=2'b01;
        else
         fa=2'b11;
   if (exmem_rd!=5'b0)
       if ({exmem_write,exmem_rd}=={1'b1,idex_rt})
          fb=2'b10;
   if (fb!=2'b10)
      if (memwb_rd!=5'b0)
        if ({memwb_write,memwb_rd}=={1'b1,idex_rt})
         if (memwb_memread==0)
           fb=2'b01;
         else
           fb=2'b11;
end
endmodule


module forward_compare(memwb_write,memwb_rd,id_rt,id_rs,
exmem_write,exmem_rd,fa,fb,memwb_memread,beq,bne);
input memwb_write,exmem_write,memwb_memread,beq,bne;
input [4:0] memwb_rd,id_rt,id_rs,exmem_rd;
output [1:0] fa,fb;
```

```
reg [1:0] fa,fb;
reg branch;
initial begin
  fa=2'b00;
  fb=2'b00;
end


always
@(memwb_write,exmem_write,memwb_rd,id_rt,id_rs,exmem_rd,
memwb_memread,beq,bne)
begin
  fa=2'b00;
  fb=2'b00;
  branch=beq||bne;
  if (exmem_rd!=5'b0)
     if ({exmem_write,exmem_rd,branch}=={1'b1,id_rs,1'b1})
        fa=2'b10;
  if (fa!=2'b10)
     if (memwb_rd!=5'b0)
        if
({memwb_write,memwb_rd,branch}=={1'b1,id_rs,1'b1})
          if (memwb_memread!=1)
            fa=2'b01;
          else
            fa=2'b11;
   if (exmem_rd!=5'b0)
      if
({exmem_write,exmem_rd,branch}=={1'b1,id_rt,1'b1})
                     fb=2'b10 ;
   if (fb!=2'b10)
       if (memwb_rd!=5'b0)
           if
({memwb_write,memwb_rd,branch}=={1'b1,id_rt,1'b1})
             if (memwb_memread!=1)
                 fb=2'b01;
             else
                 fb=2'b11;
end
endmodule
module
hazard(ifidwrite,stall,memread,idexrt,ifidrt,ifidrd,
pcwrite,bne,beq,idexrd,ifidrs,idexregwrite,settle,stall_f
inish,exmemrt,exmemread,exmemregwrite);
```

```
input memread,bne,beq,idexregwrite,stall_finish,exmemread,
exmemregwrite;
input [4:0] idexrt,ifidrt,ifidrd,idexrd,ifidrs,exmemrt;
output ifidwrite,stall,pcwrite,settle;


reg ifidwrite,stall,pcwrite,stall2,settle;


initial
begin
  ifidwrite=1;
  stall=0;
  pcwrite=1;
  stall2=0;
  settle=0;
end
always
@(memread,idexrt,ifidrt,ifidrd,bne,beq,idexrd,idexregwrit
e,
ifidrs,stall_finish,exmemrt,exmemread,exmemregwrite)
begin

     stall=0;
     stall2=0;


  stall=stall || (memread &&(idexrt!=5'b0)
&&((idexrt==ifidrt)||(idexrt==ifidrs))&&idexregwrite);
  stall=stall || (exmemread && (exmemrt!=5'b0) &&
((exmemrt==ifidrt)||(exmemrt==ifidrs))&&(bne||beq)&&exmem
regwrite);
  stall=stall ||
((idexrd!=5'b0)&&((idexrd==ifidrs)||(idexrd==ifidrt))&&(b
eq||bne)&&idexregwrite);
  ifidwrite=!stall;
  pcwrite=!stall;


end
endmodule

module top(clk, PC, registersel, FPGAout);
  input clk;
```

```
  input [4:0] registersel;
  output [31:0] PC, FPGAout;
  wire
pcwrite,branch,jump,ifidwrite,flush,bubble,compare,beq,bn
e,
memread_wb,memsrc,stall_finish,settle;
  wire [31:0]
pc_in,pc_out,pc_plus_4_if,bradd,jadd,instr_if,pc_id,
instr_id,memout_wb,aluout_wb,data1_id,data2_id,imme_id,pc
_ex,aluout_ex,aluout_mem,memout_mem,memin_mem,data2_ex,da
ta1_ex,data2_ex_in,imme_ex;
  wire [1:0]
ex_mem,wb_wb,id_wb,id_mem,ex_wb,mem_wb,fa,fb,mem_mem,
fa_id,fb_id;//stage_signal
  wire [4:0]
ex_rt2,id_rd,id_rt1,id_rt2,id_rs,w_addr_wb,ex_rs,
ex_rt1,w_addr_mem,w_addr_ex,ex_rd;
  wire [3:0] id_ex,ex_ex;
  reg [5:0] cycle;


  initial begin
    cycle=6'b0;
    $monitor("Time:",$time,", CLK=%d,
PC=%h",cycle,pc_out);
  end
  always @(posedge clk)
  begin
    cycle=cycle+6'b1;
  end

  hazard
h(ifidwrite,bubble,ex_mem[1],ex_rt2,instr_id[20:16],
instr_id[15:11],pcwrite,bne,beq,ex_rd,instr_id[25:21],ex_
wb[1],settle,stall_finish,w_addr_mem,mem_mem[1],mem_wb[1]
);
  PC p(pcwrite,pc_in,pc_out,clk);
  Mux_PC m_p(branch,jump,pc_plus_4_if,bradd,jadd,pc_in);
  if_flush if_f(jump,beq,bne,compare,flush,branch,bubble,
stall_finish,settle);
  forward_compare fc(wb_wb[1],w_addr_wb,instr_id[20:16],
instr_id[25:21],mem_wb[1],w_addr_mem,fa_id,fb_id,mem_mem[
1],beq,bne);
  forward
```

```
f(wb_wb[1],w_addr_wb,ex_rs,ex_rt1,mem_wb[1],w_addr_mem,
fa,fb,memread_wb);
  lwsw
l(memsrc,w_addr_wb,w_addr_mem,memread_wb,mem_mem[0]);
  MEM_WB
mw(mem_wb[0],mem_wb[1],memout_mem,aluout_mem,w_addr_mem,
wb_wb[0],wb_wb[1],memout_wb,aluout_wb,w_addr_wb,mem_mem[1
],memread_wb,clk);
    EX_MEM
em(ex_mem[1],ex_wb[0],ex_mem[0],ex_wb[1],aluout_ex,
data2_ex,w_addr_ex,mem_mem[1],mem_wb[0],mem_mem[0],mem_wb
[1],aluout_mem,memin_mem,w_addr_mem,clk);
    ID_EX_register
ie(ex_ex,ex_mem,ex_wb,data1_ex,data2_ex_in,
imme_ex,ex_rs,ex_rt1,ex_rd,id_ex,id_mem,id_wb,data1_id,ex
_rt2,data2_id,imme_id,id_rs,id_rt1,id_rd,id_rt2,clk,bubbl
e);
    IFID ii(ifidwrite,flush,pc_plus_4_if,instr_if,pc_id,
instr_id,clk);

  assign PC = pc_out;
  IF iff(pc_out,pc_plus_4_if,instr_if);
  idwb
i(pc_id,instr_id,bradd,jadd,memout_wb,aluout_wb,wb_wb[1],
id_wb,id_ex,id_mem,data1_id,data2_id,imme_id,id_rd,id_rt1
,id_rt2,id_rs,jump,beq,bne,pc_ex,wb_wb[0],w_addr_wb,compa
re,fa_id,fb_id,aluout_mem,clk, registersel, FPGAout);
  EX
e(data1_ex,data2_ex_in,aluout_wb,aluout_mem,memout_wb,fa,
fb,
aluout_ex,imme_ex,ex_ex,data2_ex,ex_rd,ex_rt2,w_addr_ex);
  mem m(memsrc,aluout_mem,memin_mem,memout_wb,memout_mem,
mem_mem[1],mem_mem[0],clk);

endmodule

module ID_EX_register(EX_out,MEM_out,WB_out,
regdata1_out,regdata2_out,imm_out,rs_out,rt1_out,rd_out,
EX_in,MEM_in,WB_in,regdata1_in,rt2_out,
regdata2_in,imm_in,rs_in,rt1_in,rd_in,rt2_in,clk,bubble);
    input [1:0]WB_in,MEM_in;
    input [3:0]EX_in;
    input clk,bubble;
    input [31:0]regdata1_in,regdata2_in,imm_in;
```

```
input [4:0]rs_in,rt1_in,rd_in,rt2_in;
output [1:0]WB_out,MEM_out;
output [3:0]EX_out;
output [31:0]regdata1_out,regdata2_out,imm_out;
output [4:0]rs_out,rt1_out,rd_out,rt2_out;
reg [1:0]WB_out,MEM_out;
reg [3:0]EX_out;
reg [31:0]regdata1_out,regdata2_out,imm_out;
reg [4:0]rs_out,rt1_out,rd_out,rt2_out;
reg [133:0]ID_EX;

initial begin
   WB_out=2'b0;
   MEM_out=2'b0;
   EX_out=4'b0;
   regdata1_out=32'b0;
   regdata2_out=32'b0;
   imm_out=32'b0;
   rs_out=5'b0;
   rt1_out=5'b0;
   rd_out=5'b0;
   rt2_out=5'b0;
   ID_EX=134'b0;
end

always @(posedge clk)begin
    if (!bubble) begin
        ID_EX[133:132]=WB_in;
        ID_EX[130:127]=EX_in;
        ID_EX[126:125]=MEM_in;
    end else begin
        ID_EX[133:132]=2'b0;
        ID_EX[130:127]=4'b0;
        ID_EX[126:125]=2'b0;
    end
        ID_EX[122:118]=rt2_in;
        ID_EX[110:79]=regdata1_in;
        ID_EX[78:47]=regdata2_in;
        ID_EX[46:15]=imm_in;
        ID_EX[14:10]=rs_in;
        ID_EX[9:5]=rt1_in;
        ID_EX[4:0]=rd_in;
    WB_out=ID_EX[133:132];
    EX_out=ID_EX[130:127];
```

```
        MEM_out=ID_EX[126:125];
        rt2_out=ID_EX[122:118];
        regdata1_out=ID_EX[110:79];
        regdata2_out=ID_EX[78:47];
        imm_out=ID_EX[46:15];
        rs_out=ID_EX[14:10];
        rt1_out=ID_EX[9:5];
        rd_out=ID_EX[4:0];
    end
endmodule
module control(opcode,wb,m,ex,jump,beq,bne,funct);
input [5:0] opcode,funct;
output jump,beq,bne;
output [1:0] wb,m;
output [3:0] ex;

reg jump,beq,bne;
reg [1:0] wb,m;
reg [3:0] ex;

initial begin
wb=2'b00;m=2'b00;ex=4'b0100;jump=0;beq=0;bne=0;
end

always @(opcode,funct)begin
   case (opcode)
      6'b100011:begin
wb=2'b11;m=2'b10;ex=4'b0001;jump=0;beq=0;bne=0; end
      6'b101011:begin
wb=2'b00;m=2'b01;ex=4'b0001;jump=0;beq=0;bne=0; end
      6'b001000:begin
wb=2'b10;m=2'b00;ex=4'b0001;jump=0;beq=0;bne=0; end
      6'b001100:begin
wb=2'b10;m=2'b00;ex=4'b0111;jump=0;beq=0;bne=0; end
      6'b000100:begin
wb=2'b00;m=2'b00;ex=4'b0010;jump=0;beq=1;bne=0; end
      6'b000101:begin
wb=2'b00;m=2'b00;ex=4'b0010;jump=0;beq=0;bne=1; end
      6'b000010:begin
wb=2'b00;m=2'b00;ex=4'b0000;jump=1;beq=0;bne=0; end
      6'b000000:begin
wb=2'b10;m=2'b00;ex=4'b1100;jump=0;beq=0;bne=0; end
      default: begin
wb=2'b00;m=2'b00;ex=4'b0100;jump=0;beq=0;bne=0;end
```

```
   endcase
   if ({opcode,funct}==12'b0)
      begin
wb=2'b00;m=2'b00;ex=4'b0100;jump=0;beq=0;bne=0;end
end
endmodule

module ALU(Result,A,B,ALUControl);
    parameter n=32;
    input [n-1:0]A;
    input [n-1:0]B;
    input [3:0]ALUControl;
    output [n-1:0]Result;
    reg [n-1:0]Result;

    initial begin
       Result=32'b0;
    end

    always @(A,B,ALUControl) begin
        case(ALUControl)
            4'b0010: Result=A+B;
            4'b0110: Result=A+(~B)+32'b1;
            4'b0000: Result=A&B;
            4'b0001: Result=A|B;
            4'b0111: Result=(A+(~B)+32'b1)>>31;
            default Result=32'b0;
        endcase
    end

endmodule


module ALU_Control(ALUControl,ALUop,funct);
    input [1:0]ALUop;
    input [5:0]funct;
    output [3:0]ALUControl;
    reg [3:0]ALUControl;

    initial begin
      ALUControl=4'b0;
    end

    always @(funct,ALUop) begin
```

```
        case(ALUop)
            2'b00: ALUControl=4'b0010;
            2'b01: ALUControl=4'b0110;
            2'b11: ALUControl=4'b0000;  //andi
            2'b10:
                case(funct)
                    6'b100000: ALUControl=4'b0010;  //add
                    6'b100010: ALUControl=4'b0110;  //sub
                    6'b100100: ALUControl=4'b0000;  //and
                    6'b100101: ALUControl=4'b0001;  //or
                    6'b101010: ALUControl=4'b0111;  //slt
                    default: ALUControl=4'b1111;
                endcase
        endcase
    end
endmodule

module
EX(data1,data2,aluout_wb,aluout_mem,memout_wb,fa,fb,aluou
t,
imme,ex,data2_out,rd_in,rt_in,rd);
input [31:0] data1,data2,aluout_wb,aluout_mem,memout_wb;
input [1:0] fa,fb;
input [31:0] imme;
input [4:0] rd_in,rt_in;
input [3:0] ex;
output [31:0] aluout,data2_out;
output [4:0] rd;

wire [31:0] alu1,alu2;
wire [3:0] aluc;


mux_2_1_5bit m5(rd,rt_in,rd_in,ex[3]);
ALU_Control a2(aluc,ex[2:1],imme[5:0]);
mux_4_1_32bit
m6(alu1,data1,aluout_wb,aluout_mem,memout_wb,fa);
mux_4_1_32bit
m7(data2_out,data2,aluout_wb,aluout_mem,memout_wb,fb);
mux_2_1_32bit m8(alu2,data2_out,imme,ex[0]);
ALU a3(aluout,alu1,alu2,aluc);

endmodule
module IF(pc_in,pc_out,instruction_out);
```

```
input [31:0] pc_in;
output [31:0] pc_out,instruction_out;
reg [31:0] pc_out;
initial
  pc_out=32'b0;

InstructionMem im(pc_in,instruction_out);
always @(pc_in)
   pc_out=pc_in+32'b100;

endmodule
module
mem(memsrc,addr,regout,memout_wb,memout_mem,memread,
memwrite, clk);
input memsrc,memread,memwrite, clk;
input [31:0] regout,memout_wb,addr;
output [31:0] memout_mem;


wire [31:0] write_data;



mux_2_1_32bit m9(write_data,regout,memout_wb,memsrc);
DataMem dm(memwrite,memread,write_data,addr,memout_mem,
clk);
endmodule
module shift2(in,out);
input [31:0] in;
output [31:0] out;

reg [31:0] out;

initial
   out=32'b0;
always @(in)
  out=in*4;
endmodule

module signextension(imme,imme_out);
input [15:0] imme;
output [31:0] imme_out;
```

```
reg [31:0] imme_out;
initial begin
  imme_out=32'b0;
end
always @(imme)
  imme_out={{16{imme[15]}},imme[15:0]};
endmodule

module DataMem(MemWrite, MemRead, WriteData, ALU_address,
ReadData, clk);
    input MemWrite, MemRead, clk;
    input [31:0] WriteData, ALU_address;
    output [31:0] ReadData;
    reg [31:0] DataMemory [31:0];
    wire [31:0] i;
    assign i = ALU_address >> 2;
    initial begin
       DataMemory[0]=32'b0;
       DataMemory[1]=32'b0;
       DataMemory[2]=32'b0;
       DataMemory[3]=32'b0;
       DataMemory[4]=32'b0;
       DataMemory[5]=32'b0;
       DataMemory[6]=32'b0;
       DataMemory[7]=32'b0;
       DataMemory[8]=32'b0;
       DataMemory[9]=32'b0;
       DataMemory[10]=32'b0;
       DataMemory[11]=32'b0;
       DataMemory[12]=32'b0;
       DataMemory[13]=32'b0;
       DataMemory[14]=32'b0;
       DataMemory[15]=32'b0;
       DataMemory[16]=32'b0;
       DataMemory[17]=32'b0;
       DataMemory[18]=32'b0;
       DataMemory[19]=32'b0;
       DataMemory[20]=32'b0;
       DataMemory[21]=32'b0;
       DataMemory[22]=32'b0;
       DataMemory[23]=32'b0;
       DataMemory[24]=32'b0;
       DataMemory[25]=32'b0;
       DataMemory[26]=32'b0;
```

```
        DataMemory[27]=32'b0;
        DataMemory[28]=32'b0;
        DataMemory[29]=32'b0;
        DataMemory[30]=32'b0;
        DataMemory[31]=32'b0;
    end


    always@(posedge clk) begin
        if (MemWrite==1'b1) DataMemory[i] = WriteData;
        else DataMemory[i] = DataMemory[i];
    end
    assign ReadData = (MemRead==1'b1)? DataMemory[i]:32'b0;
endmodule
module Mux_PC(Branch, Jump, pc_next, BrAdd, JAdd, out);
    input Branch, Jump;
    input [31:0] pc_next, BrAdd, JAdd;
    output reg [31:0] out;


    always @(Branch, Jump, pc_next) begin
        if (Branch)
            out <= BrAdd;
        else if (Jump)
            out <= JAdd;
        else
            out <= pc_next;
    end
endmodule

module IFID(write, flush, PC, instruction, PC_out,
instruction_out,clk);
  input write, flush,clk;
  input [31:0] PC,instruction;
  output [31:0] PC_out,instruction_out;
  reg [31:0] PC_out,instruction_out;

  initial
  begin
    PC_out=32'b0;
    instruction_out=32'b0;
  end

  always @(posedge clk) begin
```

```
    if ({flush,write}==2'b11)
       instruction_out=32'b0;
    else if ({flush,write}==2'b01)
       instruction_out=instruction;
    if (write==1)
       PC_out=PC;
  end
endmodule
module EX_MEM(MemReadEX, MemtoRegEX, MemWriteEX, RegWriteEX,
ALUResultEX, MuxForwardEX, RegDstEX, MemReadMEM,
MemtoRegMEM,
MemWriteMEM, RegWriteMEM, ALUResultMEM, MuxForwardMEM,
RegDstMEM,clk);
    input MemReadEX, MemtoRegEX, MemWriteEX,
RegWriteEX,clk;
    input [31:0] ALUResultEX, MuxForwardEX;
    input [4:0] RegDstEX;
    output reg MemReadMEM, MemtoRegMEM, MemWriteMEM,
RegWriteMEM;
    output reg [31:0] ALUResultMEM, MuxForwardMEM;
    output reg [4:0]RegDstMEM;

    initial begin
        MemReadMEM<=1'b0;
        MemtoRegMEM<=1'b0;
        MemWriteMEM<=1'b0;
        RegWriteMEM<=1'b0;
    end

    always @(posedge clk) begin
        MemReadMEM<=MemReadEX;
        MemtoRegMEM<=MemtoRegEX;
        MemWriteMEM<=MemWriteEX;
        RegWriteMEM<=RegWriteEX;
        ALUResultMEM<=ALUResultEX;
        MuxForwardMEM<=MuxForwardEX;
        RegDstMEM<=RegDstEX;
    end
endmodule
module MEM_WB(MemtoRegMEM, RegWriteMEM, ReadDataMEM,
ALUResultMEM, RegDstMEM, MemtoRegWB, RegWriteWB, ReadDataWB,
ALUResultWB, RegDstWB,MemReadMem,MemReadWB,clk);
    input MemtoRegMEM, RegWriteMEM,MemReadMem,clk;
    input [31:0] ReadDataMEM, ALUResultMEM;
```

```
    input [4:0] RegDstMEM;
    output reg MemtoRegWB, RegWriteWB,MemReadWB;
    output reg [31:0] ReadDataWB, ALUResultWB;
    output reg [4:0] RegDstWB;

    initial begin
    MemtoRegWB<=1'b0;
    RegWriteWB<=1'b0;
    end

    always @(posedge clk) begin
        MemtoRegWB<=MemtoRegMEM;
        RegWriteWB<=RegWriteMEM;
        ReadDataWB<=ReadDataMEM;
        ALUResultWB<=ALUResultMEM;
        RegDstWB<=RegDstMEM;
        MemReadWB<=MemReadMem;
    end
endmodule
module mux_2_1_32bit(out,i0,i1,sel);
  input sel;
  input [31:0] i0,i1;
  output [31:0] out;
  reg [31:0] out;

  always @(i0,i1,sel)
  begin
    case (sel)
      1'b0:out=i0;
      1'b1:out=i1;
      default out=0;
    endcase
 end
endmodule

module mux_4_1_32bit(out,i0,i1,i2,i3,sel);
  input [31:0] i0,i1,i2,i3;
  input [1:0] sel;
  output [31:0] out;
  reg [31:0] out;

  always @(i0,i1,i2,sel,i3)
  begin
    case (sel)
```

```
      2'b00:out=i0;
      2'b01:out=i1;
      2'b10:out=i2;
      2'b11:out=i3;
      default:out=0;
    endcase
  end

endmodule

module mux_2_1_5bit(out,i0,i1,sel);
  input sel;
  input [4:0] i0,i1;
  output [4:0] out;
  reg [4:0] out;

  always @(i0,i1,sel)
  begin
    case (sel)
      1'b0:out=i0;
      1'b1:out=i1;
      default out=0;
    endcase
 end
endmodule
module if_flush(jump,beq,bne,equal,flush,branch,bubble);
input jump,beq,bne,equal,bubble;
output flush,branch;

reg flush,branch;

initial
   begin
      flush=0;
      branch=0;
   end

always @(jump,beq,bne,equal,bubble)
begin
  case ({jump,beq,bne,equal,bubble})
    5'b10000:begin flush=1;branch=0;end
    5'b10010:begin flush=1;branch=0;end
    5'b01010:begin flush=1;branch=1;end
    5'b00100:begin flush=1;branch=1;end
```

```
    default:begin flush=0;branch=0;end
  endcase
end
endmodule

module lwsw(memsrc,wb_rt,mem_rt,wb_memread,mem_memwrite);
input wb_memread,mem_memwrite;
input [4:0] wb_rt,mem_rt;
output memsrc;

reg memsrc;

initial
   memsrc=0;

always @(wb_rt,mem_rt,wb_memread,mem_memwrite)
   if
({wb_rt,wb_memread,mem_memwrite}=={mem_rt,1'b1,1'b1})
      memsrc=1;
endmodule

module equal(a,b,out);
input [31:0] a,b;
output out;

reg out;
always @(a,b)
  out=a==b;
endmodule
```

Appendix B Full Textual Result

B.1 Full textual result for single cycle processor

```
================================================================
=====
Time:                 10000, CLK =   1, PC = 0x00000000
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

```
Time:                 20000, CLK =   2, PC = 0x00000004
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                 30000, CLK =   3, PC = 0x00000008
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                 40000, CLK =   4, PC = 0x0000000c
[$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                 50000, CLK =   5, PC = 0x00000010
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                 60000, CLK =   6, PC = 0x00000014
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                 70000, CLK =   7, PC = 0x00000018
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                 80000, CLK =   8, PC = 0x0000001c
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
```

```
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                90000, CLK =   9, PC = 0x00000020
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               100000, CLK =  10, PC = 0x00000024
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               110000, CLK =  11, PC = 0x00000028
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               120000, CLK =  12, PC = 0x0000002c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               130000, CLK =  13, PC = 0x00000030
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               140000, CLK =  14, PC = 0x00000034
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
```

```
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               150000, CLK =  15, PC = 0x00000038
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               160000, CLK =  16, PC = 0x0000003c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               170000, CLK =  17, PC = 0x00000040
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               180000, CLK =  18, PC = 0x00000044
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               190000, CLK =  19, PC = 0x00000038
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:               200000, CLK =  20, PC = 0x0000003c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
```

```
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
Time:                210000, CLK =  21, PC = 0x0000007c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
=============================================================
=====
```

B.2 Full textual result for pipeline processor

```
  ================================================
  Time:                      0, CLK= 0, PC=00000000
  [$s0]=00000000,[$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:                     50, CLK= 1, PC=00000004
  [$s0]=00000000,[$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:                    150, CLK= 2, PC=00000008
  [$s0]=00000000,[$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:                    250, CLK= 3, PC=0000000c
  [$s0]=00000000,[$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:                    350, CLK= 4, PC=00000010
  [$s0]=00000000,[$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:                  450, CLK= 5, PC=00000014
[$s0]=00000000,[$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:                  550, CLK= 6, PC=00000018
[$s0]=00000020,[$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:                  650, CLK= 7, PC=0000001c
[$s0]=00000037,[$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:                  750, CLK= 8, PC=00000020
[$s0]=00000037,[$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:                  850, CLK= 9, PC=00000024
[$s0]=00000037,[$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:                  950, CLK=10, PC=00000024
[$s0]=00000037,[$s1]=00000057,[$s2]=00000000
[$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:                 1050, CLK=11, PC=00000028
[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9
[$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:                 1150, CLK=12, PC=0000002c
[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9
[$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:                 1250, CLK=13, PC=0000002c
[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9
[$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:                 1350, CLK=14, PC=00000030
[$s0]=00000037,[$s1]=00000037,[$s2]=ffffffe9
[$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:                 1450, CLK=15, PC=00000034
[$s0]=00000037,[$s1]=00000037,[$s2]=ffffffe9
[$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:                 1550, CLK=16, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$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:                 1650, CLK=17, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$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:                 1750, CLK=18, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000000,[$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:                 1850, CLK=19, PC=0000003c
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000000,[$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:                 1950, CLK=20, PC=00000040
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000000,[$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:                 2050, CLK=21, PC=00000040
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000000,[$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:                 2150, CLK=22, PC=00000044
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$s4]=00000001,[$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:                 2250, CLK=23, PC=00000048
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
```

```
[$s3]=00000020,[$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,[$t9]=00000000
Time:                2350, CLK=24, PC=00000038
[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
[$s3]=00000020,[$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,[$t9]=00000000
Time:                2450, CLK=25, PC=0000003c
[$s0]=00000037,[$s1]=00000037,[$s2]=00000037
[$s3]=00000020,[$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,[$t9]=00000000
Time:                2550, CLK=26, PC=00000040
[$s0]=00000037,[$s1]=00000037,[$s2]=00000037
[$s3]=00000020,[$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,[$t9]=00000000
Time:                2650, CLK=27, PC=00000040
[$s0]=00000037,[$s1]=00000037,[$s2]=00000037
[$s3]=00000020,[$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
================================================
```