

ECEN 651

LAB NO. 7

Subroutine Calls and Static Branch Prediction for MIPS

TA: Mr. Ye Wang

DATE: 11/27/2019

Student Name: Dhiraj Dinesh Kudva

UIN: 829009538

**Objective:** The objectives of this lab is to implement jr and jal instruction in the processor created in the earlier lab. This lab also focusses on implementing a static branch predictor.

### Design:

#### Part A: Processor with Jr and Jal instructions

The below figure is the MIPS processor. In this case, we have considered a 5-stage pipelined MIPS architecture.

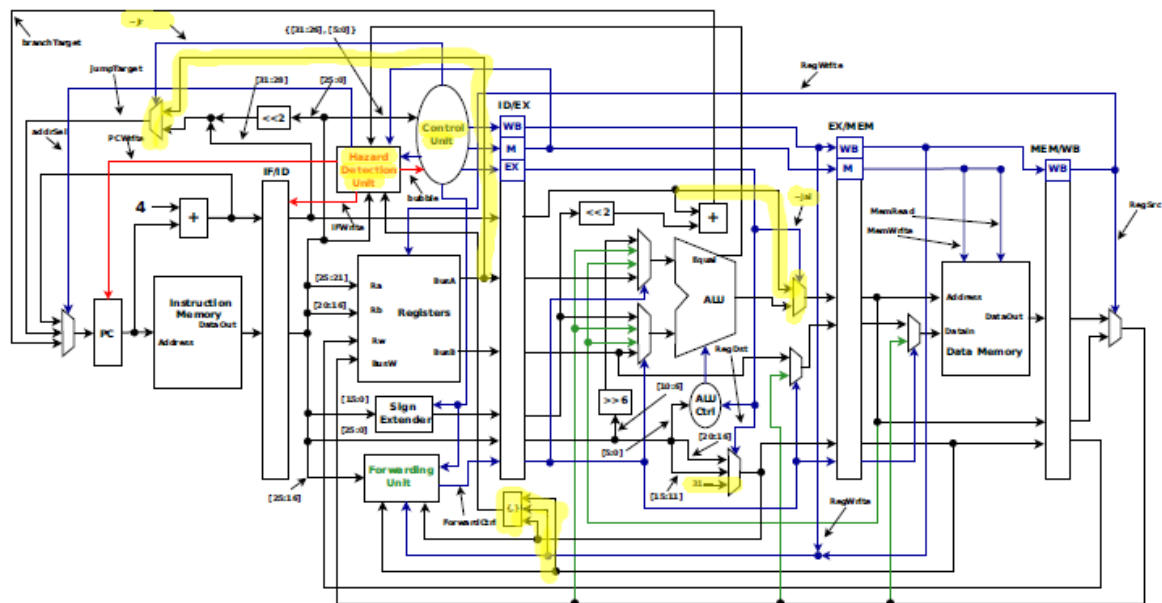


Figure 1: MIPS Block Diagram with Jal and Jr Support

As compared to the previous lab, the main changes that will take place for this lab is in the hazard unit, control unit and a few additional multiplexers.

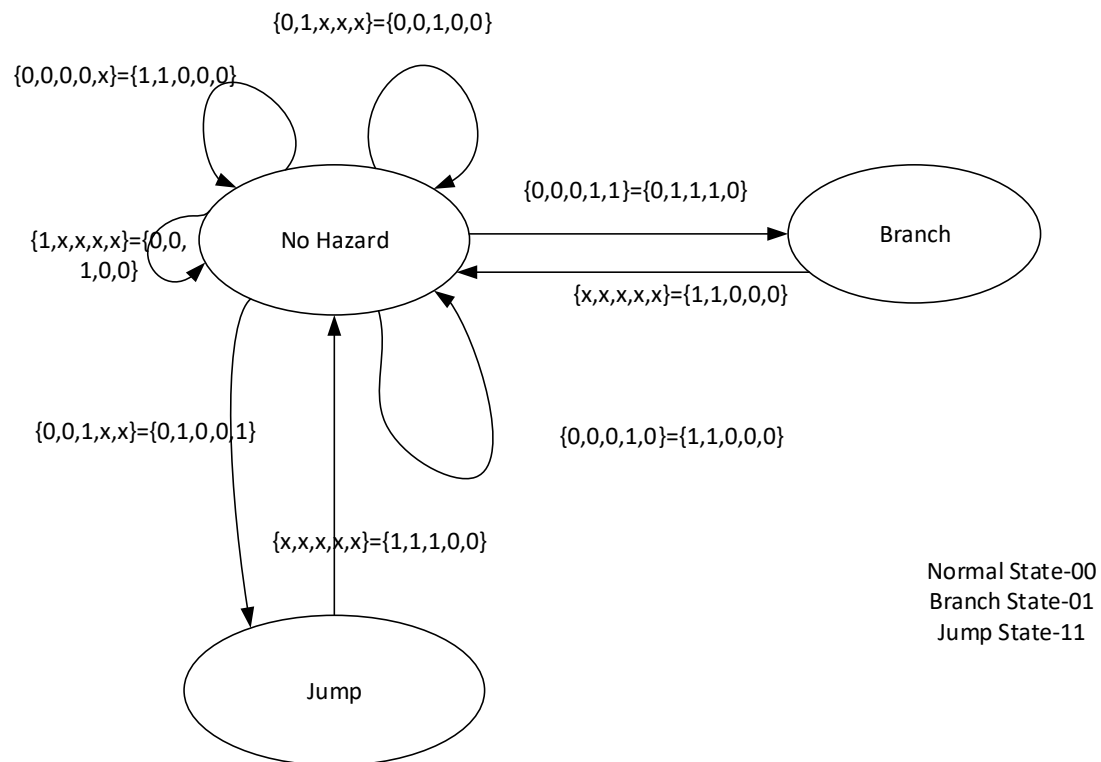
**Hazard Unit:** jal and jr instructions are unconditional jump. When these two instructions are decoded, jump is set high by the control unit. Hazard arises in jr instruction if the register which holds the target address is written by the previous instruction which is not yet completed.

**Control unit:** jal and jr instructions to be added in the control unit so that corresponding control signals are generated

Opcode for jal: 000011

Jr is a r-type function with function code as 001000

## FINITE STATE MACHINE (HAZARD UNIT)



Modified code for jal and jr instruction as follows:

### 1. ALU

```

`timescale 1ns / 1ps
// define the ALU operations (ALUOp)
`define AND 4'b0000
`define OR 4'b0001
`define ADD 4'b0010
`define SLL 4'b0011
`define SRL 4'b0100
`define SUB 4'b0110
`define SLT 4'b0111
`define ADDU 4'b1000
`define SUBU 4'b1001
`define XOR 4'b1010
`define SLTU 4'b1011
`define NOR 4'b1100
`define SRA 4'b1101
`define LUI 4'b1110
module ALU(BusW, Zero, BusA, BusB, ALUCtrl);
// Define Inputs and Outputs
input signed [31:0] BusA;
input signed [31:0] BusB;
input [3:0] ALUCtrl;
output reg [31:0] BusW;
output reg Zero;
always @(*) begin
case (ALUCtrl)

```

```

// Define ALU operations based on the ALUCtrl
`AND : BusW <= BusA & BusB;
`OR : BusW <= BusA | BusB;
`ADD : BusW <= BusA + BusB;
`SLL : BusW <= BusB << BusA;
`SRL : BusW <= BusB >> BusA;
`SUB : BusW <= BusA - BusB;
`SLT : BusW <= (BusA < BusB)? 1 : 0;
`ADDU : BusW <= BusA + BusB;
`SUBU : BusW <= BusA - BusB;
`XOR : BusW <= BusA ^ BusB;
`NOR : BusW <= ~(BusA | BusB);
`SRA : BusW <= BusB >>> BusA[4:0];
`LUI : BusW <= {BusB[15:0],16'b0};
`SLTU : begin
if(BusA[31] == 0 && BusB[31] == 0)
BusW <= (BusA < BusB)? 1 : 0;
else if(BusA[31] == 1 && BusB[31] == 1)
BusW <= ((~BusA) + 1 < (~BusB) + 1)? 1 : 0;
else if(BusA[31] == 1 && BusB[31] == 0)
BusW <= ((~BusA) + 1 < BusB)? 1 : 0;
else if (BusA[31] == 0 && BusB[31] == 1)
BusW <= (BusA < (~BusB) + 1)? 1 : 0;
else
BusW <= 0;
end
default: BusW <= 0;
endcase
// Output bit zero if BusW is 0
Zero <= (BusA == BusB) ? 1'b1 : 1'b0;
end
endmodule

```

## 2. ALU Control

```

`timescale 1ns / 1ps
// Define R-type instructions
`define SLLFunc 6'b000000
`define SRLFunc 6'b000010
`define SRAFunc 6'b000011
`define ADDFunc 6'b100000
`define ADDUFunc 6'b100001
`define SUBFunc 6'b100010
`define SUBUFunc 6'b100011
`define ANDFunc 6'b100100
`define ORFunc 6'b100101
`define XORFunc 6'b100110
`define NORFunc 6'b100111
`define SLTFunc 6'b101010
`define SLTUFunc 6'b101011
`define JRFunc 6'b001000
// Define ALU operation codes
`define AND 4'b0000
`define OR 4'b0001
`define ADD 4'b0010
`define SLL 4'b0011

```

```

`define SRL 4'b0100
`define SUB 4'b0110
`define SLT 4'b0111
`define ADDU 4'b1000
`define SUBU 4'b1001
`define XOR 4'b1010
`define SLTU 4'b1011
`define NOR 4'b1100
`define SRA 4'b1101
`define LUI 4'b1110
module ALUControl(ALUCtrl, ALUOp, FuncCode);
// Define the inputs and the outputs
input [3:0] ALUOp;
input [5:0] FuncCode;
output reg [3:0] ALUCtrl;
always @(*) begin
// if ALUOp != 1111
if (ALUOp != 4'b1111)
ALUCtrl = ALUOp;
else
case (FuncCode)
`JRFunc: ALUCtrl = `ADD;
`SLLFunc: ALUCtrl = `SLL;
`SRLFunc: ALUCtrl = `SRL;
`SRAFunc: ALUCtrl = `SRA;
`ADDFunc: ALUCtrl = `ADD;
`ADDUFunc: ALUCtrl = `ADDU;
`SUBFunc: ALUCtrl = `SUB;
`SUBUFunc: ALUCtrl = `SUBU;
`ANDFunc: ALUCtrl = `AND;
`ORFunc: ALUCtrl = `OR;
`XORFunc: ALUCtrl = `XOR;
`NORFunc: ALUCtrl = `NOR;
`SLTFunc: ALUCtrl = `SLT;
`SLTUFunc: ALUCtrl = `SLTU;
default: ALUCtrl = 0;
endcase
end
endmodule

```

### 3. Data Memory

```

`timescale 1ns / 1ps
module DataMemory( ReadData, Address, WriteData, MemoryRead,
MemoryWrite,
Clock);
// Define Inputs and Outputs
input MemoryRead, MemoryWrite, Clock;
input [5:0] Address;
input [31:0] WriteData;
output reg [31:0] ReadData;
// Define DataMemory
reg [31:0] DataMemory[63:0];
// Read at posedge
always @(posedge Clock) begin
if (MemoryRead == 1)

```

```

ReadData <= DataMemory[Address];
end
// Write at negedge
always @(negedge Clock) begin
if (MemoryWrite == 1)
DataMemory[Address] <= WriteData;
end
endmodule

```

#### 4. Forwarding Unit

```

`timescale 1ns / 1ps
module ForwardingUnit(UseShamt , UseImmed , ID_Rs , ID_Rt , EX_Rw ,
MEM_Rw,
EX_RegWrite , MEM_RegWrite , ALUOpCtrlA , ALUOpCtrlB ,
DataMemForwardCtrl_EX
,
DataMemForwardCtrl_MEM);
// Define inputs and outputs
input UseShamt , UseImmed ;
input [4:0] ID_Rs , ID_Rt , EX_Rw , MEM_Rw;
input EX_RegWrite , MEM_RegWrite ;
output reg [1:0] ALUOpCtrlA ;
output reg [1:0] ALUOpCtrlB ;
output reg DataMemForwardCtrl_EX , DataMemForwardCtrl_MEM ;
always @(*) begin
// Control Signal definitions:
if(UseShamt == 1)
ALUOpCtrlA <= 2'b00;
else if (ID_Rs == EX_Rw & EX_RegWrite == 1 & EX_Rw != 0)
ALUOpCtrlA <= 2'b10;
else if(ID_Rs == MEM_Rw & MEM_RegWrite == 1 & MEM_Rw != 0)
ALUOpCtrlA <= 2'b01;
else
ALUOpCtrlA <= 2'b11;
// If I-type
if(UseImmed == 1)
ALUOpCtrlB <= 2'b00;
else if (ID_Rt == EX_Rw & EX_RegWrite == 1 & EX_Rw != 0)
ALUOpCtrlB <= 2'b10;
else if(ID_Rt == MEM_Rw & MEM_RegWrite == 1 & MEM_Rw != 0)
ALUOpCtrlB <= 2'b01;
else
ALUOpCtrlB <= 2'b11;
// Check data dependancy, case 1
if (EX_RegWrite == 1 & ID_Rt == EX_Rw & EX_Rw != 0) begin
DataMemForwardCtrl_EX <= 1'b0;
DataMemForwardCtrl_MEM <= 1'b1;
end
// Check data dependancy, case 2
else if(MEM_RegWrite == 1 & ID_Rt == MEM_Rw & MEM_Rw != 0) begin
DataMemForwardCtrl_EX <= 1'b1;
DataMemForwardCtrl_MEM <= 1'b0;
end
else begin
// No data dependancy
DataMemForwardCtrl_EX <= 1'b0;

```

```

DataMemForwardCtrl_MEM <= 1'b0;
end
end
endmodule

```

## 5. HAZARD Unit

```

`timescale 1ns / 1ps
module HazardUnit(IF_write, PC_write, bubble, addrSel, Jump, Jr, Branch,
ALUZero,
memReadEX, ID_Rs, ID_Rt, EX_Rw, MEM_Rw, EX_RegWrite, MEM_RegWrite,
UseShamt,
UseImmed, Clk, Rst);
// Input and output ports
output reg IF_write, PC_write, bubble;
output reg [1:0] addrSel;
input Jump , Branch , ALUZero , memReadEX , Clk , Rst, Jr,
EX_RegWrite, MEM_RegWrite ;
input [4:0] EX_Rw, MEM_Rw;
input UseShamt , UseImmed ;
input [4:0] ID_Rs, ID_Rt;
wire LdHazard, JrHazard;
reg [1:0] state, Next_State;
// Load hazard, For R-type => if any register equal to load register
// For shift or I-type => if the register equal to the load inf
// Logic for detecting load hazard previous state
/*assign LdHazard =
(memReadEX == 1) & (prevRt != 0) & (((ID_Rs == prevRt || ID_Rt ==
prevRt) &
UseShamt == 0 & UseImmed == 0) ||
(UseShamt == 1 & ID_Rs == prevRt) || (UseImmed == 1 & ID_Rs == prevRt
)) ? 1 : 0 ;
*/
assign LdHazard = (((ID_Rs == EX_Rw && UseShamt != 1) || (ID_Rt == EX_Rw
&&
UseImmed != 1)) && EX_Rw != 0 && memReadEX == 1) ? 1 : 0;
// Jr hazard
assign JrHazard = ((Jr == 1) && ((EX_Rw == ID_Rs && EX_RegWrite ==
1) || (MEM_Rw == ID_Rs && MEM_RegWrite == 1))) ? 1 : 0;
always @(*) begin
// Check for hazard => Load, Branch, and Jump
case(state)
2'b00: begin
if ((LdHazard == 1) || (JrHazard == 1)) begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b001;
addrSel <= 2'b00;
end
else if (Branch == 1) begin
Next_State <= 2'b01;
{IF_write, PC_write, bubble} <= 3'b000;
addrSel <= 2'b00;
end
else if (Jump == 1) begin
Next_State <= 2'b11;
{IF_write, PC_write, bubble} <= 3'b010;
addrSel <= 2'b01;

```

```

end
// NO Hazard
else begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b110;
addrSel <= 2'b00;
end
end
// Branch 0
2'b01: begin
if (ALUZero == 1) begin
// If branch is taken
Next_State <= 2'b10;
{IF_write, PC_write, bubble} <= 3'b011;
addrSel <= 2'b10;
end
else begin
// If branch NOT taken
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b111;
addrSel <= 2'b00;
end
end
// Branch 1
2'b10: begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b111;
addrSel <= 2'b00;
end
// Jump
2'b11: begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b111;
addrSel <= 2'b00;
end
default: begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b110;
addrSel <= 2'b00;
end
endcase
end
// State is changed in negedge of the clock
always @(negedge Clk) begin
// If reset
if(Rst == 0)
state <= 2'b00;
else
state <= Next_State;
end
endmodule

```



## 6. Pipeline Control Unit

```
`timescale 1ns / 1ps
// Define Opcodes
`define RTYPEOPCODE 6'b000000
`define LWOPCODE 6'b100011
`define SWOPCODE 6'b101011
`define BEQOPCODE 6'b000100
`define JOPCODE 6'b000010
`define ORIOPCODE 6'b001101
`define ADDIOPCODE 6'b001000
`define ADDIUOPCODE 6'b001001
`define ANDIOPCODE 6'b001100
`define LUIOPCODE 6'b001111
`define SLTIOPCODE 6'b001010
`define SLTIUOPCODE 6'b001011
`define XORIOPCODE 6'b001110
`define JALOPCODE 6'b000011
// Define Function codes (used for R-type instructions)
`define SLLFunc 6'b000000
`define SRLFunc 6'b000010
`define SRAFunc 6'b000011
`define ADDFunc 6'b100000
`define ADDUFunc 6'b100001
`define SUBFunc 6'b100010
`define SUBUFunc 6'b100011
`define ANDFunc 6'b100100
`define ORFunc 6'b100101
`define XORFunc 6'b100110
`define NORFunc 6'b100111
`define SLTFunc 6'b101010
`define SLTUFunc 6'b101011
`define JRFunc 6'b001000
// Define ALU operation codes
`define AND 4'b0000
`define OR 4'b0001
`define ADD 4'b0010
`define SLL 4'b0011
`define SRL 4'b0100
`define SUB 4'b0110
`define SLT 4'b0111
`define ADDU 4'b1000
`define SUBU 4'b1001
`define XOR 4'b1010
`define SLTU 4'b1011
`define NOR 4'b1100
`define SRA 4'b1101
`define LUI 4'b1110
module PipelinedControl(Jump, Branch, RegDst, UseImmed, UseShamt, Jal,
RegSrc,
RegWrite, MemRead, MemWrite, SignExtend, ALUOp, JumpSel, OpCode,
FuncCode);
// Define inputs and outputs
input [5:0] OpCode;
input [5:0] FuncCode;
output reg UseShamt, UseImmed;
output reg [3:0] ALUOp;
```

```

output reg RegDst, RegSrc, RegWrite, MemRead, MemWrite, Branch,
Jump, JumpSel, Jal, SignExtend;
always @(*) begin
case(OpCode)
`RTYPEOPCODE: begin
// If R-Type instruction
ALUOp = 4'b1111;
case(FuncCode)
// Shift Instruction
`SLLFunc: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} = 10'b110010000x;
{JumpSel, Jal} = 2'b00;
end
// Shift Instruction
`SRLFunc: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} = 10'b110010000x;
{JumpSel, Jal} = 2'b00;
end
// Shift Instruction
`SRAFunc: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} = 10'b110010000x;
{JumpSel, Jal} = 2'b00;
end
`JRFunc: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} = 10'b0001000010;
{JumpSel, Jal} = 2'b10;
end
//R type instruction that are not shift
default: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b100010000x;
{JumpSel, Jal} = 2'b00;
end
endcase
end
// Load opcode
`LWOPCODE: begin
ALUOp = `ADD;
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0011110001;
{JumpSel, Jal} = 2'b00;
end
// Store opcode
`SWOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'bx01x001001;
ALUOp = `ADD;
{JumpSel, Jal} = 2'b00;
end
// Branch if equal opcode
`BEQOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'bx00x000101;

```

```

ALUOp = `SUB;
{JumpSel, Jal} = 2'b00;
end
// Jump opcode
`JOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'bx00x00001x;
ALUOp = 4'bxxxx;
{JumpSel, Jal} = 2'b00;
end
// Or immediate opcode
`ORIOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100000;
ALUOp = `OR;
{JumpSel, Jal} = 2'b00;
end
// Add immediate opcode
`ADDIOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100001;
ALUOp = `ADD;
{JumpSel, Jal} = 2'b00;
end
// Add immediate unsigned opcode
`ADDIUOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100000;
ALUOp = `ADDU;
{JumpSel, Jal} = 2'b00;
end
// AND bitwise with immediate opcode
`ANDIOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100000;
ALUOp = `AND;
{JumpSel, Jal} = 2'b00;
end
// Load Upper Immediate opcode
`LUIOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100000;
ALUOp = `LUI;
{JumpSel, Jal} = 2'b00;
end
// Set less than Immediate opcode
`SLTIOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100001;
ALUOp = `SLT;
{JumpSel, Jal} = 2'b00;
end
// Set less than Immediate unsigned opcode
`SLTIUOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100000;
ALUOp = `SLTU;

```

```

{JumpSel, Jal} = 2'b00;
end
// XOR bitwise Immediate opcode
`XORIOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0010100000;
ALUOp = `XOR;
{JumpSel, Jal} = 2'b00;
end
// Jump and Link
`JALOPCODE: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0000100010;
ALUOp = 4'bxxxx;
{JumpSel, Jal} = 2'b01;
end
default: begin
{RegDst, UseShamt, UseImmed, RegSrc, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} = 10'b0000000000;
ALUOp = 4'b0000;
{JumpSel, Jal} = 2'b00;
end
endcase
end
endmodule

```

## 7. Pipeline Process Unit

```

`timescale 1ns / 1ps
module PipelinedProc(CLK, Reset_L, startPC, dMemOut);
// Define the inputs and outputs
input CLK, Reset_L;
input [31:0] startPC;
output [31:0] dMemOut;
reg [31:0] PC;
// Define the wires for PC calculations
reg [31:0] PCnext;
wire [31:0] IF_Instruction;
// Define the wires for control signals
wire SignExtend;
// Define wires for ALU usage
reg [31:0] ALUInputA;
reg [31:0] ALUInputB;
wire [3:0] ALUCtrl;
wire [31:0] BusW;
wire [31:0] ID_SignExtendOut;
wire [31:0] IF_PCnew;
wire IF_write, PC_write, bubble;
wire [31:0] EX_BusB_BusW;
wire [31:0] DataMem_DataIn;
wire [1:0] addrSel;
// Pipeline Registers: IF_ID
// Input Instruction memory DataOut
reg [31:0] IF_ID_PCnew;
reg [31:0] IF_ID_Instruction;
wire [31:0] ID_Instruction = IF_ID_Instruction;

```

```

wire [31:0] ID_PCnew = IF_ID_PCnew;
// Pipeline Registers: ID_EX
// Control Signals
reg ID_EX_RegDst, ID_EX_RegWrite, ID_EX_MemRead,
ID_EX_MemWrite, ID_EX_MemToReg, ID_EX_Jal;
reg [3:0] ID_EX_ALUOp;
wire ID_RegDst, ID_RegWrite, ID_MemRead, ID_MemWrite,
ID_Branch, ID_Jump, ID_MemToReg, ID_Jr, ID_Jal;
wire [3:0] ID_ALUOp;
wire EX_RegDst, EX_RegWrite, EX_MemRead, EX_MemWrite,
EX_MemToReg, EX_Jal;
wire [3:0] EX_ALUOp;
assign {EX_MemToReg, EX_RegDst, EX_RegWrite, EX_MemRead, EX_MemWrite,
EX_ALUOp, EX_Jal} = {ID_EX_MemToReg, ID_EX_RegDst, ID_EX_RegWrite,
ID_EX_MemRead, ID_EX_MemWrite, ID_EX_ALUOp, ID_EX_Jal};
//Forwarding Unit Signals
reg [1:0] ID_EX_ALUOpCtrlA, ID_EX_ALUOpCtrlB ;
reg ID_EX_DataMemForwardCtrl_EX , ID_EX_DataMemForwardCtrl_MEM ;
wire [1:0] ID_ALUOpCtrlA, ID_ALUOpCtrlB ;
wire ID_DataMemForwardCtrl_EX , ID_DataMemForwardCtrl_MEM ;
wire [1:0] EX_ALUOpCtrlA, EX_ALUOpCtrlB ;
wire EX_DataMemForwardCtrl_EX , EX_DataMemForwardCtrl_MEM ;
assign {EX_ALUOpCtrlA, EX_ALUOpCtrlB, EX_DataMemForwardCtrl_EX ,
EX_DataMemForwardCtrl_MEM} = {ID_EX_ALUOpCtrlA, ID_EX_ALUOpCtrlB,
ID_EX_DataMemForwardCtrl_EX , ID_EX_DataMemForwardCtrl_MEM};
// Data
reg [20:0] ID_EX_Instruction;
(* keep = "true"*)reg [31:0] ID_EX_SignExtendOut;
reg [31:0] ID_EX_BusB, ID_EX_BusA, ID_EX_PCnew;
wire [31:0] ID_BusA, ID_BusB;
wire [20:0] EX_Instruction;
wire [31:0] EX_SignExtendOut, EX_BusB, EX_BusA, EX_PCnew;
assign {EX_Instruction, EX_SignExtendOut, EX_BusB, EX_BusA, EX_PCnew} =
{ID_EX_Instruction, ID_EX_SignExtendOut, ID_EX_BusB, ID_EX_BusA,
ID_EX_PCnew}
;
// Pipeline Registers: EX_MEM
// Control unit Signals
reg EX_MEM_MemToReg, EX_MEM_RegWrite, EX_MEM_MemRead,
EX_MEM_MemWrite;
wire MEM_MemToReg, MEM_RegWrite, MEM_MemRead, MEM_MemWrite;
assign {MEM_MemToReg, MEM_RegWrite, MEM_MemRead, MEM_MemWrite} =
{EX_MEM_MemToReg, EX_MEM_RegWrite, EX_MEM_MemRead, EX_MEM_MemWrite};
// Forwarding Signals
reg EX_MEM_DataMemForwardCtrl_MEM;
wire MEM_DataMemForwardCtrl_MEM;
assign MEM_DataMemForwardCtrl_MEM = EX_MEM_DataMemForwardCtrl_MEM;
// Data
reg [31:0] EX_MEM_ALUOut;
wire [31:0] EX_ALUOut, ALUOut;
assign EX_ALUOut = (EX_Jal == 1)? EX_PCnew:ALUOut;
wire [31:0] MEM_ALUOut = EX_MEM_ALUOut;
reg [31:0] EX_MEM_BusB_BusW;
wire [31:0] MEM_BusB_BusW = EX_MEM_BusB_BusW;
reg [4:0] EX_MEM_Rw;
wire [4:0] MEM_Rw = EX_MEM_Rw;
wire [4:0] EX_Rw;

```

```

// Pipeline Registers: MEM_WB
// Control unit Signals
reg MEM_WB_MemToReg, MEM_WB_RegWrite;
wire WB_MemToReg, WB_RegWrite;
assign {WB_MemToReg, WB_RegWrite} = {MEM_WB_MemToReg, MEM_WB_RegWrite};
// Data
wire [31:0] MEM_DataMem_DataOut;
reg [31:0] MEM_WB_DataMem_DataOut;
wire [31:0] WB_DataMem_DataOut = MEM_WB_DataMem_DataOut;
reg [31:0] MEM_WB_ALUOut;
wire [31:0] WB_ALUOut = MEM_WB_ALUOut;
reg [4:0] MEM_WB_Rw;
wire [4:0] WB_Rw = MEM_WB_Rw;
//Module Instantiation
// Instruction Memory
InstructionMemory InstructionMemory(
    .Data(IF_Instruction),
    .Address(PC)
);
// PipelinedControl Unit
PipelinedControl dhiraj_plc(
    .RegSrc(ID_MemToReg),
    .RegDst(ID_RegDst),
    .UseShamt(ID_UseShamt),
    .UseImmed(ID_UseImmed),
    .RegWrite(ID_RegWrite),
    .MemRead(ID_MemRead),
    .MemWrite(ID_MemWrite),
    .Branch(ID_Branch),
    .Jump(ID_Jump),
    .JumpSel(ID_Jr),
    .Jal(ID_Jal),
    .SignExtend(SignExtend),
    .ALUOp(ID_ALUOp),
    .OpCode(ID_Instruction[31:26]),
    .FuncCode(ID_Instruction[5:0])
);
// Hazard Unit
wire Zero;
HazardUnit dhiraj_hu(
    .IF_write(IF_write),
    .PC_write(PC_write),
    .bubble(bubble),
    .addrSel(addrSel),
    .Jump(ID_Jump),
    .Branch(ID_Branch),
    .ALUZero(Zero),
    .memReadEX(EX_MemRead),
    .ID_Rs(ID_Instruction[25:21]),
    .ID_Rt(ID_Instruction[20:16]),
    .UseShamt(ID_UseShamt),
    .UseImmed(ID_UseImmed),
    .Clk(CLK),
    .Jr(ID_Jr),
    .EX_RegWrite(EX_RegWrite),
    .MEM_RegWrite(MEM_RegWrite),
    .EX_Rw(EX_Rw),

```

```

.MEM_Rw(MEM_Rw),
.Rst(Reset_L)
);
// Register File
RegisterFile dhiraj_rf(
.BusA(ID_BusA) ,
.BusB(ID_BusB) ,
.BusW(BusW) ,
.RA(ID_Instruction[25:21]),
.RB(ID_Instruction[20:16]),
.RW(WB_Rw),
.RegWr(WB_RegWrite),
.Clk(CLK)
);
// Sign Extend
SignExtender SE(
.imm16(ID_Instruction[15:0]),
.signExtImm(ID_SignExtendOut),
.signExtend(SignExtend)
);
// Forwarding Unit
ForwardingUnit dhiraj_fu(
.UseShamt(ID_UseShamt) ,
.UseImmed (ID_UseImmed),
.ID_Rs(ID_Instruction[25:21])
,
.ID_Rt(ID_Instruction[20:16])
,
.EX_Rw (EX_Rw),
.MEM_Rw (MEM_Rw),
.EX_RegWrite (EX_RegWrite) ,
.MEM_RegWrite (MEM_RegWrite)
,
.ALUOpCtrlA(ID_ALUOpCtrlA) ,
.ALUOpCtrlB(ID_ALUOpCtrlB) ,
.DataMemForwardCtrl_EX(ID_DataMemForwardCtrl_EX) ,
.DataMemForwardCtrl_MEM(ID_DataMemForwardCtrl_MEM)
);
// ALU control
ALUControl ALUControl(
.ALUCtrl(ALUCtrl),
.ALUOp(EX_ALUOp),
.FuncCode(EX_Instruction[5:0])
);
// ALU
ALU ALU(
.BusW(ALUOut),
.Zero(Zero),
.BusA(ALUInputA),
.BusB(ALUInputB),
.ALUCtrl(ALUCtrl)
);
// Data Memory
DataMemory DataMemory(
.ReadData(MEM_DataMem_DataOut),
.Address(MEM_ALUOut[5:0]),
.WriteData(DataMem_DataIn),

```

```

.MemoryRead(MEM_MemRead),
.MemoryWrite(MEM_MemWrite),
.Clock(CLK)
);
always @(*) begin
// PC Operations
case(addrSel)
2'b00: PCnext <= IF_PCnew;
2'b01: PCnext <=
(ID_Jr==1)?ID_BusA:{ID_PCnew[31:28],ID_Instruction[25:0],2'b00};
2'b10: PCnext <= (EX_SignExtendOut<<2) + EX_PCnew;
default: PCnext <= IF_PCnew;
endcase
// 4:1 MUX for ALU Input A
case(EX_ALUOpCtrlA)
2'b00: ALUInputA <= EX_Instruction[10:6];
2'b01: ALUInputA <= BusW;
2'b10: ALUInputA <= MEM_ALUOut;
2'b11: ALUInputA <= EX_BusA;
default: ALUInputA <= 32'b0;
endcase
// 4:1 MUX for ALU Input B
case(EX_ALUOpCtrlB)
2'b00: ALUInputB <= EX_SignExtendOut;
2'b01: ALUInputB <= BusW;
2'b10: ALUInputB <= MEM_ALUOut;
2'b11: ALUInputB <= EX_BusB;
default: ALUInputB <= 32'b0;
endcase
end
// DEFINING MUX in top module
assign EX_Rw = (EX_Jal==0)?((EX_RegDst == 0)?
EX_Instruction[20:16] : EX_Instruction[15:11]):5'b11111;
assign EX_BusB_BusW = (EX_DataMemForwardCtrl_EX== 0) ? EX_BusB : BusW;
assign DataMem_DataIn = (MEM_DataMemForwardCtrl_MEM == 0) ? MEM_BusB_BusW :
BusW;
assign BusW = (WB_MemToReg == 1) ? WB_DataMem_DataOut :
WB_ALUOut;
// Assign the outputs
assign dMemOut = MEM_DataMem_DataOut;
// Assign the wires value according to the block diagram
assign IF_PCnew = PC + 4;
// At every negedge of clock or reset, assign the PC
always @(negedge CLK) begin
// Update The PC
if(Reset_L == 0)
PC <= startPC;
else if(PC_write == 1)
PC <= PCnext;
else
PC <= PC;
// Update the IF/ID Register
if (IF_write == 1) begin
IF_ID_PCnew <= IF_PCnew;
IF_ID_Instruction[31:0] <= IF_Instruction[31:0];
end
else begin

```



```

IF_ID_PCnew <= IF_ID_PCnew;
IF_ID_Instruction[31:0] <= IF_ID_Instruction[31:0];
end
// Update the ID_EX Register only if bubble is 0
if (bubble == 1)
{ID_EX_RegDst, ID_EX_RegWrite, ID_EX_MemRead, ID_EX_MemWrite,
ID_EX_Jal, ID_EX_MemToReg, ID_EX_ALUOp} <= 12'b000000000000;
else
{ID_EX_RegDst, ID_EX_RegWrite, ID_EX_MemRead, ID_EX_MemWrite,
ID_EX_MemToReg, ID_EX_Jal, ID_EX_ALUOp} <= {ID_RegDst, ID_RegWrite,
ID_MemRead, ID_MemWrite, ID_MemToReg, ID_Jal, ID_ALUOp};
{ID_EX_ALUOpCtrlA, ID_EX_ALUOpCtrlB, ID_EX_DataMemForwardCtrl_EX ,
ID_EX_DataMemForwardCtrl_MEM} <= {ID_ALUOpCtrlA, ID_ALUOpCtrlB,
ID_DataMemForwardCtrl_EX , ID_DataMemForwardCtrl_MEM};
{ID_EX_Instruction, ID_EX_SignExtendOut, ID_EX_BusB, ID_EX_BusA,
ID_EX_PCnew}<= {ID_Instruction[20:0], ID_SignExtendOut, ID_BusB, ID_BusA,
ID_PCnew};
// Update the EX_MEM Register
{EX_MEM_MemToReg, EX_MEM_RegWrite, EX_MEM_MemRead, EX_MEM_MemWrite} <=
{EX_MemToReg, EX_RegWrite, EX_MemRead, EX_MemWrite};
EX_MEM_DataMemForwardCtrl_MEM <= EX_DataMemForwardCtrl_MEM;
EX_MEM_ALUOut <= EX_ALUOut;
EX_MEM_BusB_BusW <= EX_BusB_BusW;
EX_MEM_Rw <= EX_Rw;
// Update the MEM_WB Register
{MEM_WB_MemToReg, MEM_WB_RegWrite} <= {MEM_MemToReg, MEM_RegWrite} ;
MEM_WB_DataMem_DataOut <= MEM_DataMem_DataOut;
MEM_WB_ALUOut <= MEM_ALUOut;
MEM_WB_Rw <= MEM_Rw;
end
endmodule

```

## 8. Register File

```

`timescale 1ns / 1ps
module RegisterFile(BusA , BusB , BusW , RA, RB, RW, RegWr , Clk);
// Define Inputs, Outputs and 32*32 RegisterMemory
input [4:0] RA;
input [4:0] RB;
input [4:0] RW;
input [31:0] BusW;
input RegWr, Clk;
output [31:0] BusA;
output [31:0] BusB;
reg [31:0] RegisterMemory[31:0];
// Read continuously from the register memory
assign BusB = (RB==0) ? 0 : RegisterMemory[RB];
assign BusA = (RA==0) ? 0 : RegisterMemory[RA];
// Write in the memory at the posedge of clock
always @(posedge Clk) begin
if (RegWr == 1 && RW!=0) begin
RegisterMemory[RW] <= BusW;
end
end
endmodule

```

## 9. Sign Extender

```
`timescale 1ns / 1ps
module SignExtender(signExtImm, signExtend, imm16);
output [31:0] signExtImm;
input [15:0] imm16;
input signExtend;
assign signExtImm = signExtend? ({16{imm16[15]}}, imm16[15:0]):
{16'b0, imm16[15:0]};
Endmodule
```

{ Testbench and Instruction memory files are not written here as it was provided by the TA}

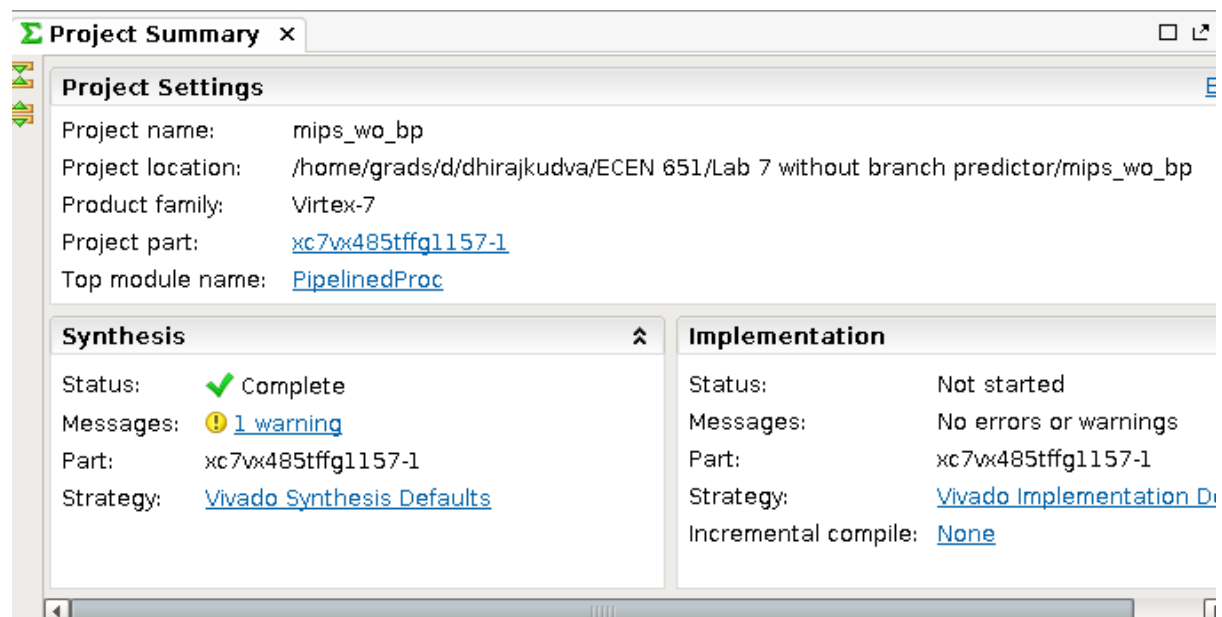
### Result:

NFO: [USF-XSim-97] XSim simulation ran for 1000ns  
Launch simulation: Time (s): cpu = 00:00:12 ; elapsed = 00:00:14 . Memory (MB): peak = 6070.109 ; gain  
un all

```
Results of Program 1 passed
Results of Program 2 passed
Result 1 of Program 3 passed
Result 2 of Program 3 passed
Result 3 of Program 3 passed
Result 4 of Program 3 passed
Result 5 of Program 3 passed
Result 6 of Program 3 passed
Result 7 of Program 3 passed
Result 8 of Program 3 passed
Result 9 of Program 3 passed
Result 10 of Program 3 passed
Result 11 of Program 3 passed
Result 12 of Program 3 passed
Result 1 of Program 4 passed
Result 2 of Program 4 passed
Result 3 of Program 4 passed
Result 4 of Program 4 passed
```

.ll tests passed

finish called at time : 18361 ns : File "/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predi  
un: Time (s): cpu = 00:00:05 ; elapsed = 00:00:06 . Memory (MB): peak = 6070.109 ; gain = 0.000 ; free



**Project Summary**

**Project Settings**

- Project name: mips\_wo\_bp
- Project location: /home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/mips\_wo\_bp
- Product family: Virtex-7
- Project part: xc7vx485tffg1157-1
- Top module name: PipelinedProc

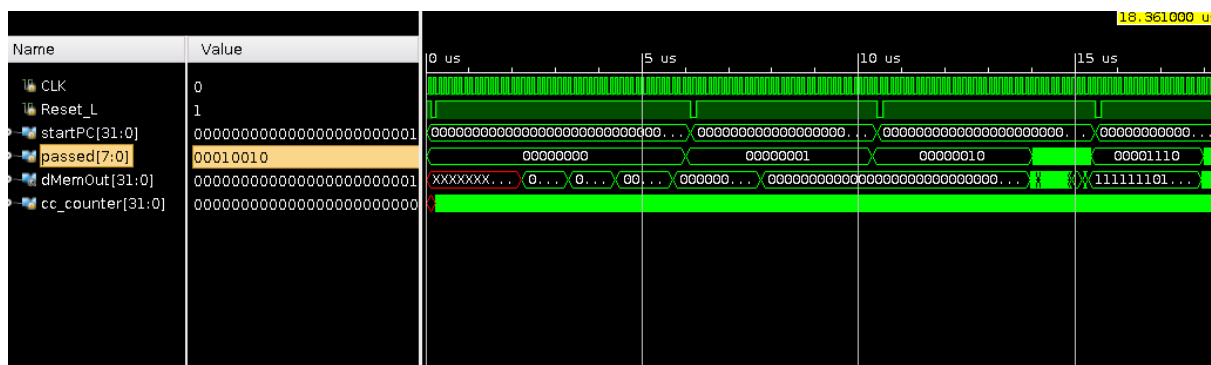
Synthesis	Implementation
Status: <span style="color: green;">✔</span> Complete	Status: Not started
Messages: <span style="color: orange;">⚠</span> 1 warning	Messages: No errors or warnings
Part: xc7vx485tffg1157-1	Part: xc7vx485tffg1157-1
Strategy: <a href="#">Vivado Synthesis Defaults</a>	Strategy: <a href="#">Vivado Implementation Defaults</a>
	Incremental compile: <a href="#">None</a>

```

-----
| Clock Summary
| -----
-----

```

Clock	Waveform(ns)	Period(ns)	Frequency(MHz)
clk1	{0.000 4.150}	8.300	120.482



## SYNTHESIS REPORT:

### \*\*\* Running vivado

```

with args -log PipelinedProc.vds -m64 -mode batch -messageDb vivado.pb -notrace -
source PipelinedProc.tcl

```

### \*\*\*\*\* Vivado v2015.2 (64-bit)

\*\*\*\* SW Build 1266856 on Fri Jun 26 16:35:25 MDT 2015

\*\*\*\* IP Build 1264090 on Wed Jun 24 14:22:01 MDT 2015

\*\* Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.

```
source PipelinedProc.tcl -notrace
```

```
Command: synth_design -top PipelinedProc -part xc7vx485tffg1157-1
```

```
Starting synth_design
```

```
Attempting to get a license for feature 'Synthesis' and/or device 'xc7vx485t'
```

```
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7vx485t'
```

```
INFO: [Common 17-1223] The version limit for your license is '2017.12' and will expire in -
696 days. A version limit expiration means that, although you may be able to continue to
use the current version of tools or IP with this license, you will not be eligible for any
updates or new releases.
```

```

-----
Starting RTL Elaboration : Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 . Memory (MB): peak
= 1082.000 ; gain = 154.520 ; free physical = 393020 ; free virtual = 537236
-----

```

INFO: [Synth 8-638] synthesizing module 'PipelinedProc' [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/pipeline\_process.v:2]  
INFO: [Synth 8-5534] Detected attribute (\* keep = "true" \*)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/pipeline\_process.v:56]  
INFO: [Synth 8-638] synthesizing module 'InstructionMemory'  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/InstructionMemory.v:8]  
Parameter T\_rd bound to: 20 - type: integer  
Parameter MemSize bound to: 40 - type: integer  
INFO: [Synth 8-256] done synthesizing module 'InstructionMemory' (1#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/InstructionMemory.v:8]  
INFO: [Synth 8-638] synthesizing module 'PipelinedControl'  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/pipeline\_control.v:47]  
INFO: [Synth 8-256] done synthesizing module 'PipelinedControl' (2#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/pipeline\_control.v:47]  
INFO: [Synth 8-638] synthesizing module 'HazardUnit' [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/hazard.v:2]  
INFO: [Synth 8-226] default block is never used [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/hazard.v:31]  
INFO: [Synth 8-256] done synthesizing module 'HazardUnit' (3#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/hazard.v:2]  
INFO: [Synth 8-638] synthesizing module 'RegisterFile' [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/register\_file.v:2]  
INFO: [Synth 8-256] done synthesizing module 'RegisterFile' (4#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/register\_file.v:2]  
INFO: [Synth 8-638] synthesizing module 'SignExtender' [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/sign\_extender.v:2]  
INFO: [Synth 8-256] done synthesizing module 'SignExtender' (5#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/sign\_extender.v:2]  
INFO: [Synth 8-638] synthesizing module 'ForwardingUnit'  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/forwarding.v:2]  
INFO: [Synth 8-256] done synthesizing module 'ForwardingUnit' (6#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/forwarding.v:2]  
INFO: [Synth 8-638] synthesizing module 'ALUControl' [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/alu\_control.v:32]  
INFO: [Synth 8-256] done synthesizing module 'ALUControl' (7#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/alu\_control.v:32]  
INFO: [Synth 8-638] synthesizing module 'ALU' [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/alu.v:17]

INFO: [Synth 8-256] done synthesizing module 'ALU' (8#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/alu.v:17]  
INFO: [Synth 8-638] synthesizing module 'DataMemory' [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/data\_memory.v:2]  
INFO: [Synth 8-256] done synthesizing module 'DataMemory' (9#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/data\_memory.v:2]  
INFO: [Synth 8-226] default block is never used [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/pipeline\_process.v:212]  
INFO: [Synth 8-226] default block is never used [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/pipeline\_process.v:220]  
INFO: [Synth 8-256] done synthesizing module 'PipelinedProc' (10#1)  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/pipeline\_process.v:2]

-----  
Finished RTL Elaboration : Time (s): cpu = 00:00:07 ; elapsed = 00:00:08 . Memory (MB):  
peak = 1118.297 ; gain = 190.816 ; free physical = 392981 ; free virtual = 537198  
-----

#### Report Check Netlist:

+	-----+	+	-----+	+	-----+	+	-----+
	Item		Errors		Warnings		Status   Description
+	-----+	+	-----+	+	-----+	+	-----+
	1		multi_driven_nets		0		0   Passed   Multi driven nets
+	-----+	+	-----+	+	-----+	+	-----+

-----  
Finished RTL Optimization Phase 1 : Time (s): cpu = 00:00:07 ; elapsed = 00:00:08 . Memory (MB): peak = 1118.297 ; gain = 190.816 ; free physical = 392981 ; free virtual = 537198  
-----

INFO: [Device 21-403] Loading part xc7vx485tffg1157-1  
INFO: [Project 1-570] Preparing netlist for logic optimization

#### Processing XDC Constraints

Initializing timing engine

Parsing XDC File [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/mips\_wo\_bp/mips\_wo\_bp.srscs/constrs\_1/new/clock.xdc]

Finished Parsing XDC File [/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/mips\_wo\_bp/mips\_wo\_bp.srscs/constrs\_1/new/clock.xdc]

Completed Processing XDC Constraints

INFO: [Project 1-111] Unisim Transformation Summary:  
No Unisim elements were transformed.

Constraint Validation Runtime : Time (s): cpu = 00:00:00.01 ; elapsed = 00:00:00.02 .  
Memory (MB): peak = 1586.102 ; gain = 0.000 ; free physical = 392673 ; free virtual = 536890

-----  
Finished Constraint Validation : Time (s): cpu = 00:00:20 ; elapsed = 00:00:22 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392671 ; free virtual = 536888  
-----

-----  
Start Loading Part and Timing Information  
-----

Loading part: xc7vx485tffg1157-1  
-----

-----  
Finished Loading Part and Timing Information : Time (s): cpu = 00:00:20 ; elapsed = 00:00:22 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392671 ; free virtual = 536888  
-----

-----  
Start Applying 'set\_property' XDC Constraints  
-----

-----  
Finished applying 'set\_property' XDC Constraints : Time (s): cpu = 00:00:20 ; elapsed = 00:00:22 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392671 ; free virtual = 536888  
-----

INFO: [Synth 8-5546] ROM "Jump" won't be mapped to RAM because it is too sparse  
INFO: [Synth 8-5546] ROM "Branch" won't be mapped to RAM because it is too sparse  
INFO: [Synth 8-5546] ROM "RegWrite" won't be mapped to RAM because it is too sparse  
ROM size is below threshold of ROM address width. It will be mapped to LUTs  
INFO: [Synth 8-5546] ROM "UseShamt" won't be mapped to RAM because it is too sparse  
INFO: [Synth 8-5546] ROM "Jal" won't be mapped to RAM because it is too sparse  
INFO: [Synth 8-5546] ROM "MemRead" won't be mapped to RAM because it is too sparse  
INFO: [Synth 8-5546] ROM "MemWrite" won't be mapped to RAM because it is too sparse  
ROM size is below threshold of ROM address width. It will be mapped to LUTs  
INFO: [Synth 8-3537] HDL ADVISOR - The operator resource <adder> is shared. To prevent sharing consider applying a KEEP on the inputs of the operator  
[/home/grads/d/dhirajkudva/ECEN 651/Lab 7 without branch predictor/alu.v:25]  
INFO: [Synth 8-5562] The signal DataMemory\_reg is implemented as block RAM but is better mapped onto distributed LUT RAM for the following reason(s): The \*depth (6 address bits)\* is shallow. Please use attribute (\* ram\_style = "distributed" \*) to instruct Vivado to infer distributed LUT RAM.  
-----

-----  
Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:21 ; elapsed = 00:00:23 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392661 ; free virtual = 536878  
-----

Report RTL Partitions:

+ +-----+-----+-----+

| |RTL Partition |Replication |Instances |

+-----+-----+-----+

+-----+-----+-----+

---

## Start RTL Component Statistics

---

### Detailed RTL Component Info :

#### +---Adders :

3 Input 32 Bit Adders := 1

2 Input 32 Bit Adders := 2

#### +---XORs :

2 Input 32 Bit XORs := 1

#### +---Registers :

32 Bit Registers := 1

#### +---RAMs :

2K Bit RAMs := 1

1024 Bit RAMs := 1

#### +---Muxes :

137 Input 32 Bit Muxes := 1

2 Input 32 Bit Muxes := 13

15 Input 32 Bit Muxes := 2

4 Input 32 Bit Muxes := 3

2 Input 5 Bit Muxes := 2

15 Input 4 Bit Muxes := 1

16 Input 4 Bit Muxes := 1

15 Input 2 Bit Muxes := 4

5 Input 2 Bit Muxes := 1

8 Input 2 Bit Muxes := 1

2 Input 2 Bit Muxes := 9

2 Input 1 Bit Muxes := 4

5 Input 1 Bit Muxes := 3

3 Input 1 Bit Muxes := 3

15 Input 1 Bit Muxes := 6

4 Input 1 Bit Muxes := 4

---

## Finished RTL Component Statistics

---

---

## Start RTL Hierarchical Component Statistics

---

### Hierarchical RTL Component report

Module PipelinedProc

### Detailed RTL Component Info :

#### +---Adders :

2 Input 32 Bit Adders := 2

+---Muxes :

2 Input 32 Bit Muxes := 6

4 Input 32 Bit Muxes := 3

2 Input 5 Bit Muxes := 2

2 Input 1 Bit Muxes := 1

Module InstructionMemory

Detailed RTL Component Info :

+---Muxes :

137 Input 32 Bit Muxes := 1

Module PipelinedControl

Detailed RTL Component Info :

+---Muxes :

15 Input 4 Bit Muxes := 1

15 Input 2 Bit Muxes := 4

2 Input 1 Bit Muxes := 3

5 Input 1 Bit Muxes := 3

3 Input 1 Bit Muxes := 3

15 Input 1 Bit Muxes := 6

4 Input 1 Bit Muxes := 1

Module HazardUnit

Detailed RTL Component Info :

+---Muxes :

5 Input 2 Bit Muxes := 1

8 Input 2 Bit Muxes := 1

4 Input 1 Bit Muxes := 3

Module RegisterFile

Detailed RTL Component Info :

+---RAMs :

1024 Bit RAMs := 1

+---Muxes :

2 Input 32 Bit Muxes := 2

Module SignExtender

Detailed RTL Component Info :

+---Muxes :

2 Input 32 Bit Muxes := 1

Module ForwardingUnit

Detailed RTL Component Info :

+---Muxes :

2 Input 2 Bit Muxes := 6

Module ALUControl

Detailed RTL Component Info :

+---Muxes :

16 Input 4 Bit Muxes := 1



## Module ALU

### Detailed RTL Component Info :

#### +---Adders :

3 Input 32 Bit Adders := 1

#### +---XORs :

2 Input 32 Bit XORs := 1

#### +---Muxes :

15 Input 32 Bit Muxes := 2

2 Input 32 Bit Muxes := 4

2 Input 2 Bit Muxes := 3

## Module DataMemory

### Detailed RTL Component Info :

#### +---Registers :

32 Bit Registers := 1

#### +---RAMs :

2K Bit RAMs := 1

---

### Finished RTL Hierarchical Component Statistics

---

---

### Start Part Resource Summary

---

#### Part Resources:

DSPs: 2800 (col length:140)

BRAMs: 2060 (col length: RAMB18 140 RAMB36 70)

---

### Finished Part Resource Summary

---

Start Parallel Synthesis Optimization : Time (s): cpu = 00:00:22 ; elapsed = 00:00:23 .

Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392661 ; free virtual = 536878

---

### Start Cross Boundary Optimization

---

INFO: [Synth 8-5544] ROM "BusW0" won't be mapped to Block RAM because address size (1) smaller than threshold (5)

INFO: [Synth 8-5544] ROM "BusW0" won't be mapped to Block RAM because address size (1) smaller than threshold (5)

INFO: [Synth 8-5544] ROM "BusW0" won't be mapped to Block RAM because address size (1) smaller than threshold (5)

---

Finished Cross Boundary Optimization : Time (s): cpu = 00:00:23 ; elapsed = 00:00:24 .

Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392661 ; free virtual = 536878

-----  
Finished Parallel Reinference : Time (s): cpu = 00:00:23 ; elapsed = 00:00:24 . Memory (MB):  
peak = 1586.102 ; gain = 658.621 ; free physical = 392661 ; free virtual = 536878

Report RTL Partitions:

```
+-----+-----+-----+
| | RTL Partition | Replication | Instances |
```

```
+-----+-----+-----+
```

```
+-----+-----+-----+
```

INFO: [Synth 8-5562] The signal DataMemory/DataMemory\_reg is implemented as block RAM but is better mapped onto distributed LUT RAM for the following reason(s): The \*depth (6 address bits)\* is shallow. Please use attribute (\* ram\_style = "distributed" \*) to instruct Vivado to infer distributed LUT RAM.

-----  
Start ROM, RAM, DSP and Shift Register Reporting  
-----

Block RAM:

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Module Name | RTL Object | PORT A (Depth x Width) | W | R | PORT B (Depth x Width) | W | R | OUT_REG | RAMB18 | RAMB36 | Hierarchical Name |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| PipelinedProc | DataMemory/DataMemory_reg | 64 x 32(READ_FIRST) | W | | 64 x 32(WRITE_FIRST) | | R | Port A and B | 1 | 0 | PipelinedProc/extram__4 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Note: The table shows the Block RAMs at the current stage of the synthesis flow. Some Block RAMs may be reimplemented as non Block RAM primitives later in the synthesis flow. Multiple instantiated Block RAMs are reported only once. "Hierarchical Name" reflects the Block RAM name as it appears in the hierarchical module and only part of it is displayed.

Distributed RAM:

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
--+
| Module Name | RTL Object | Inference | Size (Depth x Width) | Primitives | Hierarchical Name |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
--+
| PipelinedProc | dhiraj_rf/RegisterMemory_reg | Implied | 32 x 32 | RAM32M x 12 | PipelinedProc/ram__1 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
--+
```

Note: The table shows the Distributed RAMs at the current stage of the synthesis flow. Some Distributed RAMs may be reimplemented as non Distributed RAM primitives later in the synthesis flow. Multiple instantiated RAMs are reported only once. "Hierarchical Name" reflects the Distributed RAM name as it appears in the hierarchical module and only part of it is displayed.

-----  
Finished ROM, RAM, DSP and Shift Register Reporting  
-----

INFO: [Synth 8-3333] propagating constant 0 across sequential element

(\IF\_ID\_Instruction\_reg[30] )

WARNING: [Synth 8-3332] Sequential element (\IF\_ID\_Instruction\_reg[30] ) is unused and will be removed from module PipelinedProc.

-----  
Start Area Optimization  
-----

-----  
Finished Area Optimization : Time (s): cpu = 00:00:31 ; elapsed = 00:00:33 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392633 ; free virtual = 536850  
-----

Finished Parallel Area Optimization : Time (s): cpu = 00:00:31 ; elapsed = 00:00:33 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392633 ; free virtual = 536850

Report RTL Partitions:

```
+ +-----+-----+-----+
| |RTL Partition |Replication |Instances |
+ +-----+-----+-----+
+ +-----+-----+-----+
```

Finished Parallel Synthesis Optimization : Time (s): cpu = 00:00:31 ; elapsed = 00:00:33 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392633 ; free virtual = 536850

-----  
Start Timing Optimization  
-----

-----  
Start Applying XDC Timing Constraints  
-----

-----  
Finished Applying XDC Timing Constraints : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392586 ; free virtual = 536803  
-----

INFO: [Synth 8-5563] The signal DataMemory/DataMemory\_reg is implemented as distributed LUT RAM for the following reason(s): The \*timing constraints\* suggest that the chosen mapping will yield better timing results.

INFO: [Synth 8-5563] The signal DataMemory/DataMemory\_reg is implemented as distributed LUT RAM for the following reason(s): The \*timing constraints\* suggest that the chosen mapping will yield better timing results.

-----  
Finished Timing Optimization : Time (s): cpu = 00:00:42 ; elapsed = 00:00:43 . Memory (MB): peak = 1586.102 ; gain = 658.621 ; free physical = 392584 ; free virtual = 536801  
-----

Report RTL Partitions:

RTL Partition	Replication	Instances

-----  
Start Technology Mapping  
-----  
-----

Finished Technology Mapping : Time (s): cpu = 00:00:44 ; elapsed = 00:00:46 . Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392529 ; free virtual = 536746  
-----

Report RTL Partitions:

RTL Partition	Replication	Instances

-----  
Start IO Insertion  
-----  
-----

Start Final Netlist Cleanup  
-----  
-----

Finished Final Netlist Cleanup  
-----  
-----

Finished IO Insertion : Time (s): cpu = 00:00:45 ; elapsed = 00:00:47 . Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392528 ; free virtual = 536745  
-----

Report Check Netlist:

	Item	Errors	Warnings	Status	Description
1	multi_driven_nets	0	0	Passed	Multi driven nets

Start Renaming Generated Instances

Finished Renaming Generated Instances : Time (s): cpu = 00:00:45 ; elapsed = 00:00:47 .  
Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392528 ; free virtual = 536745

Report RTL Partitions:

	RTL Partition	Replication	Instances

Start Rebuilding User Hierarchy

Finished Rebuilding User Hierarchy : Time (s): cpu = 00:00:45 ; elapsed = 00:00:47 . Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392528 ; free virtual = 536745

Start Renaming Generated Ports : Time (s): cpu = 00:00:45 ; elapsed = 00:00:47 . Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392528 ; free virtual = 536745

Finished Renaming Generated Ports : Time (s): cpu = 00:00:45 ; elapsed = 00:00:47 . Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392528 ; free virtual = 536745

Start Writing Synthesis Report

Report BlackBoxes:

	BlackBox name	Instances

#### Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	59
3	LUT1	86
4	LUT2	63
5	LUT3	101
6	LUT4	257
7	LUT5	337
8	LUT6	440
9	MUXF7	67
10	RAM32M	12
11	RAM64X1S	32
12	FDRE	434
13	FDSE	5
14	IBUF	34
15	OBUF	32

#### Report Instance Areas:

	Instance	Module	Cells
1	top		1960
2	ALU	ALU	563
3	DataMemory	DataMemory	96
4	dhiraj_hu	HazardUnit	113
5	dhiraj_rf	RegisterFile	45

Finished Writing Synthesis Report : Time (s): cpu = 00:00:45 ; elapsed = 00:00:47 . Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392528 ; free virtual = 536745

Synthesis finished with 0 errors, 0 critical warnings and 1 warnings.

Synthesis Optimization Runtime : Time (s): cpu = 00:00:31 ; elapsed = 00:00:32 . Memory (MB): peak = 1603.102 ; gain = 91.297 ; free physical = 392528 ; free virtual = 536745

Synthesis Optimization Complete : Time (s): cpu = 00:00:45 ; elapsed = 00:00:47 . Memory (MB): peak = 1603.102 ; gain = 675.621 ; free physical = 392528 ; free virtual = 536745

INFO: [Project 1-571] Translating synthesized netlist

INFO: [Netlist 29-17] Analyzing 137 Unisim elements for replacement

INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds

INFO: [Project 1-570] Preparing netlist for logic optimization

INFO: [Opt 31-138] Pushed 2 inverter(s) to 439 load pin(s).

INFO: [Project 1-111] Unisim Transformation Summary:

A total of 44 instances were transformed.

RAM32M => RAM32M (RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMS32, RAMS32): 12 instances

RAM64X1S => RAM64X1S (inverted pins: WCLK) (RAMS64E): 32 instances

INFO: [Common 17-83] Releasing license: Synthesis

52 Infos, 1 Warnings, 0 Critical Warnings and 0 Errors encountered.

synth\_design completed successfully

synth\_design: Time (s): cpu = 00:00:44 ; elapsed = 00:00:46 . Memory (MB): peak = 1635.117 ; gain = 599.117 ; free physical = 392528 ; free virtual = 536745

report\_utilization: Time (s): cpu = 00:00:00.12 ; elapsed = 00:00:00.17 . Memory (MB): peak = 1667.141 ; gain = 0.000 ; free physical = 392526 ; free virtual = 536743

INFO: [Common 17-206] Exiting Vivado at Wed Nov 27 23:31:14 2019...

## PART 2: IMPLEMENTING THE STATIC BRANCH PREDICTOR

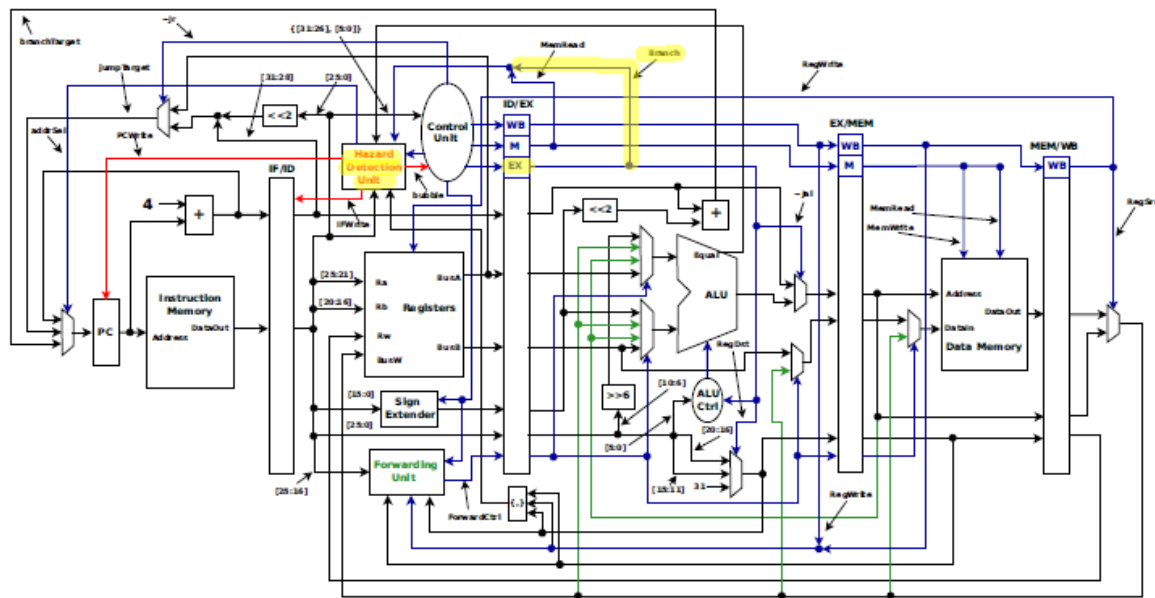


Figure 2: MIPS Block Diagram with Static Branch Prediction

Program Source code:

### 1. ALU

```
`timescale 1ns / 1ps
// define the ALU operations (ALUOp)
`define AND 4'b0000
`define OR 4'b0001
`define ADD 4'b0010
`define SLL 4'b0011
`define SRL 4'b0100
`define SUB 4'b0110
`define SLT 4'b0111
`define ADDU 4'b1000
`define SUBU 4'b1001
`define XOR 4'b1010
`define SLTU 4'b1011
`define NOR 4'b1100
`define SRA 4'b1101
`define LUI 4'b1110

module ALU(BusW, Zero, BusA, BusB, ALUCtrl);
// Define Inputs and Outputs
input signed [31:0]BusA;
input signed [31:0]BusB;
input [3:0]ALUCtrl;
output reg[31:0]BusW;
output reg Zero;
always @(*)
begin
case(ALUCtrl)
// Define ALU operations based on the ALUCtrl
`AND: BusW <= BusA & BusB;
`OR: BusW <= BusA | BusB;
```



```

`ADD: BusW <= BusA + BusB;
`SLL: BusW <= BusB << BusA;
`SRL: BusW <= BusB >> BusA;
`SUB: BusW <= BusA - BusB;
`SLT: BusW <= (BusA < BusB)? 1 : 0;
`ADDU: BusW <= BusA + BusB;
`SUBU: BusW <= BusA - BusB;
`XOR: BusW <= BusA ^ BusB;
`NOR: BusW <= ~(BusA | BusB);
`SRA: BusW <= BusB >>> BusA[4:0];
`LUI: BusW <= {BusB[15:0],16'b0};
// Set Less than unsigned - Set Bus B to 1 if BusA is less than BusB
`SLTU:
begin
if(BusA[31] == 0 && BusB[31] == 0)
BusW <= (BusA < BusB)? 1 : 0;
else if(BusA[31] == 1 && BusB[31] == 1)
BusW <= ((~BusA) + 1 < (~BusB) + 1)? 1 : 0;
else if(BusA[31] == 1 && BusB[31] == 0)
BusW <= ((~BusA) + 1 < BusB)? 1 : 0;
else if (BusA[31] == 0 && BusB[31] == 1)
BusW <= (BusA < (~BusB) + 1)? 1 : 0;
else
BusW <= 0;
end
default: BusW <= 0;
endcase
// Output bit zero =1 if BusW is 0
Zero <= (BusA == BusB)?1'b1:1'b0;
end
endmodule

```

## 2. ALUCONTROL

```

`timescale 1ns / 1ps
// Module Name: ALUControl
// Define R-Type Inst. function codes
`define SLLFunc 6'b000000
`define SRLFunc 6'b000010
`define SRAFunc 6'b000011
`define ADDFunc 6'b100000
`define ADDUFunc 6'b100001
`define SUBFunc 6'b100010
`define SUBUFunc 6'b100011
`define ANDFunc 6'b100100
`define ORFunc 6'b100101
`define XORFunc 6'b100110
`define NORFunc 6'b100111
`define SLTFunc 6'b101010
`define SLTUFunc 6'b101011
`define JRFunc 6'b001000
// Define ALU operation codes
`define AND 4'b0000
`define OR 4'b0001
`define ADD 4'b0010
`define SLL 4'b0011

```

```

`define SRL 4'b0100
`define SUB 4'b0110
`define SLT 4'b0111
`define ADDU 4'b1000
`define SUBU 4'b1001
`define XOR 4'b1010
`define SLTU 4'b1011
`define NOR 4'b1100
`define SRA 4'b1101
`define LUI 4'b1110
module ALUControl(ALUCtrl, ALUOp, FuncCode);
// Define the inputs and the outputs
input [3:0] ALUOp;
input [5:0] FuncCode;
output reg [3:0] ALUCtrl;
always@(*)
begin
// if ALUOp != 1111
if (ALUOp != 4'b1111)
ALUCtrl = ALUOp;
else
case (FuncCode)
// case statment to check FuncCode
`JRFunc: ALUCtrl = `ADD;
`SLLFunc: ALUCtrl = `SLL;
`SRLFunc: ALUCtrl = `SRL;
`SRAFunc: ALUCtrl = `SRA;
`ADDFunc: ALUCtrl = `ADD;
`ADDUFunc: ALUCtrl = `ADDU;
`SUBFunc: ALUCtrl = `SUB;
`SUBUFunc: ALUCtrl = `SUBU;
`ANDFunc: ALUCtrl = `AND;
`ORFunc: ALUCtrl = `OR;
`XORFunc: ALUCtrl = `XOR;
`NORFunc: ALUCtrl = `NOR;
`SLTFunc: ALUCtrl = `SLT;
`SLTUFunc: ALUCtrl = `SLTU;
default: ALUCtrl = 0;
endcase
end
endmodule

```

### 3. Data Memory

```

`timescale 1ns / 1ps
module DataMemory( ReadData, Address, WriteData, MemoryRead,
MemoryWrite,
Clock);
// Define Inputs and output
input MemoryRead, MemoryWrite, Clock;
input [5:0]Address;
input [31:0]WriteData;
output reg [31:0]ReadData;
// Define DataMemory
reg [31:0] DataMemory[63:0];
// Read

```

```

always @(posedge Clock)
if (MemoryRead == 1)
ReadData <= DataMemory[Address];
// Write
always @(negedge Clock)
if (MemoryWrite == 1)
DataMemory[Address] <= WriteData;
Endmodule

```

#### 4. Forwarding Unit

```

`timescale 1ns / 1ps
module ForwardingUnit(UseShamt , UseImmed , ID_Rs , ID_Rt , EX_Rw ,
MEM_Rw,
EX_RegWrite ,
MEM_RegWrite , ALUOpCtrlA , ALUOpCtrlB , DataMemForwardCtrl_EX ,
DataMemForwardCtrl_MEM);
// Define inputs and outputs
input UseShamt , UseImmed ;
input [4:0] ID_Rs , ID_Rt , EX_Rw , MEM_Rw;
input EX_RegWrite , MEM_RegWrite ;
output reg [1:0] ALUOpCtrlA ;
output reg [1:0] ALUOpCtrlB ;
output reg DataMemForwardCtrl_EX , DataMemForwardCtrl_MEM ;
// This is a combinatorial block
always @(*)
begin
// Control Signal definitions:
if(UseShamt == 1)
ALUOpCtrlA <= 2'b00;
else if (ID_Rs == EX_Rw & EX_RegWrite == 1 & EX_Rw != 0)
ALUOpCtrlA <= 2'b10;
else if(ID_Rs == MEM_Rw & MEM_RegWrite == 1 & MEM_Rw != 0)
ALUOpCtrlA <= 2'b01;
else
ALUOpCtrlA <= 2'b11;
// I-type instruction
if(UseImmed == 1)
ALUOpCtrlB <= 2'b00;
else if (ID_Rt == EX_Rw & EX_RegWrite == 1 & EX_Rw != 0)
ALUOpCtrlB <= 2'b10;
else if(ID_Rt == MEM_Rw & MEM_RegWrite == 1 & MEM_Rw != 0)
ALUOpCtrlB <= 2'b01;
else
ALUOpCtrlB <= 2'b11;
// load value is used by previous instruction
if (EX_RegWrite == 1 & ID_Rt == EX_Rw & EX_Rw != 0)
begin
DataMemForwardCtrl_EX <= 1'b0;
DataMemForwardCtrl_MEM <= 1'b1;
end
// if load value before the previous instruction
else if(MEM_RegWrite == 1 & ID_Rt == MEM_Rw & MEM_Rw != 0)
begin
DataMemForwardCtrl_EX <= 1'b1;
DataMemForwardCtrl_MEM <= 1'b0;
end
end

```

```

end
else
// If no data dependancy
begin
DataMemForwardCtrl_EX <= 1'b0;
DataMemForwardCtrl_MEM <= 1'b0;
end
end
endmodule

```

## 5. Hazard Unit

```

// 00 - Normal State, 01 - Branch, 11 - Jump
`timescale 1ns / 1ps
module HazardUnit(IF_write, PC_write, bubble, addrSel, Jump, Branch,
ALUZero, memReadEX, currRs, currRt, prevRt, UseShamt, UseImmed , Clk ,
Rst,
Jr, EX_RegWrite, MEM_RegWrite, EX_Rw, MEM_Rw);
// Input and output ports
output reg IF_write, PC_write, bubble;
output reg [1:0] addrSel;
input Jump , Branch , ALUZero , memReadEX , Clk , Rst, Jr,
EX_RegWrite, MEM_RegWrite ;
input [4:0] EX_Rw, MEM_Rw;
input UseShamt , UseImmed ;
input [4:0] currRs, currRt, prevRt;
wire LdHazard, JrHazard;
reg [1:0] state, Next_State;
// Logic for detecting load hazard
assign LdHazard = (memReadEX == 1) & (prevRt != 0) & (((currRs == prevRt
||
currRt == prevRt) & UseShamt == 0 & UseImmed == 0) || (UseShamt == 1 &
currRs
== prevRt) || (UseImmed == 1 & currRs == prevRt ))?1:0 ;
// Check Jr_hazard
assign JrHazard = ((Jr == 1) && ((EX_Rw == currRs && EX_RegWrite ==
1)|| (MEM_Rw == currRs && MEM_RegWrite == 1))) ? 1:0;
always @(*) begin
// Check for hazard
case(state)
2'b00: begin
// Load Hazard
if ((LdHazard == 1) || (JrHazard == 1)) begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b001;
addrSel <= 2'b00;
end
// Branch
else if (Branch == 1) begin
if (ALUZero == 1) begin
Next_State <= 2'b01;
{IF_write, PC_write, bubble} <=
3'b011;
addrSel <= 2'b10;
end
else begin
// Branch NOT taken

```

```

Next_State <= 2'b00;
{IF_write, PC_write, bubble} <=
3'b110;
addrSel <= 2'b00;
end
end
// Jump Hazard
else if (Jump == 1) begin
Next_State <= 2'b11;
{IF_write, PC_write, bubble} <= 3'b010;
addrSel <= 2'b01;
end
// NO Hazard
else begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b110;
addrSel <= 2'b00;
end
end
// Branch
2'b01: begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b110;
addrSel <= 2'b00;
end
// Jump
2'b11: begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b111;
addrSel <= 2'b00;
end
default: begin
Next_State <= 2'b00;
{IF_write, PC_write, bubble} <= 3'b110;
addrSel <= 2'b00;
end
endcase
end
// State is changed in negedge of the clock
always @(negedge Clk) begin
// If reset = 0, then keep state at 0
if(Rst == 0)
state <= 2'b00;
else
state <= Next_State;
end
endmodule

```

## 6. Pipeline Control Unit:

```

`timescale 1ns / 1ps
// Define Opcodes
`define RTYPEOPCODE 6'b000000
`define LWOPCODE 6'b100011
`define SWOPCODE 6'b101011
`define BEQOPCODE 6'b000100

```

```

`define JOPCODE 6'b000010
`define ORIOPCODE 6'b001101
`define ADDIOPCODE 6'b001000
`define ADDIUOPCODE 6'b001001
`define ANDIOPCODE 6'b001100
`define LUIOPCODE 6'b001111
`define SLTIOPCODE 6'b001010
`define SLTIUOPCODE 6'b001011
`define XORIOPCODE 6'b001110
`define JALOPCODE 6'b000011
// Define Function codes (used for R-type instructions)
`define SLLFunc 6'b000000
`define SRLFunc 6'b000010
`define SRAFunc 6'b000011
`define ADDFunc 6'b100000
`define ADDUFunc 6'b100001
`define SUBFunc 6'b100010
`define SUBUFunc 6'b100011
`define ANDFunc 6'b100100
`define ORFunc 6'b100101
`define XORFunc 6'b100110
`define NORFunc 6'b100111
`define SLTFunc 6'b101010
`define SLTUFunc 6'b101011
`define JRFunc 6'b001000
// Define ALU operation codes
`define AND 4'b0000
`define OR 4'b0001
`define ADD 4'b0010
`define SLL 4'b0011
`define SRL 4'b0100
`define SUB 4'b0110
`define SLT 4'b0111
`define ADDU 4'b1000
`define SUBU 4'b1001
`define XOR 4'b1010
`define SLTU 4'b1011
`define NOR 4'b1100
`define SRA 4'b1101
`define LUI 4'b1110
module ControlUnit(UseShamt, UseImmed, RegDst, MemToReg, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend, Jr, Jal, ALUOp, Opcode,
Func);
// Define inputs and outputs
input [5:0] Opcode;
input [5:0] Func;
output reg UseShamt, UseImmed;
output reg [3:0] ALUOp;
output reg RegDst, MemToReg, RegWrite, MemRead, MemWrite, Branch, Jump,
Jr,
Jal, SignExtend;
always @(*)
begin
case(Opcode)
`RTYPEOPCODE: begin
// R-Type instruction
ALUOp <= 4'b1111;

```

```

case(Func)
// Shift Instruction
`SLLFunc: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} <= 10'b110010000x;
{Jr, Jal} <= 2'b00;
end
// Shift Instruction
`SRLFunc: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} <= 10'b110010000x;
{Jr, Jal} <= 2'b00;
end
// Shift Instruction
`SRAFunc: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} <= 10'b110010000x;
{Jr, Jal} <= 2'b00;
end
`JRFunc: begin
{RegDst, UseShamt, UseImmed,
MemToReg, RegWrite, MemRead, MemWrite, Branch, Jump, SignExtend}
<= 10'b0001000010;
{Jr, Jal} <= 2'b10;
end
// R type instruction that are not shift
default: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend} <= 10'b100010000x;
{Jr, Jal} <= 2'b00;
end
endcase
end
// Load opcode
`LWOPCODE: begin
ALUOp <= `ADD;
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <= 10'b0011110001;
{Jr, Jal} <= 2'b00;
end
// Store word opcode
`SWOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <= 10'bx01x001001;
ALUOp <= `ADD;
{Jr, Jal} <= 2'b00;
end
// Branch if equal opcode
`BEQOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <= 10'bx00x000101;
ALUOp <= `SUB;
{Jr, Jal} <= 2'b00;
end
// Jump opcode
`JOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,

```

```

Branch, Jump, SignExtend} <=10'bx00x00001x;
ALUOp <=4'bxxxx;
{Jr, Jal} <=2'b00;
end
// Or immediate opcode
`ORIOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100000;
ALUOp <=`OR;
{Jr, Jal} <=2'b00;
end
// Add immediate opcode
`ADDIOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100001;
ALUOp <=`ADD;
{Jr, Jal} <=2'b00;
end
// Add immediate unsigned opcode
`ADDIUOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100000;
ALUOp <=`ADDU;
{Jr, Jal} <=2'b00;
end
// AND bitwise with immediate opcode
`ANDIOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100000;
ALUOp <=`AND;
{Jr, Jal} <=2'b00;
end
// Load Upper Immediate opcode
`LUIOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100000;
ALUOp <=`LUI;
{Jr, Jal} <=2'b00;
end
// Set less than Immediate opcode
`SLTIOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100001;
ALUOp <=`SLT;
{Jr, Jal} <=2'b00;
end
// Set less than Immediate unsigned opcode
`SLTIUOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100000;
ALUOp <=`SLTU;
{Jr, Jal} <=2'b00;
end
// XOR bitwise Immediate opcode
`XORIOPCODE: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0010100000;

```



```

ALUOp <=`XOR;
{Jr, Jal} <=2'b00;
end
`JALOPCODE:begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead,
MemWrite, Branch, Jump, SignExtend} <=10'b0000100010;
ALUOp <=4'bxxxx;
{Jr, Jal} <=2'b01;
end
default: begin
{RegDst, UseShamt, UseImmed, MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend} <=10'b0000000000;
ALUOp <=4'b0000;
{Jr, Jal} <=2'b00;
end
endcase
end
endmodule

```

## 7. Pipeline Process Unit

```

`timescale 1ns / 1ps
module PipelinedProc(CLK, Reset_L, PC, startPC, dMemOut);
// Define the inputs and outputs
input CLK, Reset_L;
input [31:0] startPC;
output [31:0] dMemOut;
output reg [31:0] PC;
// Define the wires for PC calculations
reg [31:0] PCnext;
wire [31:0] IF_Instruction;
// Define the wires for control signals
wire SignExtend;
// Define wires for ALU usage
reg [31:0] ALUInputA;
reg [31:0] ALUInputB;
wire [3:0] ALUCtrl;
wire [31:0] BusW;
wire [31:0] ID_SignExtendOut;
wire [31:0] IF_PCnew;
wire IF_write, PC_write, bubble;
wire [31:0] EX_BusB_BusW;
wire [31:0] DataMem_DataIn;
wire [1:0] addrSel;
// Pipeline Definition
// Pipeline Registers: IF_ID
// Input Instruction memory DataOut
reg [31:0] IF_ID_PCnew;
reg [31:0] IF_ID_Instruction;
wire [31:0] ID_Instruction = IF_ID_Instruction;
wire [31:0] ID_PCnew = IF_ID_PCnew;
// Pipeline Registers: ID_EX
// Control Signals
reg ID_EX_RegDst, ID_EX_RegWrite, ID_EX_MemRead, ID_EX_MemWrite,
ID_EX_MemToReg, ID_EX_Jal, ID_EX_Branch;
reg [3:0] ID_EX_ALUOp;

```

```

wire ID_RegDst, ID_RegWrite, ID_MemRead, ID_MemWrite, ID_Branch,
ID_Jump,
ID_MemToReg, ID_Jr, ID_Jal;
wire [3:0] ID_ALUOp;
wire EX_Branch;
wire EX_RegDst, EX_RegWrite, EX_MemRead, EX_MemWrite, EX_MemToReg,
EX_Jal;
wire [3:0] EX_ALUOp;
assign {EX_MemToReg, EX_RegDst, EX_RegWrite, EX_MemRead, EX_MemWrite,
EX_ALUOp, EX_Jal, EX_Branch} =
{ID_EX_MemToReg, ID_EX_RegDst, ID_EX_RegWrite, ID_EX_MemRead,
ID_EX_MemWrite,
ID_EX_ALUOp, ID_EX_Jal, ID_EX_Branch};
// Forwarding Unit Signals
reg [1:0] ID_EX_ALUOpCtrlA, ID_EX_ALUOpCtrlB ;
reg ID_EX_DataMemForwardCtrl_EX , ID_EX_DataMemForwardCtrl_MEM ;
wire [1:0] ID_ALUOpCtrlA, ID_ALUOpCtrlB ;
wire ID_DataMemForwardCtrl_EX , ID_DataMemForwardCtrl_MEM ;
wire [1:0] EX_ALUOpCtrlA, EX_ALUOpCtrlB ;
wire EX_DataMemForwardCtrl_EX , EX_DataMemForwardCtrl_MEM ;
assign {EX_ALUOpCtrlA, EX_ALUOpCtrlB, EX_DataMemForwardCtrl_EX ,
EX_DataMemForwardCtrl_MEM} =
{ID_EX_ALUOpCtrlA, ID_EX_ALUOpCtrlB, ID_EX_DataMemForwardCtrl_EX ,
ID_EX_DataMemForwardCtrl_MEM};
// Data
reg [20:0] ID_EX_Instruction;
(* keep = "true"*) reg [31:0] ID_EX_SignExtendOut;
reg [31:0] ID_EX_BusB, ID_EX_BusA, ID_EX_PCnew;
wire [31:0] ID_BusA, ID_BusB;
wire [20:0] EX_Instruction;
wire [31:0] EX_SignExtendOut, EX_BusB, EX_BusA, EX_PCnew;
assign {EX_Instruction, EX_SignExtendOut, EX_BusB, EX_BusA, EX_PCnew} =
{ID_EX_Instruction, ID_EX_SignExtendOut, ID_EX_BusB, ID_EX_BusA,
ID_EX_PCnew}
;
// Pipeline Registers: EX_MEM
// Control unit Signals
reg EX_MEM_MemToReg, EX_MEM_RegWrite, EX_MEM_MemRead, EX_MEM_MemWrite;
wire MEM_MemToReg, MEM_RegWrite, MEM_MemRead, MEM_MemWrite;
assign {MEM_MemToReg, MEM_RegWrite, MEM_MemRead, MEM_MemWrite} =
{EX_MEM_MemToReg, EX_MEM_RegWrite, EX_MEM_MemRead, EX_MEM_MemWrite};
// Forwarding Signals
reg EX_MEM_DataMemForwardCtrl_MEM;
wire MEM_DataMemForwardCtrl_MEM;
assign MEM_DataMemForwardCtrl_MEM = EX_MEM_DataMemForwardCtrl_MEM;
// Data
reg [31:0] EX_MEM_ALUOut;
wire [31:0] EX_ALUOut, ALUOut;
assign EX_ALUOut = (EX_Jal == 1)? EX_PCnew:ALUOut;
wire [31:0] MEM_ALUOut = EX_MEM_ALUOut;
reg [31:0] EX_MEM_BusB_BusW;
wire [31:0] MEM_BusB_BusW = EX_MEM_BusB_BusW;
reg [4:0] EX_MEM_Rw;
wire [4:0] MEM_Rw = EX_MEM_Rw;
wire [4:0] EX_Rw;
//Pipeline Registers: MEM_WB
// Control unit Signals

```

```

reg MEM_WB_MemToReg, MEM_WB_RegWrite;
wire WB_MemToReg, WB_RegWrite;
assign {WB_MemToReg, WB_RegWrite} = {MEM_WB_MemToReg, MEM_WB_RegWrite};
// Data
wire [31:0]MEM_DataMem_DataOut;
reg [31:0]MEM_WB_DataMem_DataOut;
wire [31:0] WB_DataMem_DataOut = MEM_WB_DataMem_DataOut;
reg [31:0] MEM_WB_ALUOut;
wire [31:0] WB_ALUOut = MEM_WB_ALUOut;
reg [4:0] MEM_WB_Rw;
wire [4:0] WB_Rw = MEM_WB_Rw;
//INSTANTIATING MODULES
// Call Intruction Memory
InstructionMemory InstructionMemory(
    .Data(IF_Instruction),
    .Address(PC));
// Call the control Unit which would create the signal controls
depending on the instruction
ControlUnit ControlUnit(
    .MemToReg(ID_MemToReg),
    .RegDst(ID_RegDst),
    .UseShamt(ID_UseShamt),
    .UseImmed(ID_UseImmed),
    .RegWrite(ID_RegWrite),
    .MemRead(ID_MemRead),
    .MemWrite(ID_MemWrite),
    .Branch(ID_Branch),
    .Jump(ID_Jump),
    .Jr(ID_Jr),
    .Jal(ID_Jal),
    .SignExtend(SignExtend),
    .ALUOp(ID_ALUOp),
    .Opcode(ID_Instruction[31:26]),
    .Func(ID_Instruction[5:0]));
// Call Hazard Unit
wire Zero;
HazardUnit dhiraj_hu(
    .IF_write(IF_write),
    .PC_write(PC_write),
    .bubble(bubble),
    .addrSel(addrSel),
    .Jump(ID_Jump),
    .Branch(EX_Branch),
    .ALUZero(Zero),
    .memReadEX(EX_MemRead),
    .currRs(ID_Instruction[25:21]),
    .currRt(ID_Instruction[20:16]),
    .prevRt(EX_Instruction[20:16]),
    .UseShamt(ID_UseShamt),
    .UseImmed(ID_UseImmed),
    .Clk(CLK),
    .Jr(ID_Jr),
    .EX_RegWrite(EX_RegWrite),
    .MEM_RegWrite(MEM_RegWrite),
    .EX_Rw(EX_Rw),
    .MEM_Rw(MEM_Rw),
    .Rst(Reset_L));

```

```

// Call the register File which contains 31 registers in MIPS
architecture
RegisterFile dhiraj_rf(
.BusA(ID_BusA) ,
.BusB(ID_BusB) ,
.BusW(BusW) ,
.RA(ID_Instruction[25:21]),
.RB(ID_Instruction[20:16]),
.RW(WB_Rw) ,
.RegWr(WB_RegWrite) ,
.Clk(CLK));
// Call the sign extend module which would extend the 16 bit number to
32 bit number
SignExtender SE(
.imm16(ID_Instruction[15:0]),
.signExtImm(ID_SignExtendOut),
.signExtend(SignExtend));
// Call Forwarding unit
ForwardingUnit dhiraj_fu(
.UseShamt(ID_UseShamt) ,
.UseImmed (ID_UseImmed),
.ID_Rs(ID_Instruction[25:21]) ,
.ID_Rt(ID_Instruction[20:16]) ,
.EX_Rw (EX_Rw),
.MEM_Rw (MEM_Rw),
.EX_RegWrite (EX_RegWrite) ,
.MEM_RegWrite (MEM_RegWrite) ,
.ALUOpCtrlA(ID_ALUOpCtrlA) ,
.ALUOpCtrlB(ID_ALUOpCtrlB) ,
.DataMemForwardCtrl_EX(ID_DataMemForwardCtrl_EX) ,
.DataMemForwardCtrl_MEM(ID_DataMemForwardCtrl_MEM)
);
// Call the ALU control block which would create appropriate control for
ALU depending on ALUopcode and the Function code
ALUControl ALUControl(
.ALUCtrl(ALUCtrl),
.ALUOp(EX_ALUOp),
.FuncCode(EX_Instruction[5:0]));
// Call the ALU which would perform the operations in inputs depending
on ALU Control signal
ALU ALU(
.BusW(ALUOut),
.Zero(Zero),
.BusA(ALUInputA),
.BusB(ALUInputB),
.ALUCtrl(ALUCtrl));
// Call Data Memory
DataMemory DataMemory(
.ReadData(MEM_DataMem_DataOut),
.Address(MEM_ALUOut[5:0]),
.WriteData(DataMem_DataIn),
.MemoryRead(MEM_MemRead),
.MemoryWrite(MEM_MemWrite),
.Clock(CLK));
always @(*)
begin
//PC Operations

```

```

case(addrSel)
2'b00: PCnext <= IF_PCnew;
2'b01: PCnext <= (ID_Jr==1)?
ID_BusA:{ID_PCnew[31:28],ID_Instruction[25:0],2'b00};
2'b10: PCnext <= (EX_SignExtendOut<<2) + EX_PCnew;
default:
PCnext <= IF_PCnew;
endcase
//Instantiating MUXes
// 4:1 MUX for ALU Input A
case(EX_ALUOpCtrlA)
2'b00: ALUInputA <= EX_Instruction[10:6];
2'b01: ALUInputA <= BusW;
2'b10: ALUInputA <= MEM_ALUOut;
2'b11: ALUInputA <= EX_BusA;
default: ALUInputA <= 32'b0;
endcase
// 4:1 MUX for ALU Input B
case(EX_ALUOpCtrlB)
2'b00: ALUInputB <= EX_SignExtendOut;
2'b01: ALUInputB <= BusW;
2'b10: ALUInputB <= MEM_ALUOut;
2'b11: ALUInputB <= EX_BusB;
default: ALUInputB <= 32'b0;
endcase
end
// MUX assignemnts and declarations
assign EX_Rw = (EX_Jal==0)?(EX_RegDst ==
0)?EX_Instruction[20:16]:EX_Instruction[15:11]):5'b11111;
assign EX_BusB_BusW = (EX_DataMemForwardCtrl_EX== 0) ? EX_BusB: BusW;
assign DataMem_DataIn = (MEM_DataMemForwardCtrl_MEM == 0)
?MEM_BusB_BusW:
BusW;
assign BusW = (WB_MemToReg == 1) ?WB_DataMem_DataOut:WB_ALUOut;
// Outputs
assign dMemOut = MEM_DataMem_DataOut;
// Assign the wires value according to the block diagram
assign IF_PCnew = PC + 4;
// At every negedge of clock or reset, assign the PC
always @(negedge CLK)// or negedge Reset_L)
begin
// Update The PC
if(Reset_L == 0)
PC <= startPC;
else if(PC_write == 1)
PC <= PCnext;
else
PC <= PC;
// Update the IF/ID Register
if (IF_write == 1)
begin
IF_ID_PCnew <= IF_PCnew;
IF_ID_Instruction[31:0] <= IF_Instruction[31:0];
end
else
begin
IF_ID_PCnew <= IF_ID_PCnew;

```

```

IF_ID_Instruction[31:0] <= IF_ID_Instruction[31:0];
end
// Update the ID_EX Register only if bubble is 0
if (bubble == 1)
{ID_EX_RegDst, ID_EX_RegWrite, ID_EX_MemRead, ID_EX_MemWrite,
ID_EX_Jal, ID_EX_MemToReg, ID_EX_ALUOp} <= 12'b000000000000;
else
begin
{ID_EX_RegDst, ID_EX_RegWrite, ID_EX_MemRead, ID_EX_MemWrite,
ID_EX_MemToReg, ID_EX_Jal, ID_EX_ALUOp} <=
{ID_RegDst, ID_RegWrite, ID_MemRead, ID_MemWrite, ID_MemToReg,
ID_Jal, ID_ALUOp};
end
{ID_EX_ALUOpCtrlA, ID_EX_ALUOpCtrlB, ID_EX_DataMemForwardCtrl_EX ,
ID_EX_DataMemForwardCtrl_MEM} <=
{ID_ALUOpCtrlA, ID_ALUOpCtrlB, ID_DataMemForwardCtrl_EX ,
ID_DataMemForwardCtrl_MEM};
{ID_EX_Instruction, ID_EX_SignExtendOut, ID_EX_BusB, ID_EX_BusA,
ID_EX_PCnew}<=
{ID_Instruction[20:0], ID_SignExtendOut, ID_BusB, ID_BusA,
ID_PCnew};
ID_EX_Branch <= ID_Branch;
// Update the EX_MEM Register
{EX_MEM_MemToReg, EX_MEM_RegWrite, EX_MEM_MemRead, EX_MEM_MemWrite}
<=
{EX_MemToReg, EX_RegWrite, EX_MemRead, EX_MemWrite};
EX_MEM_DataMemForwardCtrl_MEM <= EX_DataMemForwardCtrl_MEM;
EX_MEM_ALUOut <= EX_ALUOut;
EX_MEM_BusB_BusW <= EX_BusB_BusW;
EX_MEM_Rw <= EX_Rw;
// Update the MEM_WB Register
{MEM_WB_MemToReg, MEM_WB_RegWrite} <= {MEM_MemToReg, MEM_RegWrite} ;
MEM_WB_DataMem_DataOut <= MEM_DataMem_DataOut;
MEM_WB_ALUOut <= MEM_ALUOut;
MEM_WB_Rw <= MEM_Rw;
end
endmodule

```

## 8. Register File:

```

`timescale 1ns / 1ps
// RegisterFile
module RegisterFile(BusA , BusB , BusW , RA, RB, RW, RegWr , Clk);
// Define Inputs, Outputs and 32*32 RegisterMemory
input [4:0]RA;
input [4:0]RB;
input [4:0]RW;
input [31:0]BusW;
input RegWr, Clk;
output [31:0]BusA;
output [31:0]BusB;
reg [31:0] RegisterMemory[31:0];
// Read continuously from the register memory
assign BusB = (RB==0)?0:RegisterMemory[RB];
assign BusA = (RA==0)?0:RegisterMemory[RA];
// Write posedge to the MEM

```

```

always @(posedge Clk)
if (RegWr == 1 && RW!=0)
RegisterMemory[RW] <= BusW;
Endmodule

```

## 9. Sign Extender

```

`timescale 1ns / 1ps
module SignExtender(signExtImm, signExtend, imm16);
output [31:0] signExtImm;
input [15:0] imm16;
input signExtend;
assign signExtImm = signExtend? ({16{imm16[15]}}, imm16[15:0]}) :
{16'b0,imm16[15:0]};
Endmodule

```

{Instruction memory and testbench are not included here as they were provided by the TA and available on piazza}

### Results:

launch\_simulation: Time (s): cpu = 00:00:10 ; elapsed = 00:00:12 . Memory (MB): peak = 6140.551 ; gain = 53.05  
run all

```

Results of Program 1 passed
Results of Program 2 passed
Result 1 of Program 3 passed
Result 2 of Program 3 passed
Result 3 of Program 3 passed
Result 4 of Program 3 passed
Result 5 of Program 3 passed
Result 6 of Program 3 passed
Result 7 of Program 3 passed
Result 8 of Program 3 passed
Result 9 of Program 3 passed
Result 10 of Program 3 passed
Result 11 of Program 3 passed
Result 12 of Program 3 passed
Result 1 of Program 4 passed
Result 2 of Program 4 passed
Result 3 of Program 4 passed
Result 4 of Program 4 passed

```



All tests passed

\$finish called at time : 18361 ns : File "/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/Pipe1  
run: Time (s): cpu = 00:00:11 ; elapsed = 00:00:07 . Memory (MB): peak = 6148.004 ; gain = 7.453 ; free physic

### Project Settings

Project name: mips\_w\_bp  
Project location: /home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/mip  
Product family: Virtex-7  
Project part: xc7vx485tffg1157-1  
Top module name: PipelinedProc

### Synthesis

Status:  Complete  
Messages:  1 warning  
Part: xc7vx485tffg1157-1  
Strategy: [Vivado Synthesis Defaults](#)

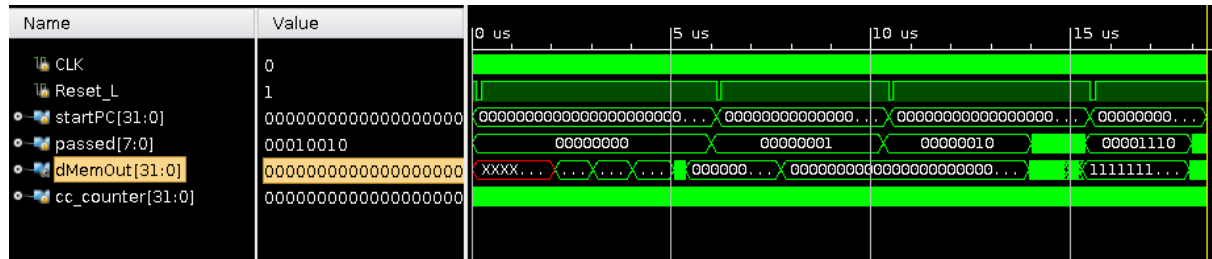
### Implementation

Status: Not started  
Messages: No errors c  
Part: xc7vx485tf  
Strategy: [Vivado Imp](#)  
Incremental compile: [None](#)

```

-----
Clock   Waveform(ns)      Period(ns)      Frequency (MHz)
-----
clk2    {0.000 4.150}      8.300          120.482

```



## Synthesis Report:

```

#-----
# Vivado v2015.2 (64-bit)
# SW Build 1266856 on Fri Jun 26 16:35:25 MDT 2015
# IP Build 1264090 on Wed Jun 24 14:22:01 MDT 2015
# Start of session at: Thu Nov 28 04:20:24 2019
# Process ID: 13808
# Log file: /home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch
predictor/mips_w_bp/mips_w_bp.runs/synth_1/PipelinedProc.vds
# Journal file: /home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch
predictor/mips_w_bp/mips_w_bp.runs/synth_1/vivado.jou
#-----
source PipelinedProc.tcl -notrace
Command: synth_design -top PipelinedProc -part xc7vx485tffg1157-1
Starting synth_design
Attempting to get a license for feature 'Synthesis' and/or device 'xc7vx485t'
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7vx485t'
INFO: [Common 17-1223] The version limit for your license is '2017.12' and will expire in
-697 days. A version limit expiration means that, although you may be able to continue
to use the current version of tools or IP with this license, you will not be eligible for any
updates or new releases.

-----
Starting RTL Elaboration : Time (s): cpu = 00:00:05 ; elapsed = 00:00:06 . Memory (MB):
peak = 1081.996 ; gain = 154.520 ; free physical = 393729 ; free virtual = 537981

-----
INFO: [Synth 8-638] synthesizing module 'PipelinedProc'
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch
predictor/pipeline_process.v:2]

```



INFO: [Synth 8-5534] Detected attribute (\* keep = "true" \*)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch  
predictor/pipeline\_process.v:59]  
INFO: [Synth 8-638] synthesizing module 'InstructionMemory'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch  
predictor/InstructionMemory.v:8]  
    Parameter T\_rd bound to: 20 - type: integer  
    Parameter MemSize bound to: 40 - type: integer  
INFO: [Synth 8-256] done synthesizing module 'InstructionMemory' (1#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch  
predictor/InstructionMemory.v:8]  
INFO: [Synth 8-638] synthesizing module 'ControlUnit'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/pipe\_control.v:47]  
INFO: [Synth 8-256] done synthesizing module 'ControlUnit' (2#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/pipe\_control.v:47]  
INFO: [Synth 8-638] synthesizing module 'HazardUnit'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/hazard.v:3]  
INFO: [Synth 8-256] done synthesizing module 'HazardUnit' (3#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/hazard.v:3]  
INFO: [Synth 8-638] synthesizing module 'RegisterFile'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/register\_file.v:3]  
INFO: [Synth 8-256] done synthesizing module 'RegisterFile' (4#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/register\_file.v:3]  
INFO: [Synth 8-638] synthesizing module 'SignExtender'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/sign\_extender.v:2]  
INFO: [Synth 8-256] done synthesizing module 'SignExtender' (5#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/sign\_extender.v:2]  
INFO: [Synth 8-638] synthesizing module 'ForwardingUnit'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/forwarding.v:2]  
INFO: [Synth 8-256] done synthesizing module 'ForwardingUnit' (6#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/forwarding.v:2]  
INFO: [Synth 8-638] synthesizing module 'ALUControl'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/alu\_control.v:33]  
INFO: [Synth 8-256] done synthesizing module 'ALUControl' (7#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/alu\_control.v:33]  
INFO: [Synth 8-638] synthesizing module 'ALU' [/home/grads/d/dhirajkudva/ECEN  
651/LAB 7 with branch predictor/alu.v:17]  
INFO: [Synth 8-256] done synthesizing module 'ALU' (8#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/alu.v:17]  
INFO: [Synth 8-638] synthesizing module 'DataMemory'  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/data\_memory.v:2]  
INFO: [Synth 8-256] done synthesizing module 'DataMemory' (9#1)  
[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/data\_memory.v:2]

INFO: [Synth 8-226] default block is never used [/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/pipeline\_process.v:208]

INFO: [Synth 8-226] default block is never used [/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/pipeline\_process.v:216]

INFO: [Synth 8-256] done synthesizing module 'PipelinedProc' (10#1)

[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/pipeline\_process.v:2]

-----  
Finished RTL Elaboration : Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 . Memory (MB): peak = 1118.293 ; gain = 190.816 ; free physical = 393690 ; free virtual = 537942  
-----

#### Report Check Netlist:

+-----+-----+-----+-----+-----+-----+							
	Item	Errors	Warnings	Status	Description		
+-----+-----+-----+-----+-----+-----+							
1	multi_driven_nets		0	0	Passed	Multi driven nets	
+-----+-----+-----+-----+-----+-----+							

-----  
Finished RTL Optimization Phase 1 : Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 .  
Memory (MB): peak = 1118.293 ; gain = 190.816 ; free physical = 393691 ; free virtual = 537943  
-----

INFO: [Device 21-403] Loading part xc7vx485tffg1157-1

INFO: [Project 1-570] Preparing netlist for logic optimization

#### Processing XDC Constraints

Initializing timing engine

Parsing XDC File [/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/mips\_w\_bp/mips\_w\_bp.srscs/constrs\_1/new/clk2.xdc]

Finished Parsing XDC File [/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/mips\_w\_bp/mips\_w\_bp.srscs/constrs\_1/new/clk2.xdc]

Completed Processing XDC Constraints

INFO: [Project 1-111] Unisim Transformation Summary:

No Unisim elements were transformed.

Constraint Validation Runtime : Time (s): cpu = 00:00:00.01 ; elapsed = 00:00:00.01 .  
Memory (MB): peak = 1586.090 ; gain = 0.000 ; free physical = 393381 ; free virtual = 537633  
-----

Finished Constraint Validation : Time (s): cpu = 00:00:18 ; elapsed = 00:00:19 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393380 ; free virtual = 537632  
-----

-----  
Start Loading Part and Timing Information  
-----

Loading part: xc7vx485tffg1157-1  
-----

Finished Loading Part and Timing Information : Time (s): cpu = 00:00:18 ; elapsed = 00:00:19 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393380 ; free virtual = 537632  
-----  
-----

-----  
Start Applying 'set\_property' XDC Constraints  
-----

-----  
Finished applying 'set\_property' XDC Constraints : Time (s): cpu = 00:00:18 ; elapsed = 00:00:19 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393380 ; free virtual = 537632  
-----

INFO: [Synth 8-5546] ROM "UseShamt" won't be mapped to RAM because it is too sparse

ROM size is below threshold of ROM address width. It will be mapped to LUTs

INFO: [Synth 8-5546] ROM "RegWrite" won't be mapped to RAM because it is too sparse

INFO: [Synth 8-5546] ROM "Jump" won't be mapped to RAM because it is too sparse

INFO: [Synth 8-5546] ROM "MemRead" won't be mapped to RAM because it is too sparse

INFO: [Synth 8-5546] ROM "MemWrite" won't be mapped to RAM because it is too sparse

INFO: [Synth 8-5546] ROM "Branch" won't be mapped to RAM because it is too sparse

INFO: [Synth 8-5546] ROM "Jal" won't be mapped to RAM because it is too sparse

ROM size is below threshold of ROM address width. It will be mapped to LUTs

INFO: [Synth 8-3537] HDL ADVISOR - The operator resource <adder> is shared. To prevent sharing consider applying a KEEP on the inputs of the operator

[/home/grads/d/dhirajkudva/ECEN 651/LAB 7 with branch predictor/alu.v:26]

INFO: [Synth 8-5562] The signal DataMemory\_reg is implemented as block RAM but is better mapped onto distributed LUT RAM for the following reason(s): The \*depth (6 address bits)\* is shallow. Please use attribute (\* ram\_style = "distributed" \*) to instruct Vivado to infer distributed LUT RAM.  
-----

Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:19 ; elapsed = 00:00:20 .  
Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393370 ; free virtual = 537622  
-----

Report RTL Partitions:

+ +-----+ +-----+ +-----+ +

| |RTL Partition |Replication |Instances |

+-----+-----+-----+

+-----+-----+-----+

---

### Start RTL Component Statistics

---

#### Detailed RTL Component Info :

##### +---Adders :

3 Input 32 Bit Adders := 1

2 Input 32 Bit Adders := 2

##### +---XORs :

2 Input 32 Bit XORs := 1

##### +---Registers :

32 Bit Registers := 1

##### +---RAMs :

2K Bit RAMs := 1

1024 Bit RAMs := 1

##### +---Muxes :

137 Input 32 Bit Muxes := 1

2 Input 32 Bit Muxes := 13

15 Input 32 Bit Muxes := 2

4 Input 32 Bit Muxes := 3

2 Input 5 Bit Muxes := 2

15 Input 4 Bit Muxes := 1

16 Input 4 Bit Muxes := 1

15 Input 2 Bit Muxes := 4

4 Input 2 Bit Muxes := 2

2 Input 2 Bit Muxes := 11

3 Input 2 Bit Muxes := 2

2 Input 1 Bit Muxes := 7

5 Input 1 Bit Muxes := 3

15 Input 1 Bit Muxes := 6

3 Input 1 Bit Muxes := 3

4 Input 1 Bit Muxes := 4

---

### Finished RTL Component Statistics

---

---

### Start RTL Hierarchical Component Statistics

---

#### Hierarchical RTL Component report

Module PipelinedProc

#### Detailed RTL Component Info :

##### +---Adders :

```

        2 Input  32 Bit  Adders := 2
+---Muxes :
        2 Input  32 Bit  Muxes := 6
        4 Input  32 Bit  Muxes := 3
        2 Input  5 Bit   Muxes := 2
        2 Input  1 Bit   Muxes := 1
Module InstructionMemory
Detailed RTL Component Info :
+---Muxes :
        137 Input 32 Bit  Muxes := 1
Module ControlUnit
Detailed RTL Component Info :
+---Muxes :
        15 Input  4 Bit   Muxes := 1
        15 Input  2 Bit   Muxes := 4
        2 Input  1 Bit   Muxes := 3
        5 Input  1 Bit   Muxes := 3
        15 Input  1 Bit   Muxes := 6
        3 Input  1 Bit   Muxes := 3
        4 Input  1 Bit   Muxes := 1
Module HazardUnit
Detailed RTL Component Info :
+---Muxes :
        4 Input  2 Bit   Muxes := 2
        2 Input  2 Bit   Muxes := 2
        3 Input  2 Bit   Muxes := 2
        2 Input  1 Bit   Muxes := 3
        4 Input  1 Bit   Muxes := 3
Module RegisterFile
Detailed RTL Component Info :
+---RAMs :
        1024 Bit  RAMs := 1
+---Muxes :
        2 Input  32 Bit  Muxes := 2
Module SignExtender
Detailed RTL Component Info :
+---Muxes :
        2 Input  32 Bit  Muxes := 1
Module ForwardingUnit
Detailed RTL Component Info :
+---Muxes :
        2 Input  2 Bit   Muxes := 6
Module ALUControl
Detailed RTL Component Info :

```

+---Muxes :

16 Input 4 Bit Muxes := 1

Module ALU

Detailed RTL Component Info :

+---Adders :

3 Input 32 Bit Adders := 1

+---XORs :

2 Input 32 Bit XORs := 1

+---Muxes :

15 Input 32 Bit Muxes := 2

2 Input 32 Bit Muxes := 4

2 Input 2 Bit Muxes := 3

Module DataMemory

Detailed RTL Component Info :

+---Registers :

32 Bit Registers := 1

+---RAMs :

2K Bit RAMs := 1

-----  
Finished RTL Hierarchical Component Statistics  
-----  
-----

Start Part Resource Summary  
-----

Part Resources:

DSPs: 2800 (col length:140)

BRAMs: 2060 (col length: RAMB18 140 RAMB36 70)  
-----

Finished Part Resource Summary  
-----

Start Parallel Synthesis Optimization : Time (s): cpu = 00:00:19 ; elapsed = 00:00:21 .

Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393370 ; free virtual = 537622  
-----

Start Cross Boundary Optimization  
-----

INFO: [Synth 8-5544] ROM "BusW0" won't be mapped to Block RAM because address size (1) smaller than threshold (5)

INFO: [Synth 8-5544] ROM "BusW0" won't be mapped to Block RAM because address size (1) smaller than threshold (5)

INFO: [Synth 8-5544] ROM "BusW0" won't be mapped to Block RAM because address size (1) smaller than threshold (5)  
-----

Finished Cross Boundary Optimization : Time (s): cpu = 00:00:20 ; elapsed = 00:00:21 .  
Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393370 ; free virtual = 537622

Finished Parallel Reinference : Time (s): cpu = 00:00:20 ; elapsed = 00:00:21 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393370 ; free virtual = 537622

Report RTL Partitions:

RTL Partition	Replication	Instances
---------------	-------------	-----------

INFO: [Synth 8-5562] The signal DataMemory/DataMemory\_reg is implemented as block RAM but is better mapped onto distributed LUT RAM for the following reason(s): The \*depth (6 address bits)\* is shallow. Please use attribute (\* ram\_style = "distributed" \*) to instruct Vivado to infer distributed LUT RAM.

Start ROM, RAM, DSP and Shift Register Reporting

Block RAM:

Module Name	RTL Object	PORT A (Depth x Width)	W	R	PORT B (Depth x Width)	W	R	OUT_REG	RAMB18	RAMB36	Hierarchical Name
PipelinedProc	DataMemory/DataMemory_reg	64 x 32(READ_FIRST)	W		64 x 32(WRITE_FIRST)		R	Port A and B	1	0	PipelinedProc/extram__4

Note: The table shows the Block RAMs at the current stage of the synthesis flow. Some Block RAMs may be reimplemented as non Block RAM primitives later in the synthesis flow. Multiple instantiated Block RAMs are reported only once. "Hierarchical Name" reflects the Block RAM name as it appears in the hierarchical module and only part of it is displayed.

Distributed RAM:

Module Name	RTL Object	Inference	Size (Depth x Width)	Primitives
Hierarchical Name				

PipelinedProc	dhiraj_rf/RegisterMemory_reg	Implied	32 x 32		RAM32M x
12	PipelinedProc/ram__1				
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
-----+					

Note: The table shows the Distributed RAMs at the current stage of the synthesis flow. Some Distributed RAMs may be reimplemented as non Distributed RAM primitives later in the synthesis flow. Multiple instantiated RAMs are reported only once. "Hierarchical Name" reflects the Distributed RAM name as it appears in the hierarchical module and only part of it is displayed.

#### Finished ROM, RAM, DSP and Shift Register Reporting

INFO: [Synth 8-3333] propagating constant 0 across sequential element (\IF\_ID\_Instruction\_reg[30] )

WARNING: [Synth 8-3332] Sequential element (\IF\_ID\_Instruction\_reg[30] ) is unused and will be removed from module PipelinedProc.

#### Start Area Optimization

Finished Area Optimization : Time (s): cpu = 00:00:28 ; elapsed = 00:00:30 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393346 ; free virtual = 537598

Finished Parallel Area Optimization : Time (s): cpu = 00:00:28 ; elapsed = 00:00:30 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393346 ; free virtual = 537598

#### Report RTL Partitions:

+-----+	+-----+	+-----+
	RTL Partition	Replication   Instances

+-----+	+-----+	+-----+
---------	---------	---------

+-----+	+-----+	+-----+
---------	---------	---------

Finished Parallel Synthesis Optimization : Time (s): cpu = 00:00:28 ; elapsed = 00:00:30 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393346 ; free virtual = 537598

#### Start Timing Optimization

#### Start Applying XDC Timing Constraints



Finished Applying XDC Timing Constraints : Time (s): cpu = 00:00:38 ; elapsed = 00:00:39  
. Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393295 ; free virtual = 537547

-----  
INFO: [Synth 8-5563] The signal DataMemory/DataMemory\_reg is implemented as distributed LUT RAM for the following reason(s): The \*timing constraints\* suggest that the chosen mapping will yield better timing results.

INFO: [Synth 8-5563] The signal DataMemory/DataMemory\_reg is implemented as distributed LUT RAM for the following reason(s): The \*timing constraints\* suggest that the chosen mapping will yield better timing results.

-----  
Finished Timing Optimization : Time (s): cpu = 00:00:38 ; elapsed = 00:00:40 . Memory (MB): peak = 1586.090 ; gain = 658.613 ; free physical = 393293 ; free virtual = 537545

-----  
Report RTL Partitions:

++-----+-----+-----+
RTL Partition  Replication  Instances
++-----+-----+-----+
++-----+-----+-----+

-----  
Start Technology Mapping

-----  
Finished Technology Mapping : Time (s): cpu = 00:00:40 ; elapsed = 00:00:42 . Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

-----  
Report RTL Partitions:

++-----+-----+-----+
RTL Partition  Replication  Instances
++-----+-----+-----+
++-----+-----+-----+

-----  
Start IO Insertion

-----  
Start Final Netlist Cleanup

-----  
Finished Final Netlist Cleanup

Finished IO Insertion : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 . Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

-----

Report Check Netlist:

	Item	Errors	Warnings	Status	Description
1	multi_driven_nets	0	0	Passed	Multi driven nets

-----

Start Renaming Generated Instances

-----

Finished Renaming Generated Instances : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 . Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

-----

Report RTL Partitions:

RTL Partition	Replication	Instances

-----

Start Rebuilding User Hierarchy

-----

Finished Rebuilding User Hierarchy : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 . Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

-----

Start Renaming Generated Ports : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 . Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

-----

Finished Renaming Generated Ports : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 . Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

-----

Start Writing Synthesis Report

---

Report BlackBoxes:

BlackBox name	Instances

Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	59
3	LUT1	86
4	LUT2	65
5	LUT3	99
6	LUT4	258
7	LUT5	340
8	LUT6	443
9	MUXF7	67
10	RAM32M	12
11	RAM64X1S	32
12	FDRE	435
13	FDSE	5
14	IBUF	34
15	OBUF	64

Report Instance Areas:

	Instance	Module	Cells
1	top		2000
2	ALU	ALU	563
3	DataMemory	DataMemory	96
4	dhiraj_hu	HazardUnit	115
5	dhiraj_rf	RegisterFile	45

---

Finished Writing Synthesis Report : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 .  
Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

---

Synthesis finished with 0 errors, 0 critical warnings and 1 warnings.

Synthesis Optimization Runtime : Time (s): cpu = 00:00:29 ; elapsed = 00:00:29 . Memory (MB): peak = 1604.090 ; gain = 92.297 ; free physical = 393236 ; free virtual = 537488

Synthesis Optimization Complete : Time (s): cpu = 00:00:41 ; elapsed = 00:00:43 .

Memory (MB): peak = 1604.090 ; gain = 676.613 ; free physical = 393236 ; free virtual = 537488

INFO: [Project 1-571] Translating synthesized netlist

INFO: [Netlist 29-17] Analyzing 137 Unisim elements for replacement

INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds

INFO: [Project 1-570] Preparing netlist for logic optimization

INFO: [Opt 31-138] Pushed 2 inverter(s) to 440 load pin(s).

INFO: [Project 1-111] Unisim Transformation Summary:

A total of 44 instances were transformed.

RAM32M => RAM32M (RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMS32, RAMS32): 12 instances

RAM64X1S => RAM64X1S (inverted pins: WCLK) (RAMS64E): 32 instances

INFO: [Common 17-83] Releasing license: Synthesis

51 Infos, 1 Warnings, 0 Critical Warnings and 0 Errors encountered.

synth\_design completed successfully

synth\_design: Time (s): cpu = 00:00:40 ; elapsed = 00:00:41 . Memory (MB): peak = 1636.105 ; gain = 600.109 ; free physical = 393236 ; free virtual = 537488

report\_utilization: Time (s): cpu = 00:00:00.11 ; elapsed = 00:00:00.17 . Memory (MB): peak = 1668.129 ; gain = 0.000 ; free physical = 393235 ; free virtual = 537487

INFO: [Common 17-206] Exiting Vivado at Thu Nov 28 04:21:13 2019...

## Questions and Answers:

### Ans 1.

Yes, the clock rate reduced to 120.482 MHz. Before adding jal and jr instruction the clock rate in lab 6 was 125.00MHz.

Reason: The presence of additional mux between the output of ALU and EX/MEM register makes the process slower. This MUX selects PC+4 or ALUOut depending on the JAL signal.

Improvements: Jr instruction jumps to the address location which is available in the register specified. If the register is being written by the previous instruction at the execute or the memory stage then hazard is called as 'JR hazard.' In case of hazard we stall the processor until the hazard is resolved. Thus, in turn we lose certain number of cycles probably one or two if the previous to previous instruction or the previous instruction is writing to the same register respectively.

Solution: 1) We can use the FORWARDING technique to solve this issue. The data which is about to be written to the register can be forwarded from the EX or MEM stage to decode stage using forwarding unit, and use this address to jump to the required location. 2) To make the frequency better, we need to make the critical path faster. This can be made faster by a better and improved VLSI design changes like optimal number of buffers and using wider wires for low resistance.

### Ans 2

Post Branch Prediction.

Total number cycles for program 1 is 46 cycles (considering the nop instructions as well.)  
Total number of instructions = 34 ( including instructions outside the loop + instructions in loop )

Therefore CPI = 1.35

When compared to lab 6, the CPI got reduced from 1.44 to 1.35. Now, the average clock cycles required per instruction have been reduced. This is because number of stalls required when there is a branch and outcome is not taken has been reduced from one to zero. We are predicting branch not taken. Thus, if we predict branch not taken and it is correct then no cycles are lost. However, if the prediction is wrong, then we need to flush ID and EX pipeline registers and we lose 2 cycles like before implementing static branch hazard.

### Ans 3:

Considering the program with branch predictor we have the CPI of programs as follows:

Program 1: 1.35( as shown above)

Program 2:  $35/11 = 3.09$

Program 3:  $40/38 = 1.05$

Program 4:  $23/19 = 1.21$

Average:  $(1.35+3.09+1.05+1.21)/4 = 1.64$

CPI lab 6 average: 2.11

Explanation: The 4th program takes considerable amount of cycles per instructions to complete its execution because of high density of jump instructions and jr hazard which is occurring in it. Thus, average of CPI including 4th program goes up.

However, on using static branch prediction, the CPI of program 1 goes down, as we need not stall for branch not taken case. Thus, average CPI goes down as compared to that without static branch predictor.

#### **Ans 4:**

Hazard unit becomes less complex because two states for branch are no longer required. One state is enough for branch, thus reducing our hardware complexity, Number of states reduces from 4 to 3. Thus, number of bit lines required for state is still same which is 2.

However, now we are taking branch execute stage instead of decode stage. This means that there is one more register added in ID/EX register.

Thus, effectively total number of hardware remains almost equal with slight additions of less than 0.2% in number of logics and flipflops. However, improvement is significant. Thus, it is worth to have a static branch predictor.

#### **Ans 5:**

In program 1, there is an unconditional jump at the end of the loop, while at the start there is a branch instruction. This branch is taken if the condition is met. Thus, If the loop iterates  $n$  times, the branch is NOT taken for  $n$  times, and at  $n+1$  instance, the branch is taken. Therefore, probability that branch is not taken is very high.

Since the probability of not taking a branch is high, it is worth predicting that branch is not taken always. Using such a prediction, no cycles are lost for waiting for the branch to get resolved for  $n$  times, though it loses 2 cycles in  $n+1$  iteration.

If the number of loops increases, i.e. the number of times the branch not taken increases then the probability of branch not taken will also increase. As probability gets closer to our prediction, the prediction matches outcome a greater number of times. Therefore, the CPI decreases.

By quantitative analysis, the branch when not take takes only 1 cycle for that 1 instruction. Whereas the CPI is more than 1. Thus, as number of iterations increases the CPI will decrease.