

SHANGHAI JIAOTONG UNIVERSITY



计算机系统结构实验报告 – Lab6

仿真测试 & 下载验证

姓名：刘欣鹏

学号：516030910259

完成时间：2017/5/23

目录

[1. 概述 2](#_Toc24516)

[1.1 实验名称 3](#_Toc20488)

[1.2 实验目的 3](#_Toc13791)

[1.3实验范围 3](#_Toc14051)

[2. 实验描述 3](#_Toc22767)

[2.1 顶层模块Top 3](#_Toc3438)

[2.1.1模块描述 3](#_Toc16952)

[2.1.2 Top模块代码 5](#_Toc1652)

[2.2 Mem\_Data模块 9](#_Toc17828)

[2.3 Inst\_Memory模块 10](#_Toc2392)

[3.仿真 11](#_Toc2980)

[3.1 仿真代码 11](#_Toc26871)

[3.2 仿真波形 11](#_Toc32521)

[4. 下载验证 12](#_Toc25250)

[4.1 实验描述 12](#_Toc30363)

[4.2 模块改进 12](#_Toc11729)

[4.2.1 分频器模块 12](#_Toc24702)

[4.2.2 Top模块改进 12](#_Toc27259)

[4.3 管脚定义 17](#_Toc9635)

[4.4 实验结果 17](#_Toc27457)

[5. 实验心得 17](#_Toc28116)

# 1. 概述

## 1.1 实验名称

简单的类 MIPS 多周期流水化处理器实现

## 1.2 实验目的

理解CPU 的Pipeline，对Data Hazard, Branch Hazard 有初步认识。

## 1.3实验范围

本次实验将覆盖以下范围:

(1) ISE的使用

(2) Xilinx Spartan 3E实验板的使用

(3)使用VerilogHDL进行逻辑设计

(4)仿真测试、下载验证

# 2. 实验描述

## 2.1 顶层模块Top

### 2.1.1模块描述

顶层模块将类MIPS单周期处理器的各个组成部分联结起来，并在其中添加4级寄存器，形成完整的流水化处理器。

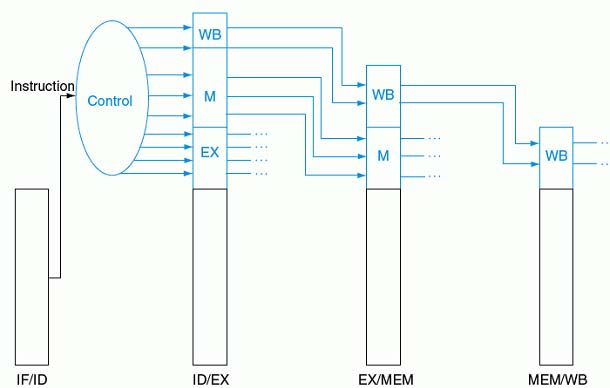


MIPS单周期流水化处理器原理图

将单周期 CPU 进行分割，插入4 级寄存器，将其分割为IF，ID，EX，M，WB五大部分：



其中 Control 的输出需要被保存下来，以供后续每级流水使用。如下所示：





### 2.1.2 Top模块代码

与实验5中的Top模块相比，本次实验主要是添加了4级流水线寄存器。在实现方面，没有对data hazard和control hazard进行处理。

**module top(**

**input clockin,**

**input reset,**

**input [2:0] sw,**

**output reg [7:0] LED,**

**output wire clock,**

**output wire [31:0] R0,**

**output wire [31:0] R1,**

**output wire [31:0] R2,**

**output wire [31:0] R3,**

**output wire [31:0] R4,**

**output wire [31:0] R5,**

**output wire [31:0] R6,**

**output wire [31:0] R7,**

**output wire [31:0] R8,**

**output reg [31:0] PC**

**);**

**// IF/ID**

**reg[31:0] S1\_PC\_ADD4;**

**reg[31:0] S1\_INST;**

**// ID/EX**

**reg S2\_REG\_WRITE, S2\_MEM\_TO\_REG; // WB**

**reg S2\_BRANCH, S2\_MEM\_READ, S2\_MEM\_WRITE; // M**

**reg S2\_REG\_DST, S2\_ALU\_SRC;**

**reg [1:0] S2\_ALU\_OP; // EX**

**reg[31:0] S2\_PC\_ADD4;**

**reg[31:0] S2\_READ\_DATA1;**

**reg[31:0] S2\_READ\_DATA2;**

**reg[31:0] S2\_SIGNEXT\_IMME;**

**reg[5:0] S2\_REG\_ADDR\_HIGH; //inst[20:16]**

**reg[5:0] S2\_REG\_ADDR\_LOW; //inst[15:11]**

**// EX/MEM**

**reg S3\_REG\_WRITE, S3\_MEM\_TO\_REG; // WB**

**reg S3\_BRANCH, S3\_MEM\_READ, S3\_MEM\_WRITE; // M**

**reg[31:0] S3\_BRANCH\_ADDR;**

**reg[31:0] S3\_ALU\_RES;**

**reg[31:0] S3\_ALU\_ZERO;**

**reg[31:0] S3\_MEM\_WRITE\_DATA; //read\_data2**

**reg[5:0] S3\_REG\_ADDR;**

**// MEM/WB**

**reg S4\_REG\_WRITE, S4\_MEM\_TO\_REG; // WB**

**reg[31:0] S4\_MEM\_READ\_DATA;**

**reg[31:0] S4\_ALU\_RES;**

**reg[5:0] S4\_REG\_ADDR;**

**// IF**

**wire PC\_SRC;**

**wire[31:0] IF\_INST;**

**// ID**

**wire ID\_REG\_DST;**

**wire ID\_ALU\_SRC;**

**wire ID\_MEM\_TO\_REG;**

**wire ID\_REG\_WRITE;**

**wire ID\_MEM\_READ;**

**wire ID\_MEM\_WRITE;**

**wire ID\_BRANCH;**

**wire[1:0] ID\_ALU\_OP;**

**wire[31:0] ID\_READ\_DATA1;**

**wire[31:0] ID\_READ\_DATA2;**

**wire[31:0] ID\_SIGNEXT\_IMME;**

**wire[31:0] MEM\_MEM\_READ\_DATA;**

**time\_divider mainTime(**

**.clock\_in(clockin),**

**.clock\_out(clock)**

**);**

**always @ (clock)**

**begin**

**case(sw[2:0])**

**3'b000: LED[2:0]<=R0[2:0];**

**3'b001: LED[2:0]<=R1[2:0];**

**3'b010: LED[2:0]<=R2[2:0];**

**3'b011: LED[2:0]<=R3[2:0];**

**3'b100: LED[2:0]<=R4[2:0];**

**3'b101: LED[2:0]<=R5[2:0];**

**3'b110: LED[2:0]<=R6[2:0];**

**3'b111: LED[3:0]<=R7[2:0];**

**endcase**

**LED[6]<=reset;**

**LED[5:3]<=sw[2:0];**

**end**

**inst\_memory mainRom(**

**.pc(PC),**

**.inst(IF\_INST),**

**.reset(reset)**

**);**

**ctr mainCtr(**

**.opCode(S1\_INST[31:26]),**

**.regDst(ID\_REG\_DST),**

**.aluSrc(ID\_ALU\_SRC),**

**.memToReg(ID\_MEM\_TO\_REG),**

**.regWrite(ID\_REG\_WRITE),**

**.memRead(ID\_MEM\_READ),**

**.memWrite(ID\_MEM\_WRITE),**

**.jump(JUMP),**

**.branch(ID\_BRANCH),**

**.aluOp(ID\_ALU\_OP)**

**);**

**register mainRegister(**

**.clock\_in(clock),**

**.readReg1(S1\_INST[25:21]),**

**.readReg2(S1\_INST[20:16]),**

**.writeReg(S4\_REG\_ADDR),**

**.regWrite(S4\_REG\_WRITE),**

**.writeData(S4\_MEM\_TO\_REG ? S4\_MEM\_READ\_DATA : S4\_ALU\_RES),**

**.reset(reset),**

**.readData1(ID\_READ\_DATA1),**

**.readData2(ID\_READ\_DATA2),**

**.reg0(R0),**

**.reg1(R1),**

**.reg2(R2),**

**.reg3(R3),**

**.reg4(R4),**

**.reg5(R5),**

**.reg6(R6),**

**.reg7(R7),**

**.reg8(R8)**

**);**

**signext mainSignext(**

**.inst(S1\_INST[15:0]),**

**.extend(ID\_SIGNEXT\_IMME)**

**);**

**wire[3:0] EX\_ALU\_CTR;**

**wire EX\_ALU\_ZERO;**

**wire[31:0] EX\_ALU\_RES;**

**AluCtr mainAluCtr(**

**.aluOp(S2\_ALU\_OP),**

**.funct(S2\_SIGNEXT\_IMME[5:0]),**

**.aluCtr(EX\_ALU\_CTR)**

**);**

**alu mainAlu(**

**.input1(S2\_READ\_DATA1),**

**.input2(S2\_ALU\_SRC ? S2\_SIGNEXT\_IMME : S2\_READ\_DATA2),**

**.aluCtr(EX\_ALU\_CTR),**

**.zero(EX\_ALU\_ZERO),**

**.aluRes(EX\_ALU\_RES)**

**);**

**data\_memory mainDataMemory(**

**.clock\_in(clock),**

**.address(S3\_ALU\_RES),**

**.writeData(S3\_MEM\_WRITE\_DATA),**

**.memRead(S3\_MEM\_READ),**

**.memWrite(S3\_MEM\_WRITE),**

**.readData(MEM\_MEM\_READ\_DATA),**

**.reset(reset)**

**);**

**assign PC\_SRC = S3\_ALU\_ZERO & S3\_BRANCH;**

**always@(posedge clock or posedge reset)**

**begin**

**if(reset)**

**begin**

**PC = 0;**

**// IF/ID**

**S1\_PC\_ADD4=0;**

**S1\_INST=32'b00000011111111111111100000100000;**

**// ID/EX**

**S2\_REG\_WRITE=0;**

**S2\_MEM\_TO\_REG=0;**

**S2\_BRANCH=0;**

**S2\_MEM\_READ=0;**

**S2\_MEM\_WRITE=0;**

**S2\_REG\_DST=0;**

**S2\_ALU\_SRC=0;**

**S2\_ALU\_OP=0;**

**S2\_PC\_ADD4=0;**

**S2\_READ\_DATA1=0;**

**S2\_READ\_DATA2=0;**

**S2\_SIGNEXT\_IMME=0;**

**S2\_REG\_ADDR\_HIGH=0; //inst[20:16]**

**S2\_REG\_ADDR\_LOW=0; //inst[15:11]**

**// EX/MEM**

**S3\_REG\_WRITE=0;**

**S3\_MEM\_TO\_REG=0; // WB**

**S3\_BRANCH=0;**

**S3\_MEM\_READ=0;**

**S3\_MEM\_WRITE=0; // M**

**S3\_BRANCH\_ADDR=0;**

**S3\_ALU\_RES=0;**

**S3\_ALU\_ZERO=0;**

**S3\_MEM\_WRITE\_DATA=0; //read\_data2**

**S3\_REG\_ADDR=0;**

**// MEM/WB**

**S4\_REG\_WRITE=0;**

**S4\_MEM\_TO\_REG=0; // WB**

**S4\_MEM\_READ\_DATA=0;**

**S4\_ALU\_RES=0;**

**S4\_REG\_ADDR=0;**

**end**

**else**

**begin**

**// MEM/WB**

**S4\_REG\_WRITE = S3\_REG\_WRITE;**

**S4\_MEM\_TO\_REG = S3\_MEM\_TO\_REG;**

**S4\_MEM\_READ\_DATA = MEM\_MEM\_READ\_DATA;**

**S4\_ALU\_RES = S3\_ALU\_RES;**

**S4\_REG\_ADDR = S3\_REG\_ADDR;**

**// PC**

**PC = PC\_SRC ? S3\_BRANCH\_ADDR: PC+4;**

**// EX/MEM**

**S3\_REG\_WRITE = S2\_REG\_WRITE;**

**S3\_MEM\_TO\_REG = S2\_MEM\_TO\_REG;**

**S3\_BRANCH = S2\_BRANCH;**

**S3\_MEM\_READ = S2\_MEM\_READ;**

**S3\_MEM\_WRITE = S2\_MEM\_WRITE;**

**S3\_BRANCH\_ADDR = S2\_PC\_ADD4 + (S2\_SIGNEXT\_IMME<<2);**

**S3\_ALU\_RES = EX\_ALU\_RES;**

**S3\_ALU\_ZERO = EX\_ALU\_ZERO;**

**S3\_MEM\_WRITE\_DATA = S2\_READ\_DATA2;**

**S3\_REG\_ADDR = S2\_REG\_DST ? S2\_REG\_ADDR\_LOW : S2\_REG\_ADDR\_HIGH;**

**// ID/EX**

**S2\_REG\_DST = ID\_REG\_DST;**

**S2\_ALU\_SRC = ID\_ALU\_SRC;**

**S2\_MEM\_TO\_REG = ID\_MEM\_TO\_REG;**

**S2\_REG\_WRITE = ID\_REG\_WRITE;**

**S2\_MEM\_READ = ID\_MEM\_READ;**

**S2\_MEM\_WRITE = ID\_MEM\_WRITE;**

**S2\_BRANCH = ID\_BRANCH;**

**S2\_ALU\_OP = ID\_ALU\_OP;**

**S2\_PC\_ADD4 = S1\_PC\_ADD4;**

**S2\_READ\_DATA1 = ID\_READ\_DATA1;**

**S2\_READ\_DATA2 = ID\_READ\_DATA2;**

**S2\_SIGNEXT\_IMME = ID\_SIGNEXT\_IMME;**

**S2\_REG\_ADDR\_HIGH = S1\_INST[20:16];**

**S2\_REG\_ADDR\_LOW = S1\_INST[15:11];**

**// IF/ID**

**S1\_PC\_ADD4 = PC+4;**

**S1\_INST = IF\_INST;**

**end**

**end**

**endmodule**

## 2.2 Mem\_Data模块

本次实验使用的Mem\_Data模块与实验5除初始化数值外完全相同。

**module data\_memory(**

**input clock\_in,**

**input [6:0] address,**

**input [31:0] writeData,**

**input reset,**

**input memRead,**

**input memWrite,**

**output reg [31:0] readData**

**);**

**reg [31:0] memFile[8:0];**

**//reg [31:0] readData;**

**always @(negedge clock\_in)**

**begin**

**if (memRead==1) readData=memFile[address];**

**else readData=0;**

**end**

**always @(posedge clock\_in)**

**begin**

**if (reset)**

**begin**

**memFile[0]<=1;**

**memFile[1]<=2;**

**memFile[2]<=3;**

**memFile[3]<=4;**

**memFile[4]<=5;**

**memFile[5]<=6;**

**memFile[6]<=7;**

**memFile[7]<=8;**

**memFile[8]<=9;**

**end**

**else**

**begin**

**if (memWrite==1) memFile[address]<=writeData;**

**end**

**end**

**endmodule**

## 2.3 Inst\_Memory模块

Inst Memory模块与Data\_Memory模块类似，也同样需要初始化，但它被用于存储MIPS指令，而且是只读的。代码如下：

**module inst\_memory(**

**input [31:0] pc,**

**input reset,**

**output reg [31:0] inst**

**);**

**reg [31:0] rom[31:0];**

**//reg [31:0] inst;**

**always @ (reset)**

**begin**

**rom[0]=32'b10001100000000010000000000000001;//lw $1,1($0)**

**rom[1]=32'b10001100000000100000000000000011;//lw $2,3($0)**

**rom[2]=32'b11111000000000000000000000000001;//nop**

**rom[3]=32'b11110000000000000000000000000000;//nop**

**rom[4]=32'b11110000000000000000000000000000;//nop**

**rom[5]=32'b00000000001000100001100000100000;//add $3,$1,$2**

**rom[6]=32'b00000000010000010010000000100010;//sub $4,$2,$1**

**rom[7]=32'b11111000000000000000000000000001;//nop**

**rom[8]=32'b11110000000000000000000000000000;//nop**

**rom[9]=32'b00000000011001000010100000101010;//slt $5,$3,$4**

**rom[10]=32'b00000000001000100011000000100100;//and $6,$1,$2**

**rom[11]=32'b00000000001000100011100000100101;//or $7,$1,$2**

**rom[12]=32'b10101100000000110000000000000001;//sw $3,1($0)**

**rom[13]=32'b10101100000001000000000000000010;//sw $4,3($0)**

**rom[14]=32'b11110000000000000000000000000000;//nop**

**rom[15]=32'b11110000000000000000000000000000;//nop**

**rom[16]=32'b00000000101001110001100000100000;//add $7,$3,$5**

**rom[17]=32'b00000000100001100100000000100000;//add $8,$4,$6**

**rom[18]=32'b00010000000000001111111111101100;//beq $0,$0,#-20**

**rom[19]=32'b11110000000000000000000000000000;//nop**

**rom[20]=32'b11110000000000000000000000000000;//nop**

**rom[21]=32'b11110000000000000000000000000000;//nop**

**end**

**always @(pc) inst=rom[pc>>2];**

**endmodule**

# 3.仿真

## 3.1 仿真代码

仿真代码主要部分如下，完整代码见test\_ for\_top.v。

**always #10 clockin=~clockin;**

**initial begin**

**clockin = 0;**

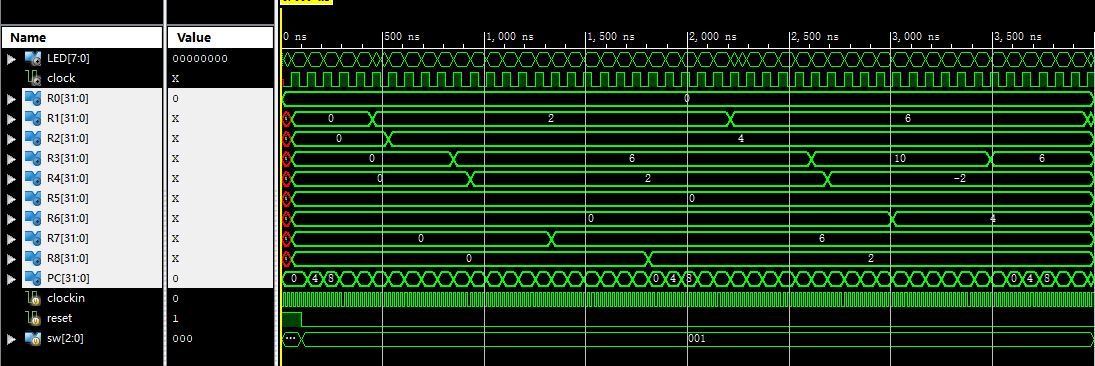
**reset = 1;**

**sw = 0;**

**#100;reset=0;sw = 1;**

**end**

## 3.2 仿真波形



仿真波形说明Top模块能够正确更新Program Counter，读取Instruction Memory，并根据指令进行相应ALU计算、寄存器与内存读写等操作。Top模块仿真实验成功。

# 4. 下载验证

## 4.1 实验描述

为在Spartan 3E实验板上模拟这一类MIPS处理器的运行，我们需要将实验板的内部时钟进行分频以便于观察，同时将板子上4个开关分别用作reset和一个用作寄存器地址的3位二进制数，实验板的八个LED灯中，最高一位用于表示时钟信号，第二高位为reset，紧接着的三位为所显示的寄存器地址，最后三位为所显示的寄存器的值。由于时钟信号上升沿被用来写寄存器，所以读取在下降沿进行。

## 4.2 模块改进

### 4.2.1 分频器模块

将时钟频率减缓倍。

**module time\_divider(**

**input clock\_in,**

**output reg clock\_out**

**);**

**reg [23:0] buffer;**

**initial begin buffer=0; end**

**always@(posedge clock\_in)**

**begin**

**buffer<=buffer+1;**

**clock\_out<=buffer[23];**

**end**

**endmodule**

### 4.2.2 Top模块改进

新增sw作为地址输入、LED作为寄存器输出。clock\_in为实验板晶振时钟，分频后得到clock才可用于观察。此外，还需要添加在时钟信号变化沿更新LED状态的代码。

**module top(**

**input clockin,**

**input reset,**

**input [2:0] sw,**

**output reg [7:0] LED,**

**output wire clock,**

**output wire [31:0] R0,**

**output wire [31:0] R1,**

**output wire [31:0] R2,**

**output wire [31:0] R3,**

**output wire [31:0] R4,**

**output wire [31:0] R5,**

**output wire [31:0] R6,**

**output wire [31:0] R7,**

**output wire [31:0] R8,**

**output reg [31:0] PC**

**);**

**// IF/ID**

**reg[31:0] S1\_PC\_ADD4;**

**reg[31:0] S1\_INST;**

**// ID/EX**

**reg S2\_REG\_WRITE, S2\_MEM\_TO\_REG; // WB**

**reg S2\_BRANCH, S2\_MEM\_READ, S2\_MEM\_WRITE; // M**

**reg S2\_REG\_DST, S2\_ALU\_SRC;**

**reg [1:0] S2\_ALU\_OP; // EX**

**reg[31:0] S2\_PC\_ADD4;**

**reg[31:0] S2\_READ\_DATA1;**

**reg[31:0] S2\_READ\_DATA2;**

**reg[31:0] S2\_SIGNEXT\_IMME;**

**reg[5:0] S2\_REG\_ADDR\_HIGH; //inst[20:16]**

**reg[5:0] S2\_REG\_ADDR\_LOW; //inst[15:11]**

**// EX/MEM**

**reg S3\_REG\_WRITE, S3\_MEM\_TO\_REG; // WB**

**reg S3\_BRANCH, S3\_MEM\_READ, S3\_MEM\_WRITE; // M**

**reg[31:0] S3\_BRANCH\_ADDR;**

**reg[31:0] S3\_ALU\_RES;**

**reg[31:0] S3\_ALU\_ZERO;**

**reg[31:0] S3\_MEM\_WRITE\_DATA; //read\_data2**

**reg[5:0] S3\_REG\_ADDR;**

**// MEM/WB**

**reg S4\_REG\_WRITE, S4\_MEM\_TO\_REG; // WB**

**reg[31:0] S4\_MEM\_READ\_DATA;**

**reg[31:0] S4\_ALU\_RES;**

**reg[5:0] S4\_REG\_ADDR;**

**// IF**

**wire PC\_SRC;**

**wire[31:0] IF\_INST;**

**// ID**

**wire ID\_REG\_DST;**

**wire ID\_ALU\_SRC;**

**wire ID\_MEM\_TO\_REG;**

**wire ID\_REG\_WRITE;**

**wire ID\_MEM\_READ;**

**wire ID\_MEM\_WRITE;**

**wire ID\_BRANCH;**

**wire[1:0] ID\_ALU\_OP;**

**wire[31:0] ID\_READ\_DATA1;**

**wire[31:0] ID\_READ\_DATA2;**

**wire[31:0] ID\_SIGNEXT\_IMME;**

**wire[31:0] MEM\_MEM\_READ\_DATA;**

**time\_divider mainTime(**

**.clock\_in(clockin),**

**.clock\_out(clock)**

**);**

**initial begin**

**LED=0;**

**end**

**always @ (posedge clock)**

**begin**

**LED[7]=~LED[7];**

**end**

**always @ (clock)**

**begin**

**case(sw[2:0])**

**3'b000: LED[2:0]<=R0[2:0];**

**3'b001: LED[2:0]<=R1[2:0];**

**3'b010: LED[2:0]<=R2[2:0];**

**3'b011: LED[2:0]<=R3[2:0];**

**3'b100: LED[2:0]<=R4[2:0];**

**3'b101: LED[2:0]<=R5[2:0];**

**3'b110: LED[2:0]<=R6[2:0];**

**3'b111: LED[3:0]<=R7[2:0];**

**endcase**

**LED[6]<=reset;**

**LED[5:3]<=sw[2:0];**

**end**

**inst\_memory mainRom(**

**.pc(PC),**

**.inst(IF\_INST),**

**.reset(reset)**

**);**

**ctr mainCtr(**

**.opCode(S1\_INST[31:26]),**

**.regDst(ID\_REG\_DST),**

**.aluSrc(ID\_ALU\_SRC),**

**.memToReg(ID\_MEM\_TO\_REG),**

**.regWrite(ID\_REG\_WRITE),**

**.memRead(ID\_MEM\_READ),**

**.memWrite(ID\_MEM\_WRITE),**

**.jump(JUMP),**

**.branch(ID\_BRANCH),**

**.aluOp(ID\_ALU\_OP)**

**);**

**register mainRegister(**

**.clock\_in(clock),**

**.readReg1(S1\_INST[25:21]),**

**.readReg2(S1\_INST[20:16]),**

**.writeReg(S4\_REG\_ADDR),**

**.regWrite(S4\_REG\_WRITE),**

**.writeData(S4\_MEM\_TO\_REG ? S4\_MEM\_READ\_DATA : S4\_ALU\_RES),**

**.reset(reset),**

**.readData1(ID\_READ\_DATA1),**

**.readData2(ID\_READ\_DATA2),**

**.reg0(R0),**

**.reg1(R1),**

**.reg2(R2),**

**.reg3(R3),**

**.reg4(R4),**

**.reg5(R5),**

**.reg6(R6),**

**.reg7(R7),**

**.reg8(R8)**

**);**

**signext mainSignext(**

**.inst(S1\_INST[15:0]),**

**.extend(ID\_SIGNEXT\_IMME)**

**);**

**wire[3:0] EX\_ALU\_CTR;**

**wire EX\_ALU\_ZERO;**

**wire[31:0] EX\_ALU\_RES;**

**AluCtr mainAluCtr(**

**.aluOp(S2\_ALU\_OP),**

**.funct(S2\_SIGNEXT\_IMME[5:0]),**

**.aluCtr(EX\_ALU\_CTR)**

**);**

**alu mainAlu(**

**.input1(S2\_READ\_DATA1),**

**.input2(S2\_ALU\_SRC ? S2\_SIGNEXT\_IMME : S2\_READ\_DATA2),**

**.aluCtr(EX\_ALU\_CTR),**

**.zero(EX\_ALU\_ZERO),**

**.aluRes(EX\_ALU\_RES)**

**);**

**data\_memory mainDataMemory(**

**.clock\_in(clock),**

**.address(S3\_ALU\_RES),**

**.writeData(S3\_MEM\_WRITE\_DATA),**

**.memRead(S3\_MEM\_READ),**

**.memWrite(S3\_MEM\_WRITE),**

**.readData(MEM\_MEM\_READ\_DATA),**

**.reset(reset)**

**);**

**assign PC\_SRC = S3\_ALU\_ZERO & S3\_BRANCH;**

**always@(posedge clock or posedge reset)**

**begin**

**if(reset)**

**begin**

**PC = 0;**

**// IF/ID**

**S1\_PC\_ADD4=0;**

**S1\_INST=32'b00000011111111111111100000100000;**

**// ID/EX**

**S2\_REG\_WRITE=0;**

**S2\_MEM\_TO\_REG=0;**

**S2\_BRANCH=0;**

**S2\_MEM\_READ=0;**

**S2\_MEM\_WRITE=0;**

**S2\_REG\_DST=0;**

**S2\_ALU\_SRC=0;**

**S2\_ALU\_OP=0;**

**S2\_PC\_ADD4=0;**

**S2\_READ\_DATA1=0;**

**S2\_READ\_DATA2=0;**

**S2\_SIGNEXT\_IMME=0;**

**S2\_REG\_ADDR\_HIGH=0; //inst[20:16]**

**S2\_REG\_ADDR\_LOW=0; //inst[15:11]**

**// EX/MEM**

**S3\_REG\_WRITE=0;**

**S3\_MEM\_TO\_REG=0; // WB**

**S3\_BRANCH=0;**

**S3\_MEM\_READ=0;**

**S3\_MEM\_WRITE=0; // M**

**S3\_BRANCH\_ADDR=0;**

**S3\_ALU\_RES=0;**

**S3\_ALU\_ZERO=0;**

**S3\_MEM\_WRITE\_DATA=0; //read\_data2**

**S3\_REG\_ADDR=0;**

**// MEM/WB**

**S4\_REG\_WRITE=0;**

**S4\_MEM\_TO\_REG=0; // WB**

**S4\_MEM\_READ\_DATA=0;**

**S4\_ALU\_RES=0;**

**S4\_REG\_ADDR=0;**

**end**

**else**

**begin**

**// MEM/WB**

**S4\_REG\_WRITE = S3\_REG\_WRITE;**

**S4\_MEM\_TO\_REG = S3\_MEM\_TO\_REG;**

**S4\_MEM\_READ\_DATA = MEM\_MEM\_READ\_DATA;**

**S4\_ALU\_RES = S3\_ALU\_RES;**

**S4\_REG\_ADDR = S3\_REG\_ADDR;**

**// PC**

**PC = PC\_SRC ? S3\_BRANCH\_ADDR: PC+4;**

**// EX/MEM**

**S3\_REG\_WRITE = S2\_REG\_WRITE;**

**S3\_MEM\_TO\_REG = S2\_MEM\_TO\_REG;**

**S3\_BRANCH = S2\_BRANCH;**

**S3\_MEM\_READ = S2\_MEM\_READ;**

**S3\_MEM\_WRITE = S2\_MEM\_WRITE;**

**S3\_BRANCH\_ADDR = S2\_PC\_ADD4 + (S2\_SIGNEXT\_IMME<<2);**

**S3\_ALU\_RES = EX\_ALU\_RES;**

**S3\_ALU\_ZERO = EX\_ALU\_ZERO;**

**S3\_MEM\_WRITE\_DATA = S2\_READ\_DATA2;**

**S3\_REG\_ADDR = S2\_REG\_DST ? S2\_REG\_ADDR\_LOW : S2\_REG\_ADDR\_HIGH;**

**// ID/EX**

**S2\_REG\_DST = ID\_REG\_DST;**

**S2\_ALU\_SRC = ID\_ALU\_SRC;**

**S2\_MEM\_TO\_REG = ID\_MEM\_TO\_REG;**

**S2\_REG\_WRITE = ID\_REG\_WRITE;**

**S2\_MEM\_READ = ID\_MEM\_READ;**

**S2\_MEM\_WRITE = ID\_MEM\_WRITE;**

**S2\_BRANCH = ID\_BRANCH;**

**S2\_ALU\_OP = ID\_ALU\_OP;**

**S2\_PC\_ADD4 = S1\_PC\_ADD4;**

**S2\_READ\_DATA1 = ID\_READ\_DATA1;**

**S2\_READ\_DATA2 = ID\_READ\_DATA2;**

**S2\_SIGNEXT\_IMME = ID\_SIGNEXT\_IMME;**

**S2\_REG\_ADDR\_HIGH = S1\_INST[20:16];**

**S2\_REG\_ADDR\_LOW = S1\_INST[15:11];**

**// IF/ID**

**S1\_PC\_ADD4 = PC+4;**

**S1\_INST = IF\_INST;**

**end**

**end**

**endmodule**

## 4.3 管脚定义

NET "clock" LOC **=** "C9" **|** IOSTANDARD **=** LVCMOS33**;**

NET "LED<7>" LOC **=** "F9" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "LED<6>" LOC **=** "E9" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "LED<5>" LOC **=** "D11" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "LED<4>" LOC **=** "C11" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "LED<3>" LOC **=** "F11" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "LED<2>" LOC **=** "E11" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "LED<1>" LOC **=** "E12" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "LED<0>" LOC **=** "F12" **|** IOSTANDARD **=** LVTTL **|** SLEW **=** SLOW **|** DRIVE **=** 8**;**

NET "reset" LOC **=** "L13" **|** IOSTANDARD **=** LVTTL **|** PULLUP**;**

NET "switches<2>" LOC **=** "N17" **|** IOSTANDARD **=** LVTTL **|** PULLUP**;**

NET "switches<1>" LOC **=** "H18" **|** IOSTANDARD **=** LVTTL **|** PULLUP**;**

NET "switches<0>" LOC **=** "L14" **|** IOSTANDARD **=** LVTTL **|** PULLUP**;**

## 4.4 实验结果

reset置为1后，拨动三个开关在8个寄存器之间选择，可观察到reg[1]至reg[7]依次产生数值，且经比对与仿真结果完全一致，而reg[0]始终为0。reset重新置为0后，寄存器全部清零，再次置为1可重新观察数值变化。由此，类MIPS流水化处理器上板模拟实验成功实现了。

# 5. 实验心得

本次实验较为复杂，虽然只是在实验5的基础上添加了4级缓冲寄存器结构，但是各级缓冲寄存器之间的逻辑、时序关系处理仍需要我们对于流水线有较深入的了解才能顺利实现。

在本次实验中，硬件设计层面上并没有考虑对于data hazard以及branch hazard的处理，需要我们在编写指令的时候从软件层面规避。由于MIPS流水化运行，如果在向寄存器写入运算结果之前，下一条指令就又读取了这个寄存器，这样就会计算出错误的结果，也即data hazard；一条branch指令若被执行，那么紧接着该指令的两条指令不应该被继续执行，但是由于流水线作业，这两条指令已经被读入了处理器，并将在接下来得到执行，这会造成control hazard。

本次实验中我选择从软件层面规避它们，对于data hazard，只需要重新安排指令的顺序，确保每个寄存器的写入与下一次读取之间间隔至少三条指令，必要时可以添加空指令；对于branch hazard，也只需要在branch指令后添加三条空指令即可解决。

由于本次实验比MIPS单周期处理器的整体结构还要复杂，对各个模块、每条线的命名都更加需要遵循良好的代码风格，以保证正确性。

这次实验的成功除了个人的努力外，也得益于同学之间的讨论、指导老师的讲解和实验指导书的详尽说明。在此感谢老师和同学们的帮助。