实验报告七

PB14000556 陈晓彤

实验题目：设计一个流水线CPU；编写合适的测试代码并成功运行

实验要求：

基本要求

60分（百分制），自行设计一个多周期MIPS CPU，包含16条指令，指令集见下页。

扩展要求

在16条的基础上，每增加1条指令，+1分

实现中断功能，+6~10分

实现流水，+12~20分

实现下载，+6~10分

设计思路：

计划实现流水线，和40条左右的指令数

基本流水线实现分五段：IF(Instruction Fetch) ID(Instruction Decode) EX(execute) MEM(memory) WB(write back) 和段寄存器IF-ID ID-EX EX-MEM MEM-WB来保存一些控制信号

另外，为实现乘法指令的结果保存，需要加hilo模块，而且为保证消除数据、指令相关等以保证流水线运行正确，需要加ctrl模块产生stall信号；ld指令的数据相关还需另外处理，所以要加上delayslot保存指令

为缩短调试时间，不采用ip核实现指令和数据存储器，而直接使用模块仿真（模块中用寄存器组实现）

具体设计情况见 cpu设计报告 文件

源代码：

“header.v”

//全局

`define RstEnable 1'b1

`define RstDisable 1'b0

`define ZeroWord 32'h00000000

`define WriteEnable 1'b1

`define WriteDisable 1'b0

`define ReadEnable 1'b1

`define ReadDisable 1'b0

`define AluOpBus 7:0

`define AluSelBus 2:0

`define InstValid 1'b0

`define InstInvalid 1'b1

`define Stop 1'b1

`define NoStop 1'b0

`define InDelaySlot 1'b1

`define NotInDelaySlot 1'b0

`define Branch 1'b1

`define NotBranch 1'b0

`define True\_v 1'b1

`define False\_v 1'b0

`define ChipEnable 1'b1

`define ChipDisable 1'b0

//指令

`define EXE\_AND 6'b100100

`define EXE\_OR 6'b100101

`define EXE\_XOR 6'b100110

`define EXE\_NOR 6'b100111

`define EXE\_ANDI 6'b001100

`define EXE\_ORI 6'b001101

`define EXE\_XORI 6'b001110

`define EXE\_LUI 6'b001111

`define EXE\_SLL 6'b000000

`define EXE\_SLLV 6'b000100

`define EXE\_SRL 6'b000010

`define EXE\_SRLV 6'b000110

`define EXE\_SRA 6'b000011

`define EXE\_SRAV 6'b000111

`define EXE\_SYNC 6'b001111

`define EXE\_PREF 6'b110011

`define EXE\_MOVZ 6'b001010

`define EXE\_MOVN 6'b001011

`define EXE\_MFHI 6'b010000

`define EXE\_MTHI 6'b010001

`define EXE\_MFLO 6'b010010

`define EXE\_MTLO 6'b010011

`define EXE\_SLT 6'b101010

`define EXE\_SLTU 6'b101011

`define EXE\_SLTI 6'b001010

`define EXE\_SLTIU 6'b001011

`define EXE\_ADD 6'b100000

`define EXE\_ADDU 6'b100001

`define EXE\_SUB 6'b100010

`define EXE\_SUBU 6'b100011

`define EXE\_ADDI 6'b001000

`define EXE\_ADDIU 6'b001001

`define EXE\_CLZ 6'b100000

`define EXE\_CLO 6'b100001

`define EXE\_MULT 6'b011000

`define EXE\_MULTU 6'b011001

`define EXE\_MUL 6'b000010

`define EXE\_MADD 6'b000000

`define EXE\_MADDU 6'b000001

`define EXE\_MSUB 6'b000100

`define EXE\_MSUBU 6'b000101

`define EXE\_J 6'b000010

`define EXE\_JAL 6'b000011

`define EXE\_JALR 6'b001001

`define EXE\_JR 6'b001000

`define EXE\_BEQ 6'b000100

`define EXE\_BGEZ 5'b00001

`define EXE\_BGEZAL 5'b10001

`define EXE\_BGTZ 6'b000111

`define EXE\_BLEZ 6'b000110

`define EXE\_BLTZ 5'b00000

`define EXE\_BLTZAL 5'b10000

`define EXE\_BNE 6'b000101

`define EXE\_LB 6'b100000

`define EXE\_LBU 6'b100100

`define EXE\_LH 6'b100001

`define EXE\_LHU 6'b100101

`define EXE\_LL 6'b110000

`define EXE\_LW 6'b100011

`define EXE\_LWL 6'b100010

`define EXE\_LWR 6'b100110

`define EXE\_SB 6'b101000

`define EXE\_SC 6'b111000

`define EXE\_SH 6'b101001

`define EXE\_SW 6'b101011

`define EXE\_SWL 6'b101010

`define EXE\_SWR 6'b101110

`define EXE\_NOP 6'b000000

`define SSNOP 32'b00000000000000000000000001000000

`define EXE\_SPECIAL\_INST 6'b000000

`define EXE\_REGIMM\_INST 6'b000001

`define EXE\_SPECIAL2\_INST 6'b011100

//AluOp

`define EXE\_AND\_OP 8'b00100100

`define EXE\_OR\_OP 8'b00100101

`define EXE\_XOR\_OP 8'b00100110

`define EXE\_NOR\_OP 8'b00100111

`define EXE\_ANDI\_OP 8'b01011001

`define EXE\_ORI\_OP 8'b01011010

`define EXE\_XORI\_OP 8'b01011011

`define EXE\_LUI\_OP 8'b01011100

`define EXE\_SLL\_OP 8'b01111100

`define EXE\_SLLV\_OP 8'b00000100

`define EXE\_SRL\_OP 8'b00000010

`define EXE\_SRLV\_OP 8'b00000110

`define EXE\_SRA\_OP 8'b00000011

`define EXE\_SRAV\_OP 8'b00000111

`define EXE\_MOVZ\_OP 8'b00001010

`define EXE\_MOVN\_OP 8'b00001011

`define EXE\_MFHI\_OP 8'b00010000

`define EXE\_MTHI\_OP 8'b00010001

`define EXE\_MFLO\_OP 8'b00010010

`define EXE\_MTLO\_OP 8'b00010011

`define EXE\_SLT\_OP 8'b00101010

`define EXE\_SLTU\_OP 8'b00101011

`define EXE\_SLTI\_OP 8'b01010111

`define EXE\_SLTIU\_OP 8'b01011000

`define EXE\_ADD\_OP 8'b00100000

`define EXE\_ADDU\_OP 8'b00100001

`define EXE\_SUB\_OP 8'b00100010

`define EXE\_SUBU\_OP 8'b00100011

`define EXE\_ADDI\_OP 8'b01010101

`define EXE\_ADDIU\_OP 8'b01010110

`define EXE\_CLZ\_OP 8'b10110000

`define EXE\_CLO\_OP 8'b10110001

`define EXE\_MULT\_OP 8'b00011000

`define EXE\_MULTU\_OP 8'b00011001

`define EXE\_MUL\_OP 8'b10101001

`define EXE\_MADD\_OP 8'b10100110

`define EXE\_MADDU\_OP 8'b10101000

`define EXE\_MSUB\_OP 8'b10101010

`define EXE\_MSUBU\_OP 8'b10101011

`define EXE\_J\_OP 8'b01001111

`define EXE\_JAL\_OP 8'b01010000

`define EXE\_JALR\_OP 8'b00001001

`define EXE\_JR\_OP 8'b00001000

`define EXE\_BEQ\_OP 8'b01010001

`define EXE\_BGEZ\_OP 8'b01000001

`define EXE\_BGEZAL\_OP 8'b01001011

`define EXE\_BGTZ\_OP 8'b01010100

`define EXE\_BLEZ\_OP 8'b01010011

`define EXE\_BLTZ\_OP 8'b01000000

`define EXE\_BLTZAL\_OP 8'b01001010

`define EXE\_BNE\_OP 8'b01010010

`define EXE\_LB\_OP 8'b11100000

`define EXE\_LBU\_OP 8'b11100100

`define EXE\_LH\_OP 8'b11100001

`define EXE\_LHU\_OP 8'b11100101

`define EXE\_LL\_OP 8'b11110000

`define EXE\_LW\_OP 8'b11100011

`define EXE\_LWL\_OP 8'b11100010

`define EXE\_LWR\_OP 8'b11100110

`define EXE\_PREF\_OP 8'b11110011

`define EXE\_SB\_OP 8'b11101000

`define EXE\_SC\_OP 8'b11111000

`define EXE\_SH\_OP 8'b11101001

`define EXE\_SW\_OP 8'b11101011

`define EXE\_SWL\_OP 8'b11101010

`define EXE\_SWR\_OP 8'b11101110

`define EXE\_SYNC\_OP 8'b00001111

`define EXE\_NOP\_OP 8'b00000000

//AluSel

`define EXE\_RES\_LOGIC 3'b001

`define EXE\_RES\_SHIFT 3'b010

`define EXE\_RES\_MOVE 3'b011

`define EXE\_RES\_ARITHMETIC 3'b100

`define EXE\_RES\_MUL 3'b101

`define EXE\_RES\_JUMP\_BRANCH 3'b110

`define EXE\_RES\_LOAD\_STORE 3'b111

`define EXE\_RES\_NOP 3'b000

//指令存储器inst\_rom

`define InstAddrBus 31:0

`define InstBus 31:0

`define InstMemNum 131071

`define InstMemNumLog2 17

//数据存储器data\_ram

`define DataAddrBus 31:0

`define DataBus 31:0

`define DataMemNum 131071

`define DataMemNumLog2 17

`define ByteWidth 7:0

//通用寄存器regfile

`define RegAddrBus 4:0

`define RegBus 31:0

`define RegWidth 32

`define DoubleRegWidth 64

`define DoubleRegBus 63:0

`define RegNum 32

`define RegNumLog2 5

`define NOPRegAddr 5'b00000

“cpu.v”

`include "header.v"

module cpu(

input clk,

input rst

);

wire [`InstAddrBus] pc;

wire [`InstAddrBus] id\_pc\_i;

wire [`InstBus] id\_inst\_i;

wire [`RegBus] rom\_data\_i;

wire rom\_ce\_o;

//连接数据存储器data\_ram

wire [`RegBus] ram\_data\_i;

wire [`RegBus] ram\_addr\_o;

wire [`RegBus] ram\_data\_o;

wire ram\_we\_o;

wire [3:0] ram\_sel\_o;

wire [3:0] ram\_ce\_o;

//连接译码阶段ID模块的输出与ID/EX模块的输入

wire [`AluOpBus] id\_aluop\_o;

wire [`AluSelBus] id\_alusel\_o;

wire [`RegBus] id\_reg1\_o;

wire [`RegBus] id\_reg2\_o;

wire id\_wreg\_o;

wire [`RegAddrBus] id\_wd\_o;

wire id\_is\_in\_delayslot\_o;

wire [`RegBus] id\_link\_address\_o;

wire [`RegBus] id\_inst\_o;

//连接ID/EX模块的输出与执行阶段EX模块的输入

wire [`AluOpBus] ex\_aluop\_i;

wire [`AluSelBus] ex\_alusel\_i;

wire [`RegBus] ex\_reg1\_i;

wire [`RegBus] ex\_reg2\_i;

wire ex\_wreg\_i;

wire [`RegAddrBus] ex\_wd\_i;

wire ex\_is\_in\_delayslot\_i;

wire [`RegBus] ex\_link\_address\_i;

wire [`RegBus] ex\_inst\_i;

//连接执行阶段EX模块的输出与EX/MEM模块的输入

wire ex\_wreg\_o;

wire [`RegAddrBus] ex\_wd\_o;

wire [`RegBus] ex\_wdata\_o;

wire [`RegBus] ex\_hi\_o;

wire [`RegBus] ex\_lo\_o;

wire ex\_whilo\_o;

wire [`AluOpBus] ex\_aluop\_o;

wire [`RegBus] ex\_mem\_addr\_o;

wire [`RegBus] ex\_reg1\_o;

wire [`RegBus] ex\_reg2\_o;

//连接EX/MEM模块的输出与访存阶段MEM模块的输入

wire mem\_wreg\_i;

wire [`RegAddrBus] mem\_wd\_i;

wire [`RegBus] mem\_wdata\_i;

wire [`RegBus] mem\_hi\_i;

wire [`RegBus] mem\_lo\_i;

wire mem\_whilo\_i;

wire [`AluOpBus] mem\_aluop\_i;

wire [`RegBus] mem\_mem\_addr\_i;

wire [`RegBus] mem\_reg1\_i;

wire [`RegBus] mem\_reg2\_i;

//连接访存阶段MEM模块的输出与MEM/WB模块的输入

wire mem\_wreg\_o;

wire [`RegAddrBus] mem\_wd\_o;

wire [`RegBus] mem\_wdata\_o;

wire [`RegBus] mem\_hi\_o;

wire [`RegBus] mem\_lo\_o;

wire mem\_whilo\_o;

wire mem\_LLbit\_value\_o;

wire mem\_LLbit\_we\_o;

//连接MEM/WB模块的输出与回写阶段的输入

wire wb\_wreg\_i;

wire [`RegAddrBus] wb\_wd\_i;

wire [`RegBus] wb\_wdata\_i;

wire [`RegBus] wb\_hi\_i;

wire [`RegBus] wb\_lo\_i;

wire wb\_whilo\_i;

wire wb\_LLbit\_value\_i;

wire wb\_LLbit\_we\_i;

//连接译码阶段ID模块与通用寄存器Regfile模块

wire reg1\_read;

wire reg2\_read;

wire [`RegBus] reg1\_data;

wire [`RegBus] reg2\_data;

wire [`RegAddrBus] reg1\_addr;

wire [`RegAddrBus] reg2\_addr;

//连接执行阶段与hilo模块的输出，读取HI、LO寄存器

wire [`RegBus] hi;

wire [`RegBus] lo;

//连接执行阶段与ex\_reg模块，用于多周期的MADD、MADDU、MSUB、MSUBU指令

wire [`DoubleRegBus] hilo\_temp\_o;

wire [1:0] cnt\_o;

wire [`DoubleRegBus] hilo\_temp\_i;

wire [1:0] cnt\_i;

wire is\_in\_delayslot\_i;

wire is\_in\_delayslot\_o;

wire next\_inst\_in\_delayslot\_o;

wire id\_branch\_flag\_o;

wire [`RegBus] branch\_target\_address;

wire [5:0] stall;

wire stallreq\_from\_id;

wire stallreq\_from\_ex;

//pc\_reg例化

pc\_reg pc\_reg0(

.clk(clk),

.rst(rst),

.stall(stall),

.branch\_flag\_i(id\_branch\_flag\_o),

.branch\_target\_address\_i(branch\_target\_address),

.pc(pc),

.ce(rom\_ce\_o)

);

//指令存储器inst\_rom例化

inst\_rom inst\_rom0(

.ce(rom\_ce\_o),

.addr(pc),

.inst(rom\_data\_i)

);

//IF/ID模块例化

if\_id if\_id0(

.clk(clk),

.rst(rst),

.stall(stall),

.if\_pc(pc),

.if\_inst(rom\_data\_i),

.id\_pc(id\_pc\_i),

.id\_inst(id\_inst\_i)

);

//译码阶段ID模块

id id0(

.rst(rst),

.pc\_i(id\_pc\_i),

.inst\_i(id\_inst\_i),

.ex\_aluop\_i(ex\_aluop\_o),

.reg1\_data\_i(reg1\_data),

.reg2\_data\_i(reg2\_data),

//处于执行阶段的指令要写入的目的寄存器信息

.ex\_wreg\_i(ex\_wreg\_o),

.ex\_wdata\_i(ex\_wdata\_o),

.ex\_wd\_i(ex\_wd\_o),

//处于访存阶段的指令要写入的目的寄存器信息

.mem\_wreg\_i(mem\_wreg\_o),

.mem\_wdata\_i(mem\_wdata\_o),

.mem\_wd\_i(mem\_wd\_o),

.is\_in\_delayslot\_i(is\_in\_delayslot\_i),

//送到regfile的信息

.reg1\_read\_o(reg1\_read),

.reg2\_read\_o(reg2\_read),

.reg1\_addr\_o(reg1\_addr),

.reg2\_addr\_o(reg2\_addr),

//送到ID/EX模块的信息

.aluop\_o(id\_aluop\_o),

.alusel\_o(id\_alusel\_o),

.reg1\_o(id\_reg1\_o),

.reg2\_o(id\_reg2\_o),

.wd\_o(id\_wd\_o),

.wreg\_o(id\_wreg\_o),

.inst\_o(id\_inst\_o),

.next\_inst\_in\_delayslot\_o(next\_inst\_in\_delayslot\_o),

.branch\_flag\_o(id\_branch\_flag\_o),

.branch\_target\_address\_o(branch\_target\_address),

.link\_addr\_o(id\_link\_address\_o),

.is\_in\_delayslot\_o(id\_is\_in\_delayslot\_o),

.stallreq(stallreq\_from\_id)

);

//通用寄存器Regfile例化

regfile regfile1(

.clk(clk),

.rst(rst),

.we(wb\_wreg\_i),

.waddr(wb\_wd\_i),

.wdata(wb\_wdata\_i),

.re1(reg1\_read),

.raddr1(reg1\_addr),

.rdata1(reg1\_data),

.re2(reg2\_read),

.raddr2(reg2\_addr),

.rdata2(reg2\_data)

);

//ID/EX模块

id\_ex id\_ex0(

.clk(clk),

.rst(rst),

.stall(stall),

//从译码阶段ID模块传递的信息

.id\_aluop(id\_aluop\_o),

.id\_alusel(id\_alusel\_o),

.id\_reg1(id\_reg1\_o),

.id\_reg2(id\_reg2\_o),

.id\_wd(id\_wd\_o),

.id\_wreg(id\_wreg\_o),

.id\_link\_address(id\_link\_address\_o),

.id\_is\_in\_delayslot(id\_is\_in\_delayslot\_o),

.next\_inst\_in\_delayslot\_i(next\_inst\_in\_delayslot\_o),

.id\_inst(id\_inst\_o),

//传递到执行阶段EX模块的信息

.ex\_aluop(ex\_aluop\_i),

.ex\_alusel(ex\_alusel\_i),

.ex\_reg1(ex\_reg1\_i),

.ex\_reg2(ex\_reg2\_i),

.ex\_wd(ex\_wd\_i),

.ex\_wreg(ex\_wreg\_i),

.ex\_link\_address(ex\_link\_address\_i),

.ex\_is\_in\_delayslot(ex\_is\_in\_delayslot\_i),

.is\_in\_delayslot\_o(is\_in\_delayslot\_i),

.ex\_inst(ex\_inst\_i)

);

//EX模块

ex ex0(

.rst(rst),

//送到执行阶段EX模块的信息

.aluop\_i(ex\_aluop\_i),

.alusel\_i(ex\_alusel\_i),

.reg1\_i(ex\_reg1\_i),

.reg2\_i(ex\_reg2\_i),

.wd\_i(ex\_wd\_i),

.wreg\_i(ex\_wreg\_i),

.hi\_i(hi),

.lo\_i(lo),

.inst\_i(ex\_inst\_i),

.wb\_hi\_i(wb\_hi\_i),

.wb\_lo\_i(wb\_lo\_i),

.wb\_whilo\_i(wb\_whilo\_i),

.mem\_hi\_i(mem\_hi\_o),

.mem\_lo\_i(mem\_lo\_o),

.mem\_whilo\_i(mem\_whilo\_o),

.hilo\_temp\_i(hilo\_temp\_i),

.cnt\_i(cnt\_i),

.link\_address\_i(ex\_link\_address\_i),

.is\_in\_delayslot\_i(ex\_is\_in\_delayslot\_i),

//EX模块的输出到EX/MEM模块信息

.wd\_o(ex\_wd\_o),

.wreg\_o(ex\_wreg\_o),

.wdata\_o(ex\_wdata\_o),

.hi\_o(ex\_hi\_o),

.lo\_o(ex\_lo\_o),

.whilo\_o(ex\_whilo\_o),

.hilo\_temp\_o(hilo\_temp\_o),

.cnt\_o(cnt\_o),

.aluop\_o(ex\_aluop\_o),

.mem\_addr\_o(ex\_mem\_addr\_o),

.reg2\_o(ex\_reg2\_o),

.stallreq(stallreq\_from\_ex)

);

//EX/MEM模块

ex\_mem ex\_mem0(

.clk(clk),

.rst(rst),

.stall(stall),

//来自执行阶段EX模块的信息

.ex\_wd(ex\_wd\_o),

.ex\_wreg(ex\_wreg\_o),

.ex\_wdata(ex\_wdata\_o),

.ex\_hi(ex\_hi\_o),

.ex\_lo(ex\_lo\_o),

.ex\_whilo(ex\_whilo\_o),

.ex\_aluop(ex\_aluop\_o),

.ex\_mem\_addr(ex\_mem\_addr\_o),

.ex\_reg2(ex\_reg2\_o),

.hilo\_i(hilo\_temp\_o),

.cnt\_i(cnt\_o),

//送到访存阶段MEM模块的信息

.mem\_wd(mem\_wd\_i),

.mem\_wreg(mem\_wreg\_i),

.mem\_wdata(mem\_wdata\_i),

.mem\_hi(mem\_hi\_i),

.mem\_lo(mem\_lo\_i),

.mem\_whilo(mem\_whilo\_i),

.mem\_aluop(mem\_aluop\_i),

.mem\_mem\_addr(mem\_mem\_addr\_i),

.mem\_reg2(mem\_reg2\_i),

.hilo\_o(hilo\_temp\_i),

.cnt\_o(cnt\_i)

);

//MEM模块例化

mem mem0(

.rst(rst),

//来自EX/MEM模块的信息

.wd\_i(mem\_wd\_i),

.wreg\_i(mem\_wreg\_i),

.wdata\_i(mem\_wdata\_i),

.hi\_i(mem\_hi\_i),

.lo\_i(mem\_lo\_i),

.whilo\_i(mem\_whilo\_i),

.aluop\_i(mem\_aluop\_i),

.mem\_addr\_i(mem\_mem\_addr\_i),

.reg2\_i(mem\_reg2\_i),

//来自memory的信息

.mem\_data\_i(ram\_data\_i),

//送到MEM/WB模块的信息

.wd\_o(mem\_wd\_o),

.wreg\_o(mem\_wreg\_o),

.wdata\_o(mem\_wdata\_o),

.hi\_o(mem\_hi\_o),

.lo\_o(mem\_lo\_o),

.whilo\_o(mem\_whilo\_o),

//送到memory的信息

.mem\_addr\_o(ram\_addr\_o),

.mem\_we\_o(ram\_we\_o),

.mem\_sel\_o(ram\_sel\_o),

.mem\_data\_o(ram\_data\_o),

.mem\_ce\_o(ram\_ce\_o)

);

//数据存储器data\_ram例化

data\_ram data\_ram0(

.clk(clk),

.ce(ram\_ce\_o),

.we(ram\_we\_o),

.sel(ram\_sel\_o),

.addr(ram\_addr\_o),

.data\_i(ram\_data\_o),

.data\_o(ram\_data\_i)

);

//MEM/WB模块

mem\_wb mem\_wb0(

.clk(clk),

.rst(rst),

.stall(stall),

//来自访存阶段MEM模块的信息

.mem\_wd(mem\_wd\_o),

.mem\_wreg(mem\_wreg\_o),

.mem\_wdata(mem\_wdata\_o),

.mem\_hi(mem\_hi\_o),

.mem\_lo(mem\_lo\_o),

.mem\_whilo(mem\_whilo\_o),

//送到回写阶段的信息

.wb\_wd(wb\_wd\_i),

.wb\_wreg(wb\_wreg\_i),

.wb\_wdata(wb\_wdata\_i),

.wb\_hi(wb\_hi\_i),

.wb\_lo(wb\_lo\_i),

.wb\_whilo(wb\_whilo\_i)

);

hilo\_reg hilo\_reg0(

.clk(clk),

.rst(rst),

//写端口

.we(wb\_whilo\_i),

.hi\_i(wb\_hi\_i),

.lo\_i(wb\_lo\_i),

//读端口1

.hi\_o(hi),

.lo\_o(lo)

);

ctrl ctrl0(

.rst(rst),

.stallreq\_from\_id(stallreq\_from\_id),

//来自执行阶段的暂停请求

.stallreq\_from\_ex(stallreq\_from\_ex),

.stall(stall)

);

endmodule

其余文件内容过多，不在此一一显示，详见工程文件

仿真结果

具体情况见仿真说明文件







