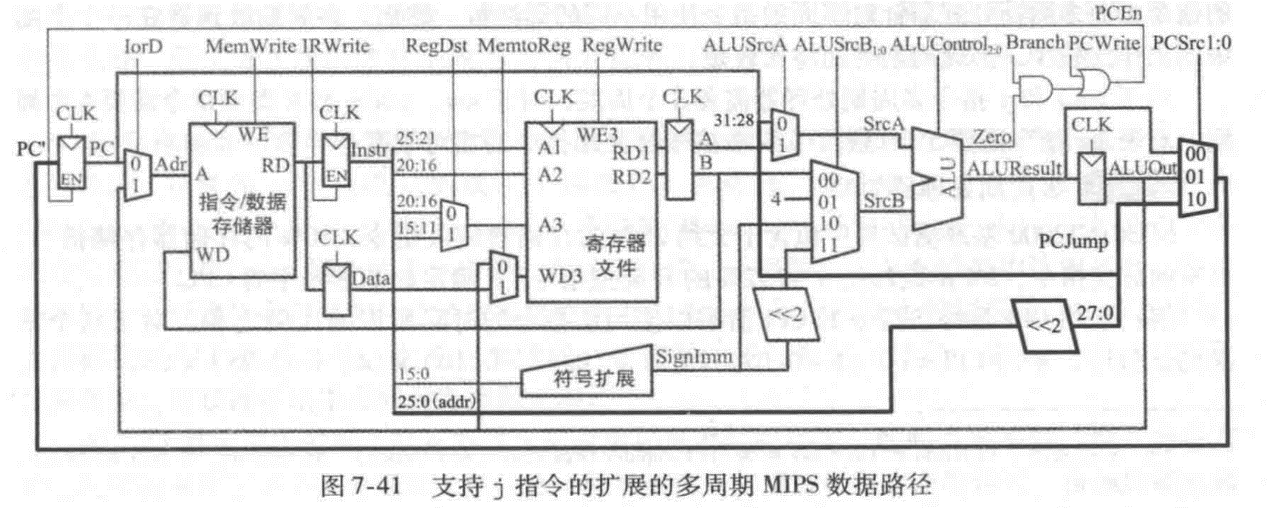
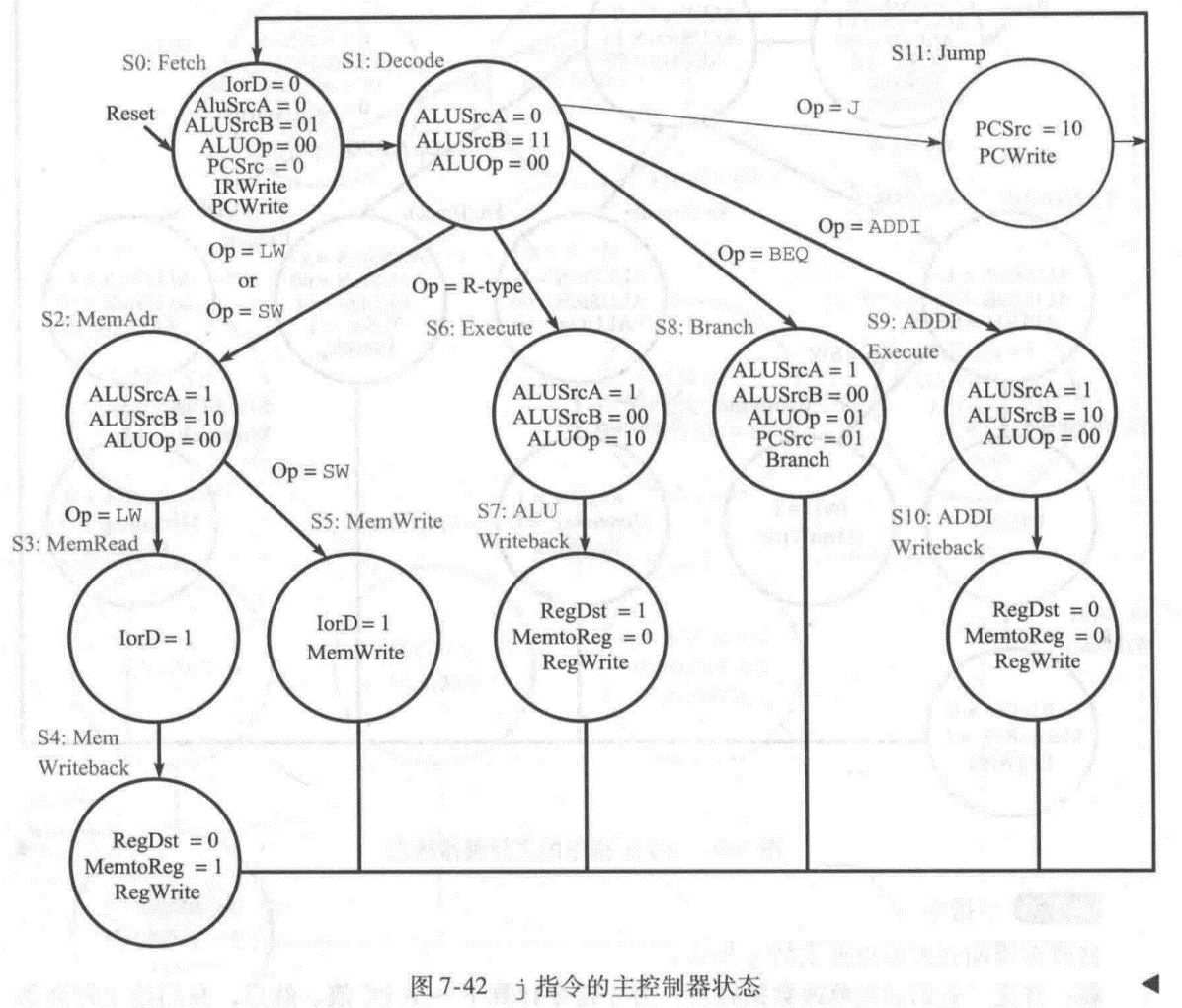
计算机体系结构实验3：32位多周期处理器设计

1. 实验要求
2. 仔细阅读**教材**7.4节(P240-255)内容
3. 完成MIPS多周期处理器设计
4. 用**教材**图7-60(P276)测试代码，测试上述设计
5. 参考《Multicycle Processor.pdf》完成设计、模拟
6. MIPS处理器的设计
7. 教材上给出了多周期MIPS处理器的设计图如下

数据路径：



控制路径：



1. 编码

将教材上给定的状态机翻译成Verilog代码即可。需要注意的代码要与习题7.25中给出的顶层模块代码兼容。我们对顶层模块稍作修改以便于显示出寄存器的值。

多周期处理器与单周期处理器的主要区别在于，一是在数据路径中插入了许多非体系结构状态单元以保存中间状态，二是主控制器是一个有限状态机而不是单纯的组合逻辑。下面给出了数据路径和主控制器的代码。其余部分的代码请查看项目。

//The select signals are MemtoReg, RegDst, IorD, PCSrc, ALUSrcB, and ALUSrcA.

//The enable signals are IRWrite, MemWrite, PCWrite, Branch, and RegWrite.

**module** datapath**(**

**input** **logic** CLK**,** reset**,** MemtoReg**,** RegDst**,** IorD**,** ALUSrcA**,**

**input** **logic** **[**2**:**0**]** ALUControl**,**

**input** **logic** **[**1**:**0**]** ALUSrcB**,** PCSrc**,**

**input** **logic** IRWrite**,** PCWrite**,** Branch**,** RegWrite**,**

**output** **logic** **[**5**:**0**]** Op**,** Funct**,**

**input** **logic** **[**31**:**0**]** RD**,**

**output** **logic** **[**31**:**0**]** Adr**,** B**,**

**output** **logic** **[**31**:**0**]** regs **[**31**:**0**]**

**);**

**logic** PCEn**,** Zero**;**

**logic** **[**4**:**0**]** A3**,** writereg**;**

**logic** **[**31**:**0**]** PC\_**,** PC**,** ALUOut**,** Instr**,** Data**,** RD1**,** RD2**,** SignImm**,** SignImmSh**,** SrcA**,** SrcB**,** A**,** WD3**,** ALUResult**,** PCJump**;**

**always\_ff@(posedge** CLK**)** **begin**

$display**(**"PC:%H, Instr:%H, PCNext:%H, SrcA:%H, SrcB:%H, ALUControl:%H, ALUResult:%H, ALUOut:%H"**,** PC**,** Instr**,** PC\_**,** SrcA**,** SrcB**,** ALUControl**,** ALUResult**,** ALUOut**);**

**end**

// next PC 1ogic

//PC

//instruction address, converted to @instr

**assign** PCEn **=** PCWrite **|** **(**Branch **&** Zero**);**

flopenr**#(**32**)** pcreg**(**CLK**,** reset**,** PCEn**,** PC\_**,** PC**);**

mux2**#(**32**)** addr**(**PC**,** ALUOut**,** IorD**,** Adr**);**

flopenr**#(**32**)** instrReg**(**CLK**,** reset**,** IRWrite**,** RD**,** Instr**);**

**assign** Op **=** Instr**[**31**:**26**];**

**assign** Funct **=** Instr**[**5**:**0**];**

mux2**#(**5**)** a3Mux**(**Instr**[**20**:**16**],** Instr**[**15**:**11**],** RegDst**,** A3**);**

flopr**#(**32**)** datareg**(**CLK**,** reset**,** RD**,** Data**);**

mux2**#(**32**)** wd3Mux**(**ALUOut**,** Data**,** MemtoReg**,** WD3**);**

regfile rf**(**CLK**,** RegWrite**,** Instr**[**25**:**21**],** Instr**[**20**:**16**],** A3**,** WD3**,** RD1**,** RD2**,** regs**);**

flopr**#(**32**)** a**(**CLK**,** reset**,** RD1**,** A**);**

flopr**#(**32**)** b**(**CLK**,** reset**,** RD2**,** B**);**

signext se**(**Instr**[**15**:**0**],** SignImm**);**

sl2 immsh**(**SignImm**,** SignImmSh**)** **;**

mux2**#(**32**)** srcAMux2**(**PC**,** A**,** ALUSrcA**,** SrcA**);**

mux4**#(**32**)** srcBMux4**(**B**,** 32'b100**,** SignImm**,** SignImmSh**,** ALUSrcB**,** SrcB**);**

alu alu**(**SrcA**,** SrcB**,** ALUControl**,** ALUResult**,** Zero**);**

flopr**#(**32**)** aluout**(**CLK**,** reset**,** ALUResult**,** ALUOut**);**

**assign** PCJump**[**31**:**28**]** **=** PC**[**31**:**28**];**

sl2 pcjumpsh**(**Instr**[**25**:**0**],** PCJump**[**27**:**0**]);**

mux4**#(**32**)** mux2pc**(**ALUResult**,** ALUOut**,** PCJump**,** 0**,** PCSrc**,** PC\_**);**

**endmodule**

**module** maindec**(**

**input** **logic** CLK**,** reset**,**

**input** **logic** **[**5**:**0**]** Op**,**

**output** **logic** **[**1**:**0**]** ALUOp**,**

**output** **logic** MemtoReg**,** RegDst**,** IorD**,**

**output** **logic** **[**1**:**0**]** PCSrc**,**

**output** **logic** ALUSrcA**,**

**output** **logic** **[**1**:**0**]** ALUSrcB**,**

**output** **logic** IRWrite**,** MemWrite**,** PCWrite**,** Branch**,** RegWrite

**);**

**logic** **[**3**:**0**]** nextState**,** S**,** counter**;**

**logic** **[**14**:**0**]** controlbits**;**

**assign** MemtoReg **=** controlbits**[**14**];**

**assign** RegDst **=** controlbits**[**13**];**

**assign** IorD **=** controlbits**[**12**];**

**assign** ALUSrcA **=** controlbits**[**11**];**

**assign** PCSrc **=** controlbits**[**10**:**9**];**

**assign** ALUSrcB **=** controlbits**[**8**:**7**];**

**assign** ALUOp **=** controlbits**[**6**:**5**];**

**assign** **{**IRWrite**,** MemWrite**,** PCWrite**,** Branch**,** RegWrite**}** **=** controlbits**[**4**:**0**];**

**always\_comb** **begin**

**case(**S**)**

0**:** controlbits **=** 15'b0000\_00\_01\_00\_10100**;**

1**:** controlbits **=** 15'b0000\_00\_11\_00\_00000**;**

2**:** controlbits **=** 15'b0001\_00\_10\_00\_00000**;**

3**:** controlbits **=** 15'b0010\_00\_00\_00\_00000**;**

4**:** controlbits **=** 15'b1000\_00\_00\_00\_00001**;**

5**:** controlbits **=** 15'b0010\_00\_00\_00\_01000**;**

6**:** controlbits **=** 15'b0001\_00\_00\_10\_00000**;**

7**:** controlbits **=** 15'b0100\_00\_00\_00\_00001**;**

8**:** controlbits **=** 15'b0001\_01\_00\_01\_00010**;**

9**:** controlbits **=** 15'b0001\_00\_10\_00\_00000**;**

10**:** controlbits **=** 15'b0000\_00\_00\_00\_00001**;**

11**:** controlbits **=** 15'b0000\_10\_00\_00\_00100**;**

**endcase**

**end**

**always\_comb** **begin**

**case(**S**)**

0**:** nextState **=** 1**;**

1**:** **case(**Op**)**

6'b100011**:** nextState **=** 2**;**

6'b101011**:** nextState **=** 2**;**

6'b000000**:** nextState **=** 6**;**

6'b000100**:** nextState **=** 8**;**

6'b001000**:** nextState **=** 9**;**

6'b000010**:** nextState **=** 11**;**

**endcase**

2**:** **case(**Op**)**

6'b100011**:** nextState **=** 3**;**

6'b101011**:** nextState **=** 5**;**

**endcase**

3**:** nextState **=** 4**;**

4**:** nextState **=** 0**;**

5**:** nextState **=** 0**;**

6**:** nextState **=** 7**;**

7**:** nextState **=** 0**;**

8**:** nextState **=** 0**;**

9**:** nextState **=** 10**;**

10**:** nextState **=** 0**;**

11**:** nextState **=** 0**;**

**endcase**

**end**

**always\_ff** **@(posedge** CLK**,** **posedge** reset**)** **begin**

**if(**reset**)** **begin**

S **<=** 0**;**

counter **<=** 0**;**

**end**

**else** **begin**

S **<=** nextState**;**//S == 0 ? 1: (S == 1 ? (Op == 6'b100011 | Op == 6'b101011 ? 2 : (Op==6'b000000 ? 6 : (Op==6'b000100 ? 8 : 9))):(S == 2 ? (Op==6'b100011 ? 3 : 5) : (S == 3 ? 4 : (S == 4 | S == 5 | S == 7 | S == 8 | S == 10? 0 : ( S == 6 ? 7 : 10)))));

counter **<=** counter **+** 1**;**

**end**

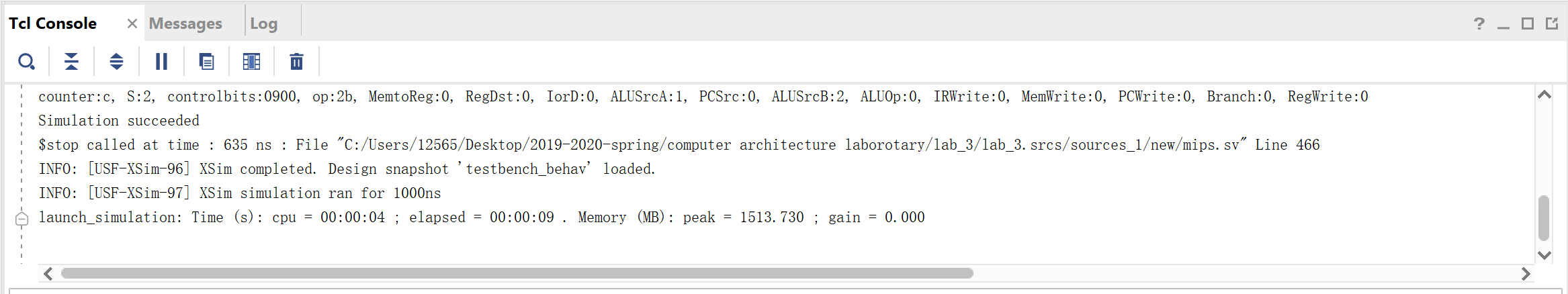
$display**(**"counter:%H, S:%H, controlbits:%H, op:%H, MemtoReg:%H, RegDst:%H, IorD:%H, ALUSrcA:%H, PCSrc:%H, ALUSrcB:%H, ALUOp:%H, IRWrite:%H, MemWrite:%H, PCWrite:%H, Branch:%H, RegWrite:%H"**,** counter**,** S**,** controlbits**,** Op**,** MemtoReg**,** RegDst**,** IorD**,** ALUSrcA**,** PCSrc**,** ALUSrcB**,** ALUOp**,** IRWrite**,** MemWrite**,** PCWrite**,** Branch**,** RegWrite**);**

**end**

**endmodule**

1. 测试和验证

我们使用了教材7.6.3中给出的基准测试程序进行测试。通过$display打印出状态值来跟踪和调试错误。命令行的输出提示模拟成功。



为了更加清楚地查看模拟的结果，我们将寄存器的值也显示了出来。可以从过程中寄存器值的变化验证程序的正确性。



