第5章 拓展实验

经过前面章节的学习，大家已经掌握了单周期CPU设计的基本方法。本章内容只是提出具体的设计要求，需要大家经过学习自行完成相关实验内容，进一步提高系统设计的能力。

5.1 多周期CPU的设计与实现

实验目的

(1) 深入理解每条指令的执行过程。

(2) 掌握多周期CPU的设计方法。

实验原理

多周期CPU根据不同指令各自的功能需求为其选择不同的执行步骤，每个步骤占用一个时钟周期。设计时尽量平衡各步骤间的延迟，就能从总体上提高CPU的运行速度。

CPU包括控制器、运算器、寄存器文件等部件。控制器解析指令并在每个步骤向各个操作部件发送控制信号；运算器负责执行算术/逻辑运算或计算访存地址；寄存器文件用于存储数据。具体的多周期CPU运行原理及结构，请参考计算机组成原理相关书籍。

实验内容

在理解多周期CPU工作原理的基础上，使用Verilog语言设计一款能够支持7条指令(add、addi、and、beq、j、lw、sw)的多周期CPU。本实验中，重点是多周期CPU控制器模块的设计。

实验过程

1. 确定执行流程图

分析给定指令(add、addi、and、beq、j、lw、sw)的执行过程，划分指令的执行步骤。

2. 确定数据通路

因为多周期CPU将指令的执行分解成多个步骤，每个步骤占用一个时钟周期，所以多周期CPU的数据通路是在单周期CPU数据通路的基础上插入寄存器实现的，参考框架如图5.1所示。在每个时钟周期结束时把本周期执行的结果保存在某个寄存器，便于下一个时钟周期使用。与前面的单周期CPU数据通路相比，多周期CPU将指令存储器(IM)和数据存储器(DM)合并为一个存储器，并增加了指令寄存器(IR)存放从存储器中

取出的指令。数据寄存器(DR),存放从存储器中读出的数据。A、B寄存器存放从寄存器文件读出的数据,C寄存器存放运算器的结果。

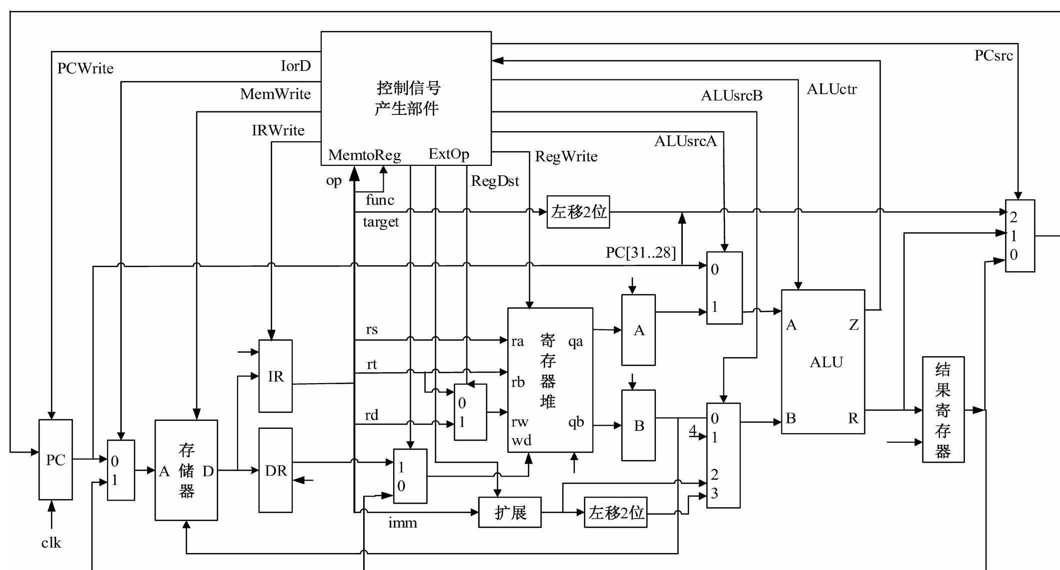

图 5.1 多周期 CPU 的数据通路

相应的控制信号也有所变化,具体描述如下。

PCWrite:程序计数器(PC)写使能信号,1有效。

IorD:存储器地址选择,1:选择寄存器C作为访存地址。0:选择PC作为访存地址。

IRWrite:指令寄存器(IR)写使能信号,1有效。

ALUsrcA:运算器a端数据选择,0:pc;1:A寄存器。

ALUsrcB[1:0]:运算器b端数据选择,00:B寄存器;01:4;10:符号扩展的立即数;11:符号扩展左移2位的立即数。

ALUctr[1:0]:运算器ALU控制信号,00:加法;01:减法;10:或运算;11:与运算。

RegDst:控制RF写寄存器编号,1:rd;0:rt。

MemtoReg:控制RF写入数据,0:ALU运算的结果;1:存储器读出的数据。

RegWrite:寄存器写使能信号,1有效。

MemWrite:存储器写使能信号,1有效。

ExtOp:扩展选择,0:零扩展;1:符号扩展。

PCsrc[1:0]:下地址选择:00:寄存器C的输出;01:运算器输出;10:J指令地址;11:无用。

3. 确定指令流程表

依据指令的执行步骤,确定每条指令执行过程中每个阶段所需的控制信号。

1)取指令周期(IF)

- 取指令周期完成两件事:读取指令和PC+4。

IR←Memory[PC]；

PC←PC＋4。

所需的控制信号：IRWrite＝1，PCWrite＝1，ALUsrcA＝0，ALUsrcB＝01，ALUctr[1:0]＝00，PCsrc[1:0]＝01。

2）指令译码周期（ID）

- 结束j指令无条件转移：把PC中的高4位与指令中的target左移两位拼接，得到j指令的转移地址。

PC←{PC[32:28]，target，00}

所需的控制信号：PCWrite＝1，PCsrc[1:0]＝10。

- 其他指令完成3件事：根据寄存器号rs，rt从寄存器文件中读出两个操作数，将它们分别放在寄存器A和B中。同时，ALU计算转移地址，即把PC与指令中的偏移量符号扩展、左移两位相加。计算出的转移地址写入寄存器C，为转移指令beq做准备。

A←RF[rs]：取第一个操作数

B←RF[rt]：取第二个操作数

C←PC＋sign_ext(imm) <<2。

所需的控制信号：IRWrite＝0，PCWrite＝0，ALUsrcA＝0，ALUsrcB＝11，ALUctr[1:0]＝00，ExtOp＝1。

3）指令执行周期（EXE）

- beq指令：判断A和B两个寄存器的内容是否相等，若相等，则修改PC实现转移。

if (A＝B)　PC←C(PC＋sign_ext(imm)<<2)

所需的控制信号：ALUsrcA＝1，ALUsrcB＝00，ALUctr[1:0]＝01，PCWrite＝1，PCsrc[1:0]＝00。

- 寄存器类指令的算术和逻辑运算指令(add/and)，完成相应的操作。

C←A op B

所需的控制信号：ALUsrcA＝1，ALUsrcB＝00，ALUctr[1:0]＝00/11。

- 立即数类指令(addi、lw、sw)。

addi：C←A＋sign_ext(imm)

lw/sw：C←A＋sign_ext(offset)；计算存储器地址，存放在C寄存器中。

所需的控制信号：ALUsrcA＝1，ALUsrcB＝10，ALUctr[1:0]＝00。

4）存储器访问周期（MEM）

只有lw和sw指令进入存储器访问周期，实现存储器中的读(lw)、写(sw)功能。

- lw：DR←Memory[C]（指令执行周期计算的存储器地址存放在C寄存器中）

所需的控制信号：IorD＝1。

- sw：Memory[C]←B（将B寄存器中的内容写入存储器）

所需的控制信号：IorD＝1，MemWrite＝1。

表 5.1 控制信号取值

输入			输出										
状态	指令	z	PCWrite	PCsrc[1:0]	IRWrite	IorD	MemWrite	ALUctr[1:0]	ALUsrcA	ALUsrcB[1:0]	RegWrite	MemtoReg	RegDst
sif	x	x											
sid	i—j	x											
	others	x											
sexe	I_add	x	0	xx	0	x	0	00	1	00	0	x	x
	I_sub	x											
	I_and	x											
	I_addi	x											
	I_beq	0											
		1											
	I_lw	x											
	I_sw	x											
smem	I_lw												
	I_sw												
swb	R_type												
	I_lw												

5）结果写回周期(WB)

结果写回周期把ALU的计算结果或者从存储器取来的数据写入寄存器文件。

add,sub,and：RF[rd]←C

addi：RF[rd]←C

lw：RF[rt]←DR

所需的控制信号：RegWrite=1;MemtoReg 选择 DR 或 C;RegDst 选择 rd 或 rt。

依照前面的分析,填写表5.1的控制信号取值。

4. 用状态机实现多周期CPU的控制模块

如图5.2所示,采用典型的时序电路——状态机实现多周期CPU的控制模块。

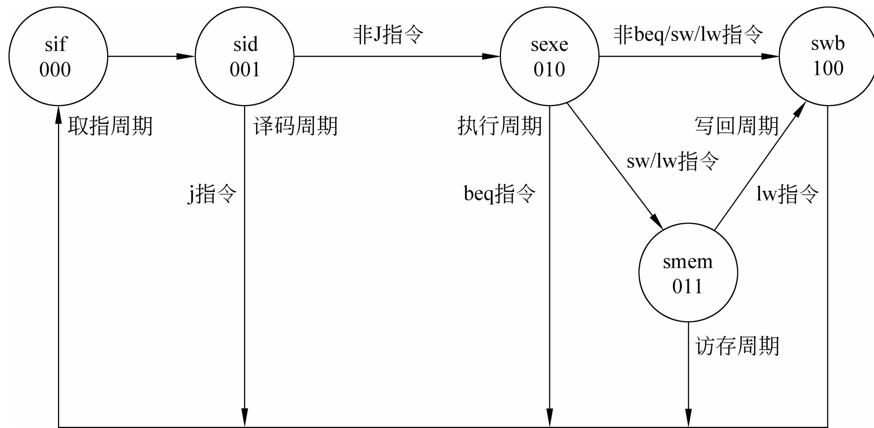

图5.2 多周期状态转移图

控制器参考代码如下。

```
module ctrl(clk,reset,func,op,z,regwrite,alucontrol,alusrca,alusrcb,regdst,
pcsrc,memwrite,irwrite,iord,pcwrite,memtoreg,state);
input clk;                          //时钟
input reset;                        //复位,低有效
input [5:0] func;                   //指令的 func 字段
input [5:0] op;                     //指令的 op 字段
input z;                            //运算结果零标志
output reg regwrite;                //寄存器写使能
output reg [2:0] alucontrol;        //ALU 控制信号(000:加法;001:减法;010:或;011:与)
output reg alusrca;                 //ALU 的 a 端数据来源(0:pc;1:A 寄存器)
output reg [1:0] alusrcb;           //ALU 的 b 端数据来源(00:B 寄存器;01:4;10:扩展;
                                      11:偏移)
output reg regdst;                  //写寄存器号:(为 1,写入 rt,否则写入 rd)
output reg memtoreg;                //rf 数据来源(0:寄存器 C;1:数据寄存器)
output reg memwrite;                //存储器写使能
output reg irwrite;                 //指令寄存器(IR)的写使能
output reg pcwrite;                 //PC 寄存器的写使能
```

```
output reg [2:0] state;       //状态(000:取指;001:译码;010:执行;011:访存;100:写回)
output reg iord;              //存储器地址选择:0:PC;1:C 寄存器)
output reg [1:0] pcsrc;       //PC 数据来源(00:C 寄存器;01: ALU;10:j 指令转移地址)
reg [2:0] next_state;
parameter [2:0] sif=3'b000;
parameter [2:0] sid=3'b001;
parameter [2:0] sexe=3'b010;
parameter [2:0] smem=3'b011;
parameter [2:0] swb=3'b100;
wire r_type;
wire i_add;
wire i_sub;
wire i_and;
wire i_addi;
wire i_beq;
wire i_j;
wire i_sw;
wire i_lw;
assign r_type=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0]);
                              //op:000000
assign i_add=r_type &(func[5])&(~func[4])&(~func[3])&(~func[2])&(~func[1])&
(~func[0]);                   //func:100000
assign i_sub=r_type &(func[5])&(~func[4])&(~func[3])&(~func[2])&(func[1])&(~
func[0]);                     //func:100010
assign i_and=r_type &(func[5])&(~func[4])&(~func[3])&(func[2])&(~func[1])&(~
func[0]);                     //func:100100
assign i_addi=(~op[5])&(~op[4])&(op[3])&(~op[2])&(~op[1])&(~op[0]);
                              //op:001000
assign i_beq=(~op[5])&(~op[4])&(~op[3])&(op[2])&(~op[1])&(~op[0]);
                              //op:000100
assign i_j=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(~op[0]);
                              //op:000010
assign i_sw=(op[5])&(~op[4])&(op[3])&(~op[2])&(op[1])&(op[0]);
                              //op:101011
assign i_lw=(op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(op[0]);
                              //op:100011
always @(*)                   //控制信号的默认输出
begin
    pcwrite=0;
    irwrite=0;
    regwrite=0;
    memwrite=0;
    regdst=0;
    memtoreg=0;
```

```
alusrca=1;
alusrcb=0;
alucontrol=3'b000;
iord=0;
pcsrc=0;
case(state)
    sif:begin                       //sif(0):取指令周期
        iord=0;
        pcwrite=1;                  //写 PC
        irwrite=1;                  //写 IR
        alusrca=0;                  //pc,pc+4
        alusrcb=2'b01;              //4
        next_state=sid;             //转指令译码周期
    end
    //--------------------------------指令译码周期
    sid:begin
        if(i_j)begin                //j 指令?
            pcsrc=2'h3;             //jump address,Pc<-{pc[31:28],adr,00}
            pcwrite=1;              //写 PC
            next_state=sif;         //是,结束 j 指令
            end
        else begin
            xxx                     //补充代码
            xxx
            next_state=sexe;
        end
    end
    //--------------------------------------------指令执行周期
    sexe:begin                      //补充代码
        xxx
        xxx
        end
    //-------------------------------------------访存周期
        smem:begin
            iord=1;                 //memory addr=C
            if(i_lw)                //lw 指令,转结果写回周期

                next_state=swb;
            else                    //sw 指令,写存储器并返回取指令周期
                begin
                xxx                 //补充代码
                xxx
                end
        end
```

```
    //-------------------------------------------写回周期
            swb: begin
              if(i_lw)
                memtoreg=1;             //lw 指令选择存储器数据写入寄存器
              if(i_lw || i_addi)
                xxx;                    //补充代码
                xxx;                    //结束指令,取下一条指令
            end
            default:next_state=sif;
        endcase
end
always@ (posedge clk or negedge reset)
begin
    if(reset==0)begin
        state <=sif;
    end else begin
        state <=next_state;
    end
end
endmodule
```

5. 多周期 CPU 顶层设计

自行完成各模块的设计后,建立顶层文件,要求接口关系如图 5.3 所示,然后参照图 5.1 导入设计完成的各模块。

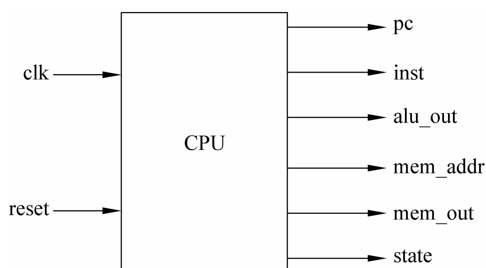

图 5.3 多周期 CPU 接口

clk:系统时钟	reset:复位信号,低电平有效
pc:PC 寄存器的输出	inst:当前指令。
alu_out:运算器的输出	mem_addr:存储器地址
mem_out:存储器输出	state:状态输出

6. 仿真测试

利用给定程序或自己编制一段程序,对设计的多周期 CPU 进行功能仿真。

7. 下载实现

将仿真正确的 CPU 设计下载到 Minisys 开发板上进行验证。

要求:ALU 的运算结果显示在 Minisys 开发板的 8 个数码管上,便于人眼观察程序

的执行过程，时钟周期设置为200ms。选3个LED指示灯显示当前的状态。

5.2 5级流水线CPU设计与实现

实验目的

(1) 深入理解流水线的原理。

(2) 掌握5级流水线CPU的设计方法。

(3) 掌握5级流水线CPU冲突的解决办法。

实验原理

5级流水线CPU就是把CPU的处理过程分为5个阶段，每个阶段占用一个时钟周期。在一个时钟周期内，各流水段为不同的指令服务。也就是说，在一个时钟周期内，每条指令只在一个流水段上是活动的。指令的执行时间有所重叠，每结束指令的一个执行步骤，就启动下一条指令，流水线中的所有部件都高速运行，尖峰速度每个CPU时钟执行一条指令。

实验内容

在理解5级流水线CPU工作原理的基础上，使用Verilog语言设计一款能够支持7条指令(add、addi、and、beq、j、lw、sw)的5级流水线CPU。本实验中，重点解决流水线中的数据冲突和控制冲突。

实验过程

1. 流水段划分

每条指令的执行经过需要5个时钟周期。具体时钟周期如下。

- 取指令周期(IF)。
- 指令译码/读寄存器周期(ID)。
- 执行/有效地址计算周期(EXE)。
- 存储器访问/分支完成周期(MEM)。
- 写回周期(WB)。

2. 流水段寄存器设计

同一时间各功能部件为不同的指令服务，这就需要识别各功能部件在当前时刻为哪条指令服务，这些功能通过插入流水段寄存器实现。流水段寄存器保存着从一个流水段传送到下一个流水段的所有数据和控制信息。PC多路选择器被移到IF段，这样做的目的是保证对PC值的写操作只出现在一个流水段内，否则当分支转移成功时，流水线中的两条指令都试图在不同的流水段修改PC值，从而发生写冲突。每个时刻每条指令都只在一个流水段上是活动的，因此，任何指令所做的任何动作都发生在一对流水段寄存器之间，具体操作由指令类型决定。各流水段寄存器保存的内容见表5.2。

表 5.2 各流水段寄存器保存的内容

流水段寄存器	寄存器保存的内容
IF/ID	PC+4、IR
ID/EXE	A、B、Ext(imm)、PC+4;rt/rd
EXE/MEM	PCbr、ALUout、结果状态;B、目的寄存器
MEM/WB	目的寄存器、ALUout 结果、存储器读出的结果

3. 确定数据通路

根据上述分析,5 级流水线 CPU 的数据通路就是在单周期 CPU 数据通路的基础上插入流水段寄存器实现的,具体设计可参考图 5.4 的结构。

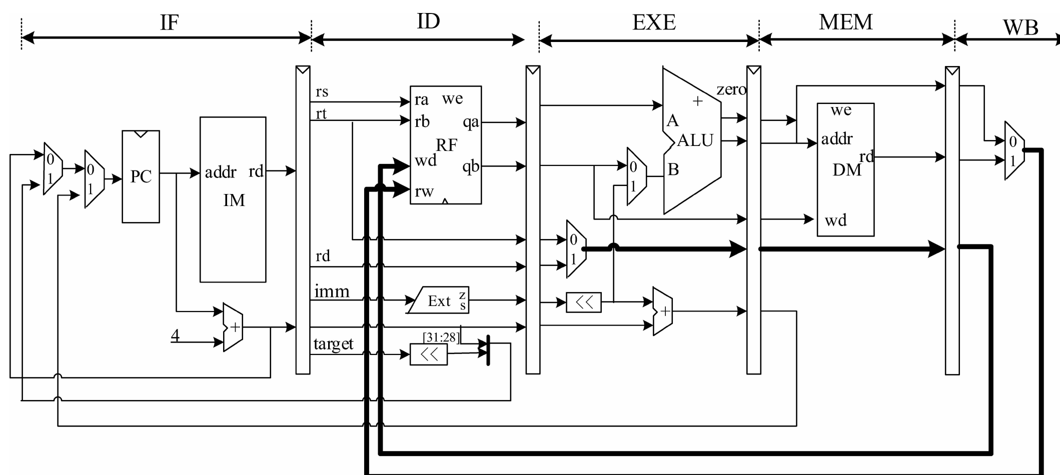

图 5.4 流水线 CPU 的数据通路

4. 流水线控制器设计

5 级流水线 CPU 的控制器设计和单周期完全一样,都是在译码阶段生成本条指令所需要的所有控制信号,只是流水线 CPU 在同一时间,不同的流水段为不同的指令服务,所以需要把控制信号和数据一样流动起来,使正确的控制信号到达正确的位置。

5. 冲突检测单元设计

因为相关的存在使得指令流中的下一条指令不能在指定的时钟周期执行,所以会引起流水线冲突。流水线冲突主要有 3 种类型:结构冲突、数据冲突和控制冲突。

MIPS 采用指令存储器和数据存储器分开的方式解决存储器访问的结构冲突,同时在寄存器文件设计时采用内部转发解决了读写同时发生的冲突。对于常见的写后读数据冲突,解决办法是采用重定向技术,将结果尽快传送到需要使用它的位置。当然,重定向技术解决不了的冲突只能暂停流水线。对于由程序执行转移类指令而引起的控制冲突,建议大家采用暂停流水线的方法使程序先运行起来,然后再考虑其他提高性能的方法。

6. 仿真测试

利用给定程序或自己编制一段程序,对设计的 5 级流水线 CPU 进行功能仿真。