上海交通大學

《计算机组成与系统结构》实验 6 报告

学生姓名　　：梁展诚 Zhin Sheng Leong

学生学院（系）　：船舶海洋与建筑工程学院（土木工程）

学生学号　　：517010990022

实验名称　　：简单的类 MIPS 多周期流水化处理器实现

目录

简单的类 MIPS 多周期流水化处理器实现

1. 实验目的

- 了解处理器流水线的结构、相关和冒险性 (*hazard*)
- 设计支持流水化的多周期 *MIPS* 处理器

2. 实验原理

2.1 流水化多周期处理器的设计

为了提高处理器的运行效率，往往会把处理器的工作分为多段，在段与段之间插入段寄存器，使各段在同一时间内能独立为不同的指令工作，实现多周期的处理器。本次实验将 *MIPS* 处理器分为 5 段处理，分别是取指阶段 *Instruction Fetch*、译码阶段 *Instruction Decode*、运算阶段 *Execute*、访存阶段 *Memory Access* 以及写回阶段 *Write Back*，如图 2 所示。

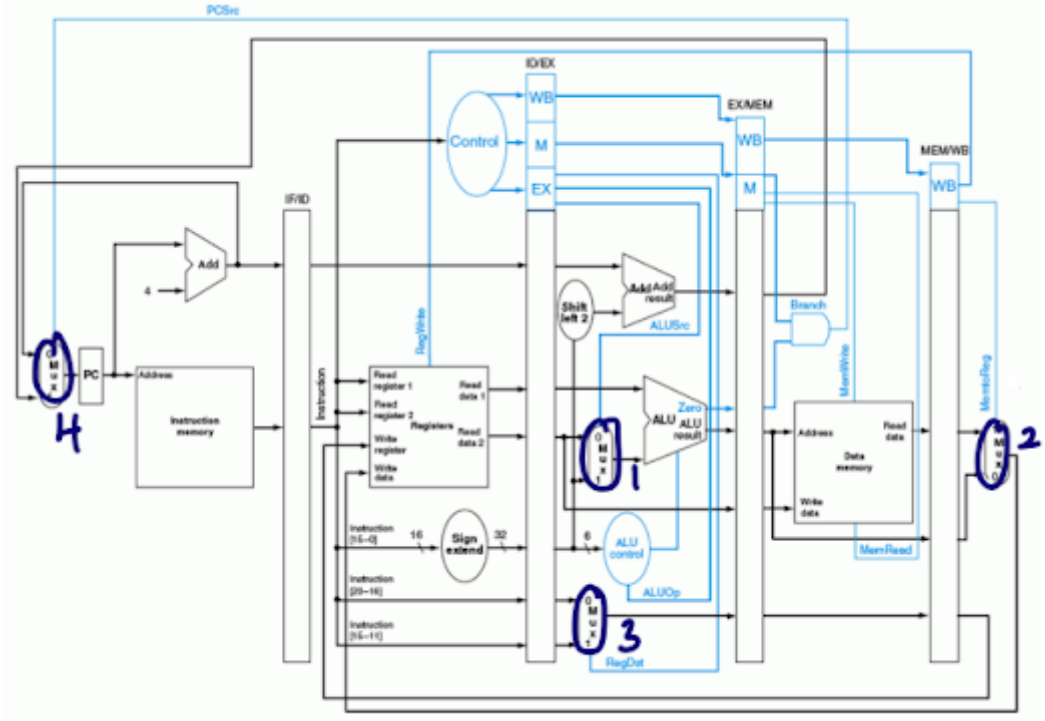

图 1：*MIPS* 多周期处理器原理图

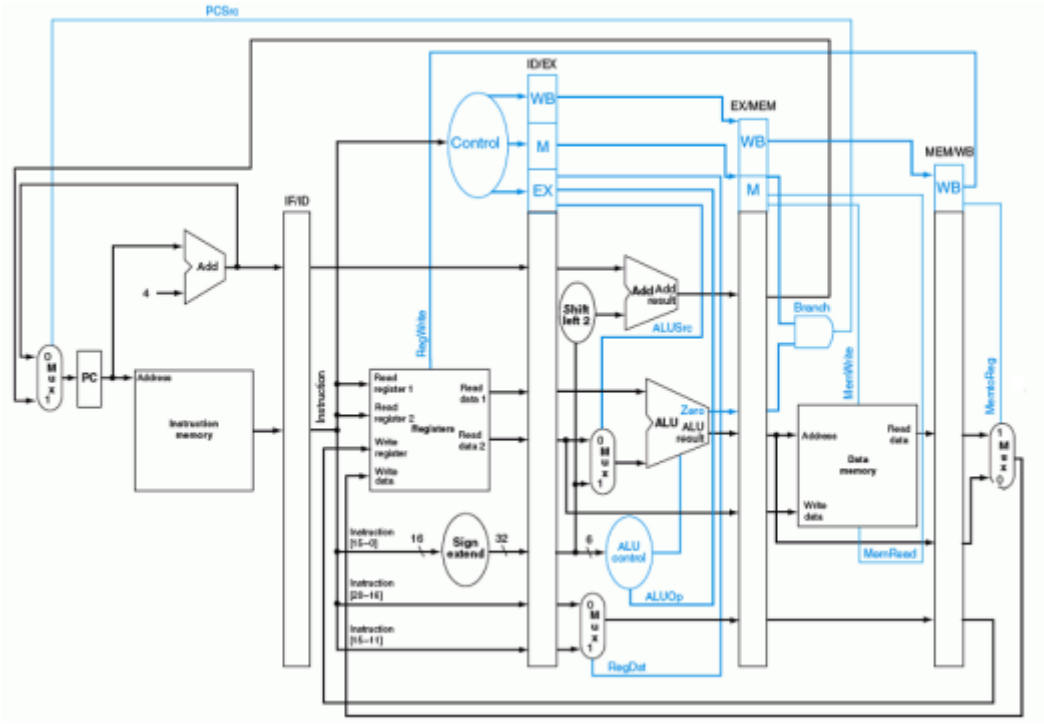

图 2：*MIPS* 多周期处理器的 5 大阶段

2.2 段寄存器的设计

每个段寄存器的大小不一，均与其应执行的操作和传递给下一段的数据和信号相关，具体信息参见表 1。不同类型的指令在各段寄存器的控制信号真值表参见表 2。

段寄存器名称	总位数	控制信号/数据	所占位数
IF/ID	64	*PC + 4*	32
		Instruction	32
ID/EX	147	*Instruction* [15:11]	5
		Instruction [20:16]	5
		RegDst	1
		AluSrc	1
		MemToReg	1
		RegWrite	1
		MemRead	1
		MemWrite	1

		$Branch$	1
		$ALUOp$	2
		$SignExt$	32
		$Read\ Data\ 2$	32
		$Read\ Data\ 1$	32
		$PC + 4$	32
EX/MEM	107	$Zero$	1
		$MemToReg$	1
		$Branch$	1
		$RegWrite$	1
		$MemWrite$	1
		$MemRead$	1
		$Write\ Register$	5
		$ALU\ Result$	32
		$Jump\ Address$	32
		$Read\ Data\ 2$	32
MEM/WB	71	$MemToReg$	1
		$RegWrite$	1
		$Write\ Register$	5
		$ALU\ Result$	32
		$Write\ Data$	32

表 1：段寄存的总位数及其所存放的信号与数据

指令类型		$R-format$	lw	sw	beq
$Execution\ stage$	$RegDst$	1	0	x	x
	$ALUOp1$	1	0	0	0
	$ALUOp0$	0	0	0	1
	$ALUSrc$	0	1	1	0
	$Branch$	0	0	0	1

Memory Access stage	*Mem Read*	0	1	0	0
	Mem Write	0	0	1	0
Write − back stage	*Reg Write*	1	1	0	0
	Mem to Reg	0	1	x	x

表 2：不同指令在各段寄存器的控制信号真值表

3. 实验设计

3.1 段寄存器的实现

段寄存器是使处理器能够流水化操作的关键，其存放着各段的控制信号及数据，使每一段在同一时间内能够执行不同的指令。此处设计的段寄存器是在时钟的下降沿才将相关信号及数据写入段寄存器内，具体实现段寄存器的 *Verilog* 代码如下：

（A）*IF/ID* 段寄存器：

```
module part_reg1(input clock, input reset, input [31:0] pcin, input [31:0] instin, output [63:0] result);
    reg [63:0] temp;

    always @(negedge clock)
    begin
     temp = {pcin+4,instin};
     if (reset)
         temp = 64'bx;
    end

    assign result = temp;

endmodule
```

（B） *ID/EX* 段寄存器：

```
module part_reg2(input clock, input reset, input [4:0] instlow, input [4:0] insthigh, input regdst, input alusrc, input memtoreg,
input regwrite, input memread, input memwrite, input branch, input [1:0] aluop, input [31:0] signext, input [31:0] readdata2, input [31:0] readdata1, input [31:0] pc,

output [146:0] result);

    reg [146:0] temp;

    always @(negedge clock)
    begin
```

```
9.          temp = {pc,readdata1,readdata2,signext,aluop,branch,memwrite,memread,regwri
    te,memtoreg,alusrc,regdst,insthigh,instlow};
10.         if (reset)
11.             temp = 147'bx;
12.     end
13.
14.     assign result = temp;
15.
16. endmodule
```

（C） *EX/MEM* 段寄存器：

```
1.  module part_reg3(
2.      input clock,
3.      input reset,
4.      input zero,
5.      input memtoreg,
6.      input branch,
7.      input regwrite,
8.      input memwrite,
9.      input memread,
10.     input [4:0] mux3,
11.     input [31:0] aluout,
12.     input [31:0] alujumpadd1,
13.     input [31:0] alujumpadd2,
14.     input [31:0] readdata2,
15.     output [106:0] result
16.     );
17.
18.     reg [106:0] temp;
19.     reg [31:0] temp2;
20.
21.     always @(negedge clock)
22.     begin
23.         temp2 = alujumpadd1<<2 + alujumpadd2;
24.         temp = {readdata2,temp2,aluout,mux3,memread,memwrite,regwrite,branch,memtor
    eg,zero};
25.         if (reset)
26.             temp = 107'bx;
27.     end
28.
29.     assign result = temp;
30.
31.
32. endmodule
```

（D） *MEM/WB* 段寄存器：

```
1.  module part_reg4(
2.      input clock,
3.      input memtoreg,
4.      input regwrite,
5.      input [31:0] aluout,
6.      input [31:0] readdata,
7.      input [4:0] mux3,
8.      output [70:0] result
9.      );
10.
11.     reg [70:0] temp;
12.
13.     always @(negedge clock) temp ={readdata,aluout,mux3,regwrite,memtoreg};
14.
```

```
    assign result = temp;

endmodule
```

3.2 流水化多周期处理器的实现

将先前的顶层模块导入到工程目录下，对其进行修改，并对以建的段寄存器模块进行实例化和逻辑上的连接。

```
module top_tb(
    );

    wire [63:0] part1out;
    wire [146:0] part2out;
    wire [106:0] part3out;
    wire [70:0] part4out;
    wire [31:0] pcout,inst,readdata1,readdata2,readdata,aluout,signext,mux1out,mux2out,mux4out;
    wire [4:0] mux3out;
    wire [3:0] aluctrout;
    wire regdst,branch,memread,memtoreg,memwrite,alusrc,regwrite,zero;
    wire reset2; //for branch = 1
    wire [1:0] aluop;

    PC mainpc (.in(mux4out),.clock(clock),.reset(reset),.result(pcout));

    InstMemory maininstmem(.ReadAddress(pcout),.Instruction(inst));

    part_reg1 mainpart_reg1(.clock(clock),.reset(reset2),.pcin(pcout),.instin(inst),.result(part1out));

    ctr mainctr(.clock(clock),.OpCode(part1out[31:26]),.reset(reset2),.RegDst(regdst),.ALUSrc(alusrc),
    .RegWrite(regwrite),.MemToReg(memtoreg),.MemRead(memread),.MemWrite(memwrite),.Branch(branch),.ALUOp(aluop));

    Registers mainregisters(.reset(reset),.clock(clock),.readReg1(part1out[25:21]),.readReg2(part1out[20:16]),.writeReg(part4out[6:2]),
    .writeData(mux2out),.RegWrite(part4out[1]),.ReadData1(readdata1),.ReadData2(readdata2));

    signext mainsignext(.clock(clock),.reset(reset),.inst(part1out[15:0]),.data(signext));

    part_reg2 mainpart_reg2(.clock(clock),.reset(reset2),.instlow(part1out[15:11]),.insthigh(part1out[20:16]),.regdst(regdst),.alusrc(alusrc)
    ,.memtoreg(memtoreg),.regwrite(regwrite),.memread(memread),.memwrite(memwrite),.branch(branch),.aluop(aluop),.signext(signext),
    .readdata2(readdata2),.readdata1(readdata1),.pc(part1out[63:32]+4),.result(part2out));

    mux1 mainmux1(.ALUSrc(part2out[11]),.ReadData2(part2out[82:51]),.signextout(part2out[50:19]),.result(mux1out));

    mux3 mainmux3(.RegDst(part2out[10]),.input1(part2out[4:0]),.input2(part2out[9:5]),.result(mux3out));

    ALUCtr mainaluctr(.clock(clock),.ALUOp(part2out[18:17]),.Funct(part2out[24:19]),.reset(reset),.ALUCtrOut(aluctrout));

    ALURes mainalures(.input1(part2out[114:83]),.input2(mux1out),.clock(clock),.reset(reset),.ALUCtr(aluctrout),.Zero(zero),.ALURes(aluout));
```

```
40.
41.     part_reg3 mainpart_reg3(.clock(clock),.zero(zero),.reset(reset),.memtoreg(part2
    out[12]),.branch(part2out[16]),.regwrite(part2out[13]),
42.     .memwrite(part2out[15]),.memread(part2out[14]),.mux3(mux3out),.aluout(aluout),.
    alujumpadd1(part2out[50:19]),.alujumpadd2(part2out[146:115]),
43.     .readdata2(part2out[82:51]),.result(part3out));
44.
45.     branch mainbranch(.branch(part3out[2]),.zero(part3out[0]),.result(reset2));
46.
47.     Memory mainmemory(.reset(reset),.clock(clock),.Address(part3out[42:11]),.WriteD
    ata(part3out[106:75]),.memWrite(part3out[4]),.memRead(part3out[5]),
48.     .readData(readdata));
49.
50.     part_reg4 mainpart_reg4(.clock(clock),.memtoreg(part3out[1]),.regwrite(part3out
    [3]),.aluout(part3out[42:11]),.readdata(readdata),.mux3(part3out[10:6]),
51.     .result(part4out));
52.
53.     mux2 mainmux2(.MemtoReg(part4out[0]),.MemoryResult(part4out[70:39]),.ALUResult(
    part4out[38:7]),.result(mux2out));
54.
55.     mux4 mainmux4(.pc(pcout),.jumpadd(part3out[74:43]),.PCSrc(reset2),.result(mux4o
    ut));
56.
```

4. 结果仿真

通过编写二进制测试程序，采用软件仿真的方式进行校核测试。通过 *Verilog* 编写相应的激励文件，进行仿真，观察其结果，与预期的结果进行对比。

4.1 指令模块、主控制寄存器模块、内存模块的初始化

对上述的各个模块编写初始化数据的文件，通过系统任务 $\$readmemb/\$readmemh$，在各模块内以二进制/十六进制的方式将相应文件中的数据读入模块内的寄存器。通过如下的 *Verilog* 代码实现其相应的功能：

```
1.  initial begin
2.      $readmemh("mem_data.mem",memFile);
3.  end
```

其余两个模块的初始化代码与上述相似，在此省略

初始化文件如下：

(A) 指令模块的初始化文件 $inst\_mem.mem$ ：

指令二进制的形式	具体含义
10001100111000010000000000000000	$lw\ (\$7+0),\$1$
10001101000000100000000000000000	$lw\ (\$8+0),\$2$
10001101001000110000000000000000	$lw\ (\$9+0),\$3$

10101101010001000000000000000000	$lw\ (\$10+0), \$4$
00000000001000100010100000100000	$add\ \$5, \$1, \$2$
00000000110001100100000000100010	$sub\ S8, \$6, \$6$
00000001001010010101000000100100	$and\ \$10, \$9, \$9$
10101101011001010000000000000000	$sw\ (\$11+0), \$5$
00000001000001100100100000100010	$sub\ \$9, \$8, \$6$
00000000101010000110000000100101	$or\ \$5, \$8, \$12$

表 3：指令模块的初始化文件

(B) 主控制寄存器模块的初始化文件 $mem\_reg.mem$ ：

数据的十六进制的形式	寄存器编号
00000000	1
00000001	2
00000002	3
00000003	4
……	……
0000001b	29
0000001c	30
0000001d	31
0000001e	32

表 4：主控制寄存器模块的初始化文件

(C) 内存模块的初始化文件 $mem\_data.mem$ ：

数据的十六进制的形式	内存地址
5672$efac$	1
6583fac9	2
12438efa	3
92740fe2	4
……	……
$bdefac$64	61
8493027d	62
01928372	63
be7320ed	64

表 5：内存模块的初始化文件

4.2 $MIPS$ 5 阶段流水化多周期处理器的仿真

对 PC 的值进行初始化（设置为 0），对 $MIPS$ 5 阶段流水化多周期处理器进行仿真，得出如下的仿真波形图：

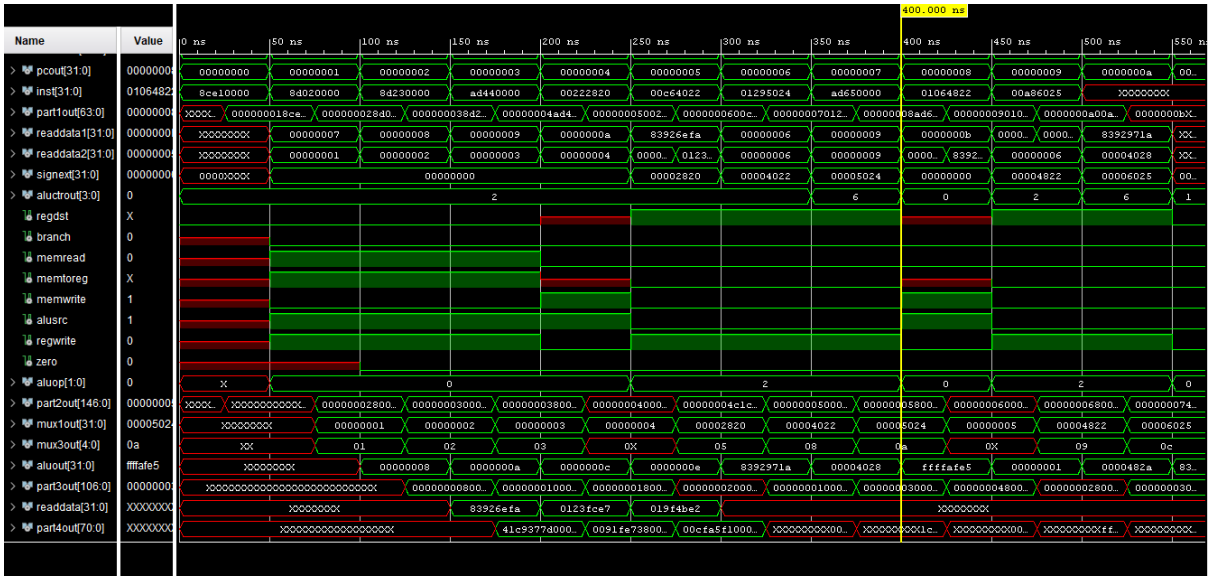

图 3：*MIPS* 5 阶段流水化多周期处理器的仿真结果

由上图可看出，在处理器刚开始工作的前 3 周期，其内部的各阶段正逐步填满指令，因此后续段寄存器（*ID/EX*、*EX/MEM*、*MEM/WB*）存在处于未知“x”状态的信号及数据。从第 4 周期开始，各阶段均已填满指令，处理器开始流水化的操作，各段在同一时间为不同的指令服务。对照自己的测试程序可知，波形正确，仿真结果正确。

5. 实验总结

本实验是在先前已设计 *MIPS* 处理器的基础上，通过增加段寄存器模块及修改部分代码内容，实现一个流水化多周期的处理器。由于存在大量的变量、线路、信号等，对其进行命名和整理时存在一定的难度，即使在开始编码之前已定义一套命名规范，但仍在逻辑上进行多次错误的拼接。本实验另一个难点在于如何体现所设计的处理器能够实现流水化的操作，经过一番思考后，决定在时钟的上升沿和下降沿，对不同的部件和段寄存器进行相应的操作，以体现各段能在同一时间为不同指令服务和工作。

由于疫情原因无法返校，这一系列的 *MIPS* 处理器设计实验只能靠自己通过软件的方式完成，而无法在实验室老师指导的氛围下完成、给予相应的评语、不能亲自动手进行上板调试，对我而言是一个遗憾。这几次的实验，带给我的收获的不仅仅是一门新的硬件设计语言的掌握、综合素质上的提升，更是让我意识到在在今后的学习和工作生涯中，用于探索、不畏失败、从错误中学习才是迈向成功的关键。