上海交通大學

《计算机组成与系统结构》实验 5 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 单周期处理器实现 ——整体调试

目录

1. 实验目的……2
2. 实验原理……2
2.1. 指令模块 Instruction Memory 的设计 ……2
2.2. 程序计数器 Program Counter 的设计……2
2.3. 重置信号 Reset ……2
2.4. MUX 模块的设计 ……3
2.5. 顶层模块的设计 ……3
3. 实验设计……4
3.1. 指令模块 Instruction Memory 的实现 ……4
3.2. 程序计数器 Program Counter 的实现……5
3.3. MUX 模块的实现……5
3.4. 顶层模块的实现 ……7
4. 结果仿真……9
4.1. 指令模块、主控制寄存器模块、内存模块的初始化 ……9
4.2. MIPS 单周期处理器的仿真 ……10
5. 实验总结 ……11

简单的类 *MIPS* 单周期处理器实现 ——整体调试

1. 实验目的

- 完成单周期类的 *MIPS* 的处理器
- 设计支持 16 条 *MIPS* 指令

  $(ADD, SUB, AND, OR, ADDI, ANDI, ORI, SLT, LW, SW, BEQ, J, JAL, JR, SLL, SRL)$

  的单周期处理器

2. 实验原理

2.1 指令模块 *Instruction Memory* 的设计

指令模块是程序在运行时其指令存放的单元部件，通过输入信号 *PC* (*Instruction Address*)，访问该模块内相应地址的指令，取出后送入主控制模块和后续部件进行相应的运算和操作。

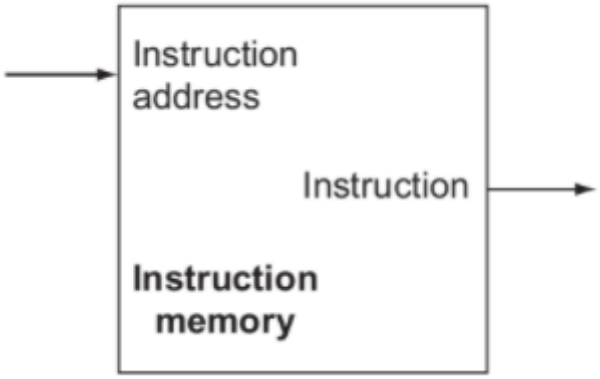

图 1：*MIPS* 指令模块 *Instruction Memory*

2.2 程序计数器 *Program Counter* 的设计

程序计数器 *PC* 是驱动一个处理器正常运行的核心部件，只有不断改变程序计数器的值，才能有次序地实现一系列的运算和操作，使一个程序运行起来。

2.3 重置信号 *Reset*

Reset 作为一个输入信号，当其处于高电平时，将 *PC* 重置为 $0x00000000$，并将处理器内的各寄存器清空置零。

2.4 *MUX* 模块的设计

本处理器存在 5 个 *MUX* 单元部件，本质上是一个三目运算符，通过主控制单元 *Control* 相应的输出信号作为输入信号，决定 2 个源操作数的选择。当 *SEL* 信号为高电平时，其输出为 *INPUT*1，反之为 *INPUT*2，其中$SEL, INPUT1, INPUT2$ 均为预先定义的信号。处理器内各 *MUX* 单元的具体输入输出信号关系参见表 3。

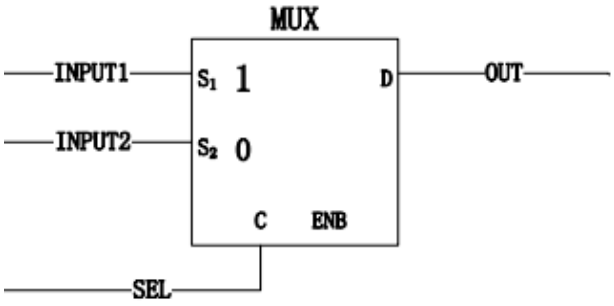

图 2：*MIPS* 处理器 *MUX* 模块

编号	*SEL*	*INPUT*1	*INPUT*2
MUX 1	*ALUSrc*	*SIGN EXTEND*	*READ DATA* 2 (*REGISTER*)
MUX 2	*MemtoReg*	*READ DATA* (*MEMORY*)	*ALU result*
MUX 3	*RegDst*	*Instruction* [15:11]	*Instruction* [20:16]
MUX 4	*Jump*	*Jump Address*	*MUX* 5
MUX 5	*Branch* & *Zero*	*SIGN EXTEND* $\ll 2 + (PC + 4)$	$PC + 4$

表 1：*MUX* 模块 *SEL* 信号与输出信号的关系表

2.5 顶层模块的设计

MIPS 作为一种 *RISC* 处理器，初始时 *PC* 的值设置为 0，通过不断重复地取指、译码、执行指令的动作以运行程序，完成各种不同类型的运算和操作。通过创建一个顶层模块，结合先前所设计的各个元件模块，将相关模块实例化并对其进行逻辑功能上的连接，实现一个简单的类 *MIPS* 单周期处理器。

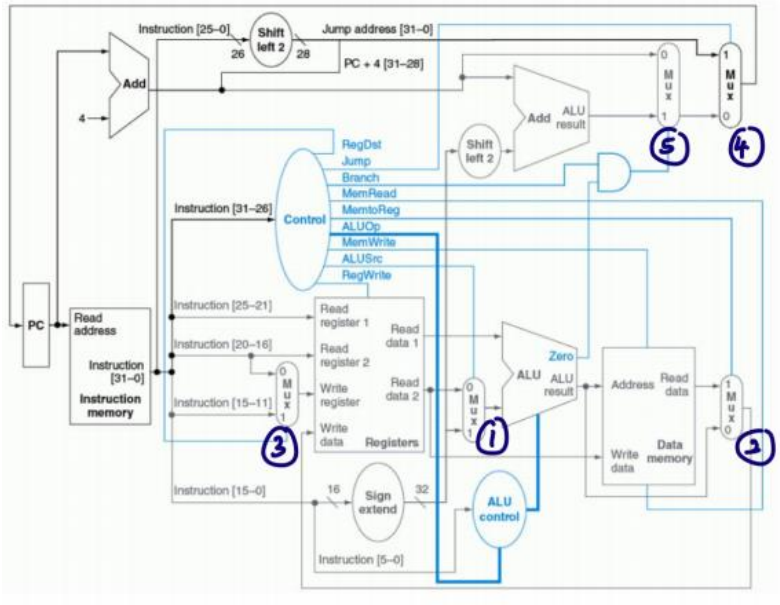

图3：*MIPS* 单周期处理器原理图

3. 实验设计

3.1 指令模块 *Instruction Memory* 的实现

指令模块只需将相应的输入信号作为其在模块内要访问的地址，将相关指令取出后作为输出信号即可。通过寄存器的方式实现其存放指令的功能，此处设计的指令模块内部的寄存器 *InstMemFile* 只能存放 32 条 32 位的指令，可通过 *Verilog* 中如下的代码实现其功能：

```
module InstMemory(
    input [31:0] ReadAddress,
    output [31:0] Instruction
    );

    reg [31:0] InstMemFile [31:0];

    initial begin
        $readmemb("inst_mem.mem",InstMemFile);
    end

    assign Instruction = InstMemFile[ReadAddress];

endmodule
```

代码中 *inst_mem.mem* 是初始化指令模块数据的文件，通过系统任务 *$readmemb* 将该文件中的数据以二进制的方式读取到指令模块的寄存器中，作为初始化的数据指令。

3.2 程序计数器 *Program Counter* 的实现

定义一个 32 位的寄存器，初始时将其值设置为 0，这里不将 $PC <= PC + 4$ 的功能在此模块内实现，而是在其他相应的模块内实现。实现其功能的 *Verilog* 代码如下：

```
1.  module PC(
2.      input [31:0] in,
3.      input clock,
4.      input reset,
5.      output [31:0] result
6.      );
7.
8.      reg [31:0] PC;
9.
10.     initial begin
11.         PC <= 0;
12.     end
13.
14.     always @(posedge clock)
15.     begin
16.         PC = in;
17.         if (reset)
18.             PC = 32'b0;
19.     end
20.
21.     assign result = PC;
22.
23. endmodule
```

3.3 *MUX* 模块的实现

MUX 实现简单，作为一个三目运算符，其逻辑表达式如下：

$$Assign\ OUT = SEL?\ INPUT1 : INPUT2$$

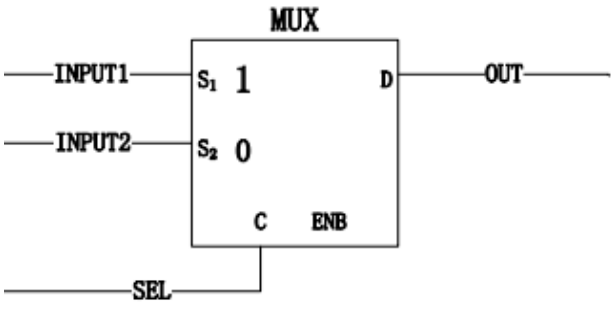

图 2：*MIPS* 处理器 *MUX* 模块

上述表达式可解释为当 *SEL* 信号为高电平时，输出信号为 *INPUT*1，反之为 *INPUT*2。可通过 *Verilog* 中如下的代码实现各 *MUX* 模块功能：

（A）*MUX* 1:

```
module mux1(
    input ALUSrc,
    input [31:0] ReadData2,
    input [31:0] signextout,
    output [31:0] result
    );

    assign result = ALUSrc? signextout:ReadData2;
Endmodule
```

（B）*MUX* 2:

```
module mux2(
    input MemtoReg,
    input [31:0] MemoryResult,
    input [31:0] ALUResult,
    output [31:0] result
    );

    assign result = MemtoReg? MemoryResult:ALUResult;
endmodule
```

（C）*MUX* 3:

```
module mux3(
    input RegDst,
    input [20:16] input1,
    input [15:11] input2,
    output [4:0] result
    );

    assign result = RegDst? input2:input1;

endmodule
```

（D）*MUX* 4:

```
module mux4(
    input Jump,
    input [25:0] inst,
    input [31:0] PC,
    input [31:0] mux5out,
    output [31:0] result
    );

    reg [31:0] JumpAddress;
    reg [31:0] temp;

    always @(*)
    begin
        temp = PC+4;
        JumpAddress = {temp[31:28],inst<<2};
    end

    assign result = Jump? JumpAddress:mux5out;

endmodule
```

（E）*MUX* 5:

```
module mux5(
    input [31:0] PC,
    input Branch,
    input Zero,
    input [31:0] signextout,
    output [31:0] result
    );

    reg [31:0] aluout;
    reg [31:0] temp;

    always @(*)
    begin
        temp = PC+4;
        aluout = temp[31:28]+signextout[25:0]<<2;
    end


    assign result = (Branch & Zero)? aluout:temp;

endmodule
```

从上述代码中可看出，$PC <= PC + 4$ 的操作在 $MUX\,4$ 及 $MUX\,5$ 模块内实现

3.4 顶层模块的实现

将先前所设计的各个单元模块添加到工程目录下，如下图所示：

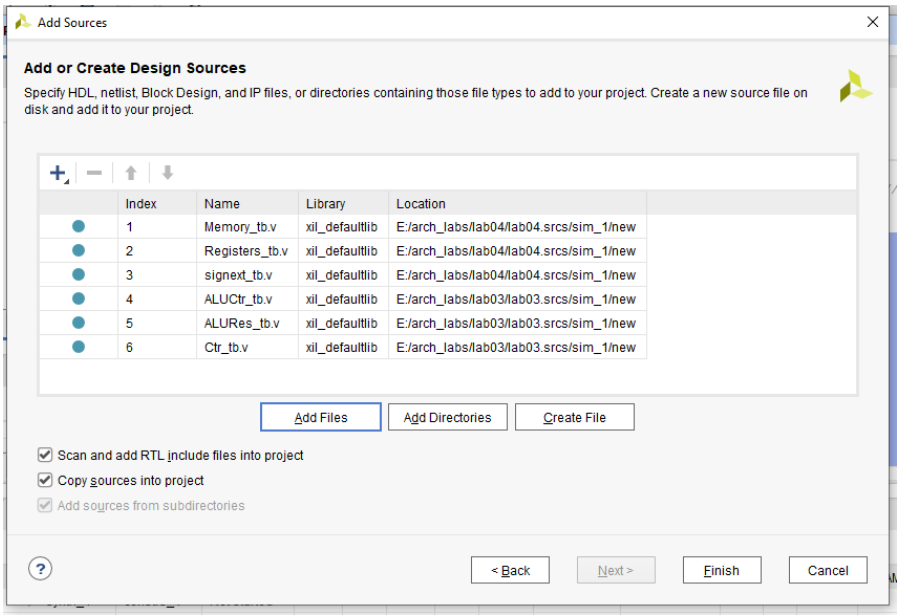

图 4：将先前相应的模块添加到工程目录下

为处理器的每一组连接信号线命名，并在顶层模块内声明定义它们，如下所示：

```
wire [31:0] PCwire;
wire [31:0] inst;
wire [31:0] aluout;
wire [31:0] ReadData1, ReadData2;
wire [31:0] memoryout;
wire [31:0] mux1out,mux2out,mux4out,mux5out;
```

```
wire [4:0] mux3out;
wire [31:0] signextout;
wire [3:0] aluctrout;
wire Zero;

wire RegDst;
wire Jump;
wire Branch;
wire MemRead;
wire MemtoReg;
wire [1:0] ALUOp;
wire MemWrite;
wire ALUSrc;
wire RegWrite;
```

将全部已设计好的单元模块实例化，采用如下的方法进行实例化：

在连接时用“.”符号，表明原模块是定义时规定的端口名，如：

模块　模块名（.端口 1 名（信号 1），.端口 2 名（信号 2）……）

各模块实例化的 *Verilog* 代码如下：

```
ALUCtr mainaluctr(.ALUOp(ALUOp),.Funct(inst[5:0]),.ALUCtrOut(aluctrout),.reset(reset));

Registers mainregister (.reset(reset),.clock(clock),.readReg1(inst[25:21]),.readReg2(inst[20:16]),
.writeReg(mux3out),.writeData(mux2out),.RegWrite(RegWrite),.ReadData1(ReadData1),.ReadData2(ReadData2));

ALURes mainalures (.input1(ReadData1),.input2(mux1out),.ALUCtr(aluctrout),.Zero(Zero),.ALURes(aluout),.reset(reset));

ctr mainctr (.OpCode(inst[31:26]),.RegDst(RegDst),.ALUSrc(ALUSrc),.RegWrite(RegWrite),.MemToReg(MemtoReg),
.MemRead(MemRead),.MemWrite(MemWrite),.Branch(Branch),.ALUOp(ALUOp),.Jump(Jump),.reset(reset));

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

Memory mainmemory(.clock(clock),.Address(aluout),.WriteData(ReadData2),.memWrite(MemWrite),
.memRead(MemRead),.readData(memoryout),.reset(reset));

mux1 mainmux1(.ALUSrc(ALUSrc),.ReadData2(ReadData2),.signextout(signextout),.result(mux1out));

mux2 mainmux2(.MemtoReg(MemtoReg),.MemoryResult(memoryout),.ALUResult(aluout),.result(mux2out));

mux3 mainmux3(.RegDst(RegDst),.input1(inst[20:16]),.input2(inst[15:11]),.result(mux3out));

mux5 mainmux5(.PC(PCwire),.Branch(Branch),.Zero(Zero),.signextout(signextout),.result(mux5out));

mux4 mainmux4(.Jump(Jump),.inst(inst[25:0]),.PC(PCwire),.mux5out(mux5out),.result(mux4out));

PC mainPC(.in(mux4out),.result(PCwire),.clock(clock),.reset(reset));

```

```
28. InstMemory maininstmemory(.ReadAddress(PCwire),.Instruction(inst));
```

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
10101100101001000000000000000000	$sw\ (\$5+0),\$4$
00001000000000000000000000000001	$jump$
00000000000000000000000000000000	nop
10101100111001010000000000000000	$sw\ (\$7+0),\$5$
10001100011001100000000000000010	$lw\ (\$3+2),\$6$
00000000100001010100000000100000	$add\ \$8,\$4,\$5$
00000001000001100100100000100010	$sub\ \$9,\$8,\$6$
00000001100101000010100000100100	$and\ \$5,\$12\ \$20$
10001101111010010000000000000000	$lw\ (\$15+0),\$9$
10101101001110000000000000000000	$sw\ (\$9+0),\$24$
00000010001101101111000000100101	$or\ \$30,\$17,\$22$
00000000001011010101100000100000	$add\ \$11,\$1,\$13$
00000011000010110110100000100000	$add\ \$13,\$24,\$11$
00000011110011011001100000100010	$sub\ \$19,\$30,\$13$

表 2：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 3：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 4：内存模块的初始化文件

4.2 $MIPS$ 单周期处理器的仿真

对 PC 的值进行初始化（设置为 0），对 MIPS 单周期处理器进行仿真，得出如下的仿真波形图：

https://wenku.baidu.com/?fr=logo

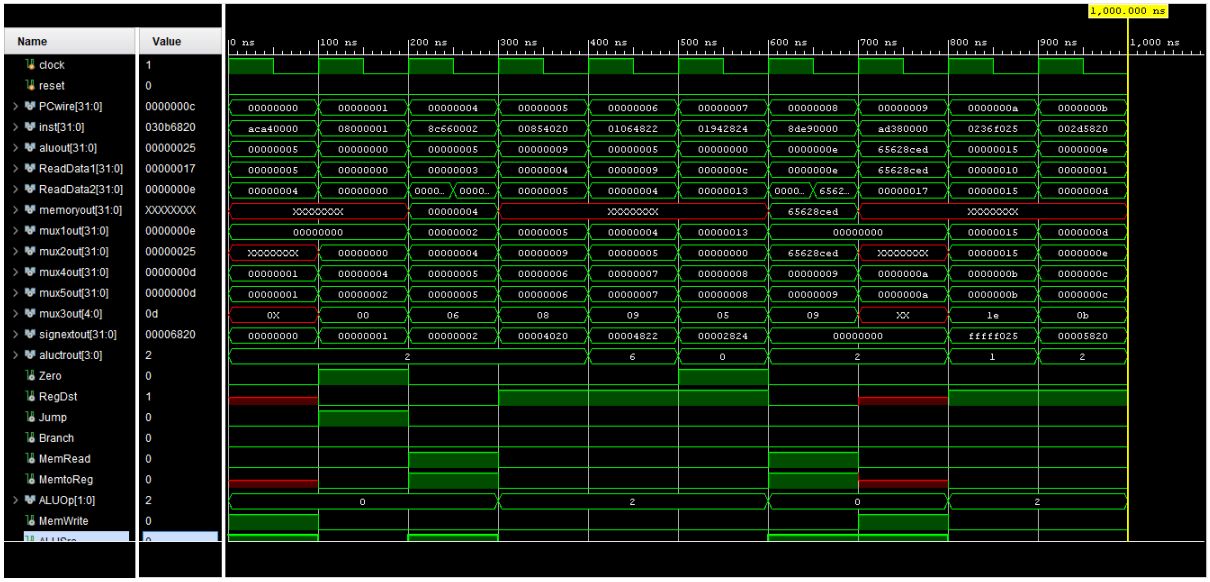

图 5：*MIPS* 单周期处理器的仿真结果

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

本次实验是前两次实验的一个综合，通过定义一个顶层模块、程序计数器以及指令模块单元，将所有已设计的单元模块进行实例化并结合，实现一个简单的类 *MIPS* 单周期处理器。由于本次实验指导书提供的资料比较有限，需要自己复习课堂所学的理论知识，再结合处理器的原理图思考各模块之间的逻辑关系才能完成，因此在设计和实现处理器的功能上比较吃力。这样的一个开发和设计过程培养了我系统化的思维，令我意识到设计过程中有着许多需要注意的细节和事项。通过本次实验，也让我认识到各种不同系统任务的调用及其含义，让我对开发软件和环境有了更深一层的了解和认识。