计算机系统结构实验报告　实验 5

颜培深 518030910094

2020 年 5 月 22 日

摘要

本实验实现了单周期类 MIPS 处理器，支持 16 条 MIPS 指令（包括 R 型指令：add、sub、and、or、slt、sll、srl、jr；I 型指令：lw、sw、addi、andi、ori、beq；J 型指令：j、jal），在继承了实验 3、4 中部分模块的基础上进行了功能修改以适配顶层组织文件，同时在原有 MIPS 接线基础上为了方便操作添加了部分辅助控制信号，实验通过软件仿真的方式进行结果验证。

目录

1 实验目的

1. 理解 MIPS 处理器的组织方式
2. MIPS 处理器多模块的设计调试
3. 调试顶层模块
4. 使用功能仿真验证实现的正确性

2 原理分析

2.1 主控制器-算术逻辑运算控制单元 (Ctr-ALUCtr) 的设计

主控制器 (Ctr) 以及算术逻辑运算控制单元 (ALUCtr) 的实现与实验 3 中基本一致，在细节上由于增加了一些指令，所以控制信号也相对增加了一些，同时，为了实现方便，这里我们将两个模块合二为一，根据解析的结果产生并输出对应的控制信号以及对应的 ALU 操作类型。本实验中用到的控制信号如表1所示。

信号名称	具体含义
ALUSrc	ALU 的第二个输入操作数来源（0：使用 rt 读出值；1：使用立即数）
RegWrite	寄存器写使能信号，高电平有效
RegDst	目标寄存器选择信号（0：写入 rt；1：写入 rd）
MemRead	内存读使能信号，高电平有效
MemWrite	内存写使能信号，高电平有效
MemToReg	选择是否将内存读取内容写入寄存器（0：不写回；1：写回）
Branch	条件跳转使能信号，高电平有效
Jump	无条件跳转使能信号，高电平有效
Jr	读寄存器跳转使能信号，高电平有效
Shift	逻辑位移操作选择信号（0：写入 rt 读出值；1：写入 inst[10:6]）

表 1: 主控制器-算术逻辑运算控制单元发出的控制信号含义

非 R 型指令对应的控制信号可以完全由指令 OpCode 域决定，对应关系如表5所示。

OpCode	$6'b001000$	$6'b001100$	$6'b001101$	$6'b100011$	$6'b101011$	$6'b000100$	$6'b000010$
指令类型	addi 指令	andi 指令	ori 指令	lw 指令	sw 指令	beq 指令	j 指令
ALUSrc	1	1	1	1	1	0	0
RegWrite	1	1	1	1	0	0	0
RegDst	0	0	0	0	0	0	0
MemWrite	0	0	0	0	1	0	0
MemToReg	0	0	0	1	0	0	0
Branch	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
Jr	0	0	0	0	0	0	0
Shift	0	0	0	0	0	0	0
ALUCtrOut	$4'b0010$	$4'b0000$	$4'b0001$	$4'b0010$	$4'b0010$	$4'b0110$	$4'b0010$

表 2: 主控制器-算术逻辑运算控制单元发出的控制信号对应关系

这里输出信号为四位功能信号 ALUCtrOut，该信号取值与 ALU 功能对应关系如表3所示。

ALUCtrOut 取值	ALU 功能
0000	与运算
0001	或运算
0010	加法运算
0011	逻辑左移运算
0100	逻辑右移运算
0101	jr 操作运算
0110	减法运算
0111	小于时置位

表 3: ALUCtrOut 信号含义

R 型指令	funct 域	ALUCtrOut
and	100100	0000
or	100101	0001
add	100000	0010
sll	000000	0011
srl	000010	0100
jr	001000	0101
sub	100010	0110
slt	101010	0111

表 4: R 型指令对应 ALUCtrOut 输出

由于仅凭借指令 OpCode 域无法将 R 型指令具体内容进行解析，所以在 R 型指令的判断部分，利用指令 funct 域进行分类，对 ALUCtrOut 输出值进行更新，同时，当控制器判定该指令为 jr 指令时，对应的 Jr 控制信号置为 1，当控制器判定该指令为 sll 或 srl 指令（即为逻辑位移指令）时，对应的 Shift 控制信号置为 1。R 型指令判断对应关系如表4所示。

2.2 算术逻辑运算单元 ALU 的设计

与实验 3 实现基本一致，ALU 的主要功能是根据算术逻辑运算控制单元输出的 ALUCtrOut 信号，对两个输入操作数进行相应的算术逻辑运算，并输出运算结果与部分控制信号（如 zero 信号：运算结果为 0 时信号为高电平），只是在原有运算类型的基础上 ALU 的算术逻辑运算类型如表3所示。

2.3 寄存器 (Register) 的设计

与实验 4 中寄存器唯一不同之处就是在完整 MIPS 处理器中寄存器文件可以在 Reset 信号高电平状态时进行重置清零的操作，所以在输入信号中增加一个 reset 端，在接收到重置信号后将 32 个寄存

器全部赋值为 0. 其余细节不再赘述。

2.4 数据存储器 (Data Memory) 以及有符号拓展 (Signal Extend) 的设计

该部分设计与实验 4 中内容完全一致，不再展开。

2.5 指令存储器 (Instruction Memory) 的设计

该部分仅为简单的存储模块封装，对于任意时刻输入的 32 位指令地址 instAddr，都将对应位置存储的指令（长度为 32 位）输出，细节过于简单不再描述。

2.6 PC 的设计

PC 的功能是记录当前执行指令执行情况（即当前执行指令的地址），并在时钟周期上升沿进行一次更新操作，将输入的 PC 值记录，输出端始终保持与记录数据一致，这里在 PC 模块内部我们不关心输入数据的来源，具体 PC 赋值的来源由下文中 section**??**模块中的代码给出。

2.7 数据选择器 (Mux) 的设计

本实验中数据选择器分为 5 位数据选择器以及 32 位数据选择器两种，两种数据选择器都是根据选择控制信号 signal 的控制下，选择将两个输入数据之一接通到输出端口。
在单周期类 MIPS 处理器中，我们利用这两种数据选择器的模板构建了多种数据选择器，名称及用途如表**??**所示。

数据选择器名称	选择控制信号	用途
Branch_selector	Branch	用于选择输入 PC 的值是否为 branch 地址
J_or_Jr	Jr	用于在无条件跳转情况下选择输入 PC 的值是 jump/jr 指令来源
Jump_selector	Jump	用于选择输入 PC 的值是否为无条件跳转地址
Mem_Reg_selector	MemToReg	用于选择送给写入寄存器流程的值是 ALU 运算结果或访存结果
Reg_Imm_selector	ALUSrc	用于选择 ALU 二号输入数据来自寄存器 rt 读取数据或拓展立即数
Rt_selector	RegDst	用于选择写入寄存器地址为 rt 或 rd
Shift_selector	Shift	用于选择 ALU 一号输入数据来自寄存器 rs 数据或位移量

表 5: 各数据选择器用途

2.8 顶层模块 (Top) 的设计

顶层模块的功能是在功能仿真前对所有部件进行实例化组装，将上述的所有部件模块输入输出端利用本模块中定义的 wire 接线以及 reg 变量连接起来，同时对一些输出地址在本模块中进行运算后赋值给部分模块的输入端，完成一部分计算工作。

3 功能实现

3.1 主控制器-算术逻辑运算控制单元 (Ctr-ALUCtr) 的实现

主控制器-算术逻辑运算控制单元的功能是利用指令 OpCode 域以及 funct 域进行指令操作解析，可以利用 Verilog 中 case 语句进行实现，R 型指令部分代码如下，具体详见工程文件：

```
always@(*) begin
    Shift = 0;
    case(opCode)
    6'b000000: //R type
     begin
         RegDst = 1;
         ALUSrc = 0;
         MemToReg = 0;
         RegWrite = 1;
         MemWrite = 0;
         Branch = 0;
         Jump = 0;
         JumpMux = 0;
         case (Funct)
             6'b100000: ALUCtrOut = 4'b0010;//add 2
             6'b100010: ALUCtrOut = 4'b0110;//sub 6
             6'b100100: ALUCtrOut = 4'b0000;//and 0
             6'b100101: ALUCtrOut = 4'b0001;//or 1
             6'b101010: ALUCtrOut = 4'b0111;//slt 7
             6'b000000: begin ALUCtrOut = 4'b0011;//sll 3
             Shift = 1;
             6'b000010: ALUCtrOut = 4'b0100;//srl 4
             Shift = 1;
             6'b001000: begin
               ALUCtrOut = 4'b0101;//jr 5
                 RegDst = 0;
                 ALUSrc = 0;
                 MemToReg = 0;
                 RegWrite = 0;
                 MemWrite = 0;
                 Branch = 0;
                 Jump = 1;
                 JumpMux = 1;
             end
         endcase
     end
```

```
    ...
    ...
    endcase
end
```

由于我们将主控制器与算术逻辑运算控制单元合并实现，所以这里的输出端直接变成了 ALUCtrOut 的四位 ALU 操作类型编码。

3.2 算术逻辑运算单元 ALU 的实现

由于这里只是进行了 ALU 运算操作的拓展，语法逻辑等方面没有发生变化，详见工程文件，这里只给出小部分示例代码：

```
always @ (input1 or input2 or aluCtr) begin
    if ( aluCtr == 4'b0000)//and
    begin
        ALURes = input1 & input2;
    end
    else if(...)
    ...
    ...
    ...
    if(ALURes == 0)
        zero = 1;
    else
        zero = 0;
end
```

3.3 寄存器 (Register) 的实现

由于本实验中寄存器文件添加了重置功能，输入端增加了 reset 接口，实现代码如下：

```
always @(posedge Clk or Ret) begin
    if(Ret)
       for(i = 0;i <= 31;i = i+1)
           regFile[i] = 32'h00000000;
    else if(regWrite)
        regFile[writeReg] = writeData;
end
always @(readReg1 or readReg2) begin
    ReadData1 = regFile[readReg1];
    ReadData2 = regFile[readReg2];
end
```

3.4 数据存储器 (Data Memory) 以及有符号拓展 (Signal Extend) 的实现

本部分实现方式与实验 4 完全一致，不再赘述。

3.5 指令存储器 (Instruction Memory) 的实现

由于指令计算以字 (Byte) 为单位，而指令存储在实现中我设置了 4Byte 对齐（即每次取值都为 32 位二进制编码），所以我们在进行指令地址换算时直接除 4，并且将整个 32 位指令编码送入输出端，实现代码如下：

```
reg[31:0] instFile[0:1023];
assign inst = instFile[instAddr/4];
```

3.6 PC 的实现

PC 模块在时钟周期的上升沿进行数值更新，当重置信号为高电平时，PC 寄存器置为零，实现代码如下：

```
always @(posedge clk or ret) begin
    if(!ret)
        pc_out = pc_in;
    else
        pc_out = 0;
end
```

3.7 数据选择器 (Mux) 的实现

数据选择器在选择控制信号的控制下对两个输入值之一进行选择并送入输出端，直接利用 Verilog 中 if-else 语句实现，实现代码如下：

```
always @(input1 or input2 or signal) begin
    if(signal == 0)
        output_data = input1;
    else
        output_data = input2;
end
```

我们只需在顶层模块中进行实例化时对输入输出端位数进行分别确认即可使用，两种类型分别为 32 位数据选择器和 5 位数据选择器。

3.8 顶层模块 (Top) 的实现

顶层模块中仅对各模块实例化并阐明了他们之间的接线关系，具体细节见工程文件，这里不再赘述。

4 结果验证（单周期类 MIPS 处理器的测试）

对于本实验中实现的单周期类 MIPS 处理器我们采用软件仿真的方法进行测试，用 Verilog 语言编写测试激励文件，进行仿真，主要观察寄存器以及存储器信号波形变化，并通过我在各模块中内置的一些输出语句（$display(...);），观察 Tcl Console 输出框内的输出，比对是否符合预期。

在仿真激励文件中，我们通过 Verilog 指令直接将指令存储器文件以及数据存储器文件读入到对应存储位置，这里我们给出测试用指令文件以及对应的测试内容，如表6所示。

指令字	具体含义
10001100000000010000000000001010	lw $1, 10($0)
10001100000000010000000000001011	lw $2, 11($0)
00000000001000100001100000100000	add $3, $1, $2
00000000001000100010000000100010	sub $4, $1, $2
00000000001000100010100000100100	and $5, $1, $2
00000000001000100011000000100101	or $6, $1, $2
00000000001000100101000000101010	slt $10, $1, $2
00000000000000001010110001000000	sll $11, $1, 2
00000000000000001011000001000010	srl $12, $1, 2
00001000000000000000000000001011	j 11
00000011111100000000000000001000	jr $31
00001100000000000000000000001100	jal 12
10101100000000010000000000001010	sw $1,10($0)
00010011111111110000000000000000	beq $31, $30, 0
00100000001001110000000000000010	addi $7, $1, 2
00110000001010000000000000000010	andi $8, $1, 2
00110100001010010000000000000010	ori $9, $1, 2

表 6: 指令字及对应含义

根据如上述指令字序列执行仿真程序得到的程序输出如图1所示；仿真波形结果如图2、图3所示。

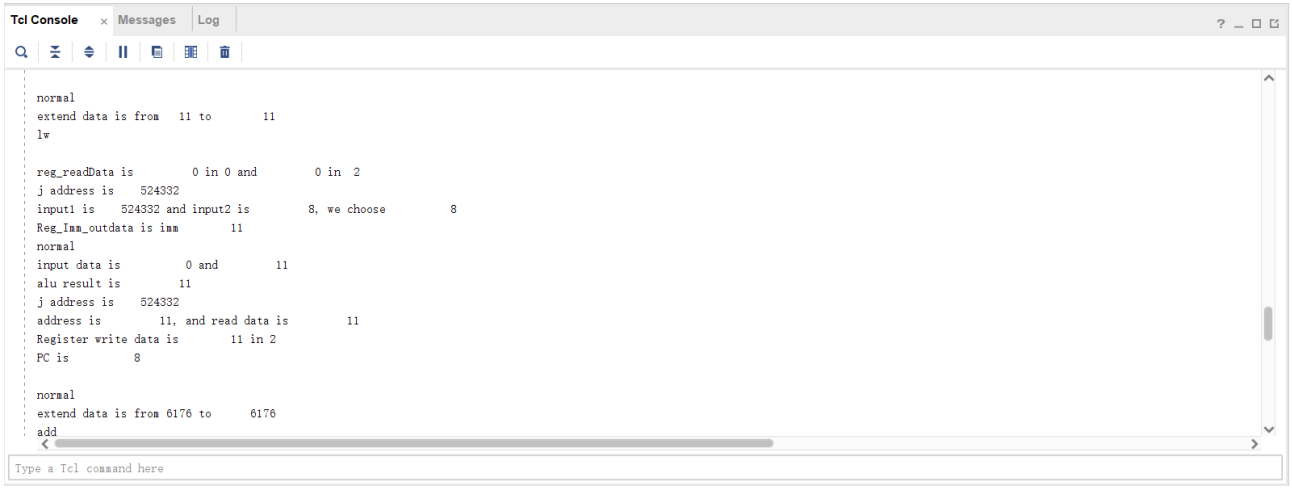

图 1: 仿真程序部分输出

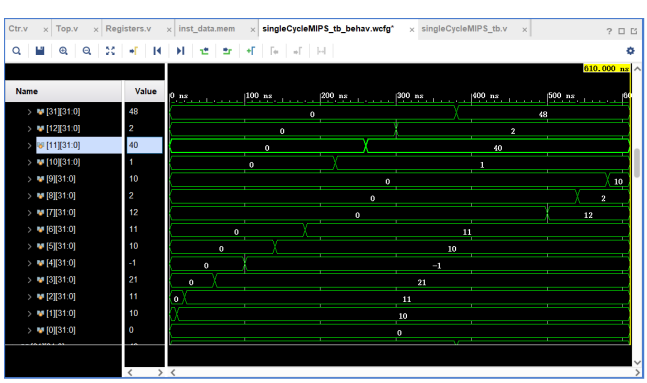

图 2: 单周期处理器仿真波形-1

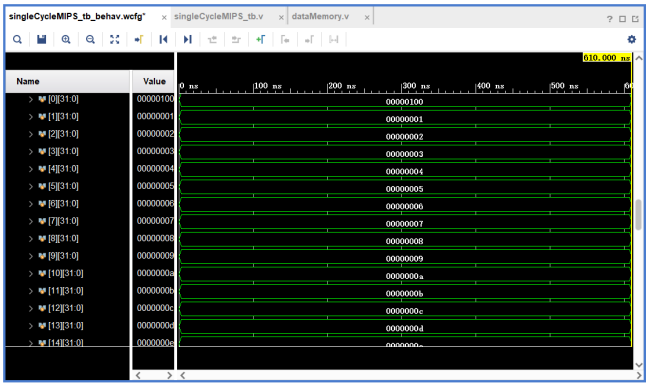

图 3: 单周期处理器仿真波形-2

另外想要测试 jr 指令时，仅需将 jump 指令跳转地址从 44 更改为 40，即将指令字中 1011 更改为 1010，修改后的仿真程序输出如图4所示；仿真波形结果如图5、图6所示。

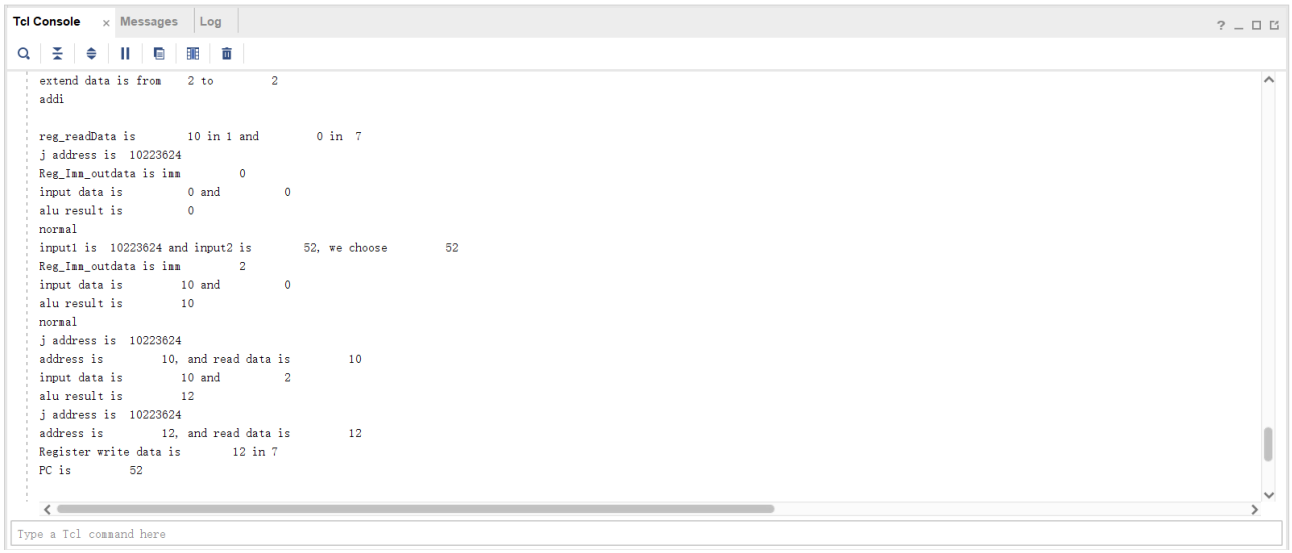

图 4: 修改后仿真程序部分输出

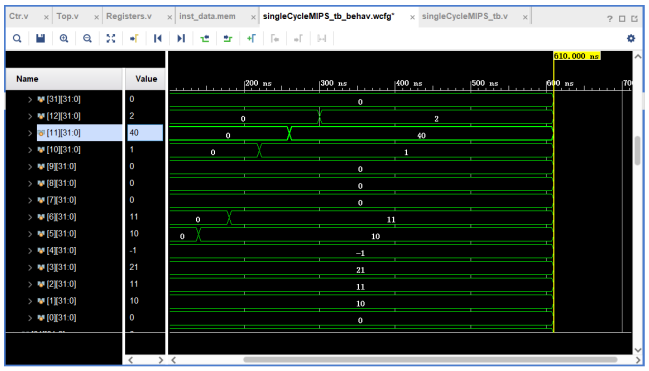

图 5: 单周期处理器仿真波形-3

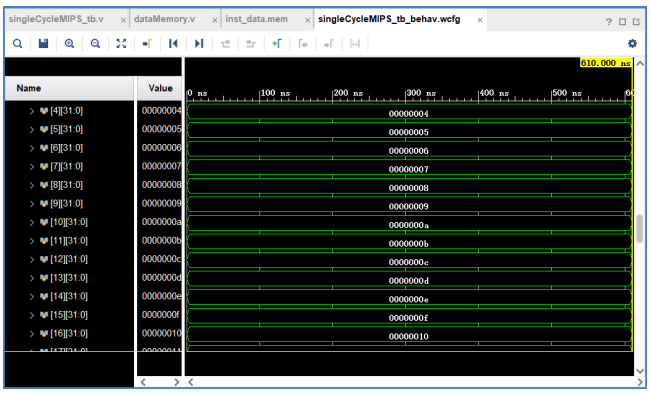

图 6: 单周期处理器仿真波形-4

比照以上六张图以及程序的完整输出逻辑，验证仿真实验功能完全成功。

5 总结与反思

本实验完整实现了支持 16 指令的单周期类 MIPS 处理器的仿真，通过对十四个模块的组装接线，加深了我对单周期处理器运行过程中，各数据流向（即数据通路），各控制信号的变化，多个处理模块之间的相互协作以及其具体功能的实现的了解。另外由于我在最后的功能仿真中，直接对整个处理器的功能实现进行仿真测试，在最初的时候虽然通过了编译，但是数据的流向以及选择器的选择都存在很大的问题，在利用程序输出的信息对每一步的处理器动作有了具体的了解以后，我逐渐找到了问题所在，并且从中更加体会到到处理器的运行机制。但是不足的是，由于主控制器与算数运算逻辑控制单元实现放在了一个单元中，导致这种实现难以拓展到流水线处理器的实现中，是一大隐患，但是多个数据选择器的使用将数据的来源去向标注的较为明确，确实方便了我在最终修改代码过程中的订正与优化。