2022《数字逻辑与处理器基础》处理器大作业

第二部分

无 08 刘星雨 2020010850

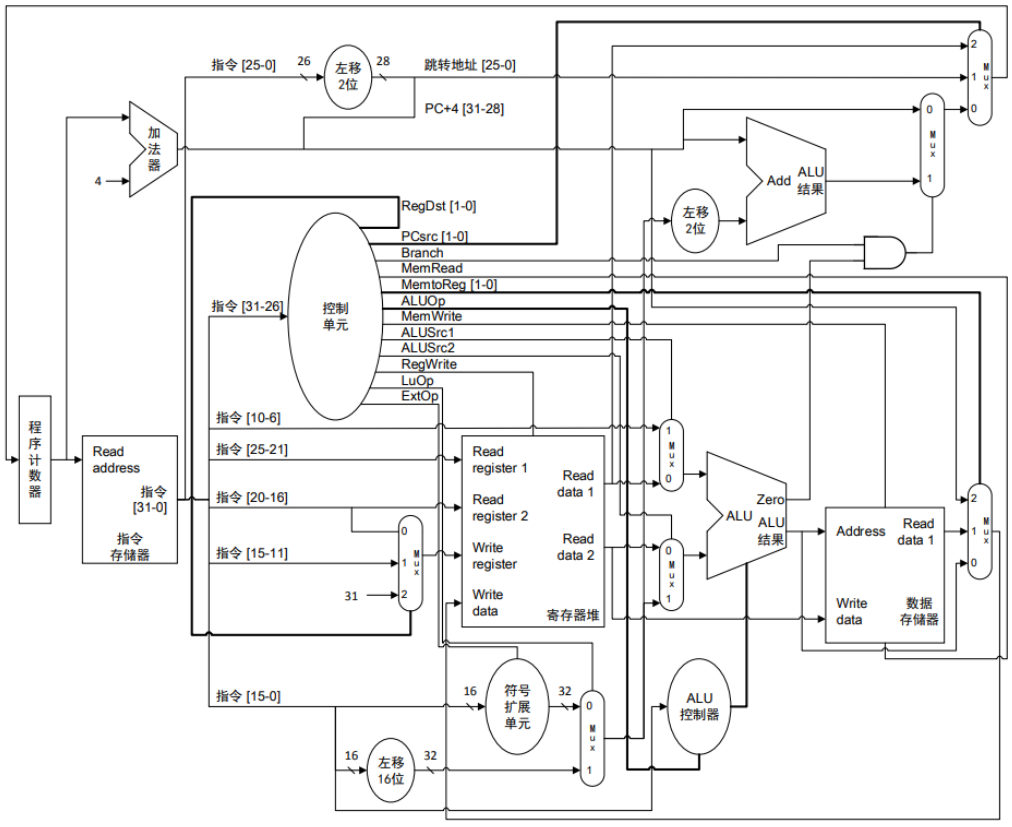
实验目的

1. 掌握 MIPS 单周期处理器的控制通路和数据通路的设计原理和 RTL 实现方式
2. 掌握 MIPS 多周期处理器的控制通路和数据通路的设计原理和 RTL 实现方式
3. 深入理解 MIPS 单周期和多周期哦处理器在资源和性能上的折衷设计

实验内容：

1.

（a）控制器模块设计：



PCSrc： 0→pc+4;1→pc+imm;2→pc=$r

Brach:0→no 1→yes

RegWrite:1→write to register;0→do not

RegDst:0→rt;1→rd;2→jal ;x→do not write register

MemRead:0→read 1→do not read

MemWrite:0→do not write 1→write

MemroReg:0→from memory 1→from alu;2→pc+4;x→do not write register

ALUSrc1:0→register ;1→imm;x→no alu calculate

ALUSrc2:0→register ;1→imm;x→no lau calulate

Extop:0→no sign 1→with sign x→do not need
LuOp:0→extend;1→<<16

Instruction	PCSrc[1:0]	Branch	RegWrite	RegDst[1:0]	MemRead	MemWrite	MemtoReg[1:0]	ALUSrc1	ALUSrc2	Extop	LuOp
lw	0	0	1	0	1	0	1	0	1	1	0
sw	0	0	0	x	0	1	x	0	1	1	0
lui	0	0	1	0	0	0	0	0	1	x	1
add	0	0	1	1	0	0	0	0	0	x	x
addu	0	0	1	1	0	0	0	0	0	x	x
sub	0	0	1	1	0	0	0	0	0	x	x
subu	0	0	1	1	0	0	0	0	0	x	x
addi	0	0	1	0	0	0	0	0	1	1	0
addiu	0	0	1	0	0	0	0	0	1	1	0
and	0	0	1	1	0	0	0	0	0	x	x
or	0	0	1	1	0	0	0	0	0	x	x
xor	0	0	1	1	0	0	0	0	0	x	x
nor	0	0	1	1	0	0	0	0	0	x	x
andi	0	0	1	0	0	0	0	0	1	0	0
sll	0	0	1	1	0	0	0	1	0	x	x
srl	0	0	1	1	0	0	0	1	0	x	x
sra	0	0	1	1	0	0	0	1	0	x	x
slt	0	0	1	1	0	0	0	0	0	x	x
sltu	0	0	1	1	0	0	0	0	0	x	x
slti	0	0	1	0	0	0	0	0	1	1	0
sltiu	0	0	1	0	0	0	0	0	1	1	0
beq	0	1	0	x	0	0	x	0	0	1	0
j	1	x	0	x	0	0	x	x	x	x	x
jal	1	x	1	2	0	0	2	x	x	x	x
jr	2	x	0	x	0	0	x	x	x	x	x
jalr	2	x	1	1	0	0	2	x	x	x	x

(b)数据通路设计:

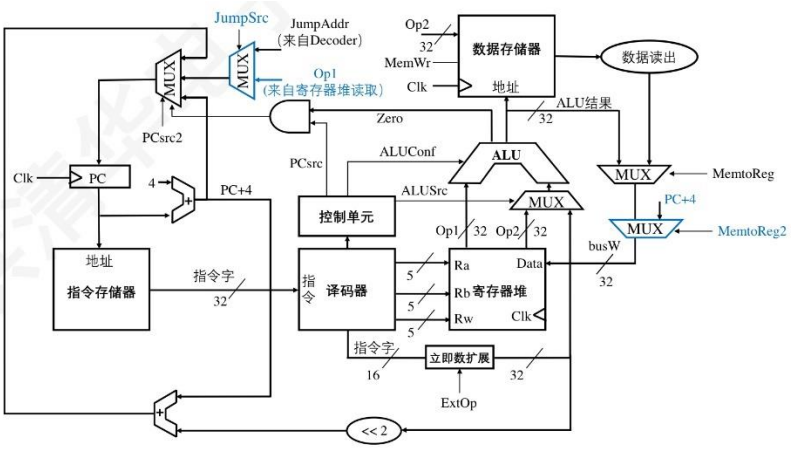


图 VI-19 支持 jal 的数据通路

？这个图是不是有些问

PCSrc 控制的 mux

代码：

assign PC_next=(PCSrc==2'b00)?Branch_target:(PCSrc==2'b01)?Jump_target:readdata1;

意义：下一个指令地址是：

00—beq 的 branch 或者是 pc+4， 01—jump 的地址，10—寄存器的地址

Branch&zero 控制的 mux

代码：

assign Branch_target=(Branch&zero)?pc_4+{luires[29:0],2'b00}:pc_4;

意义：pc 其实一共有 4 个可能，branch 合并了其中两个，1—相对地址寻址，0—pc+4

RegDst 控制的 mux：

代码：

Assign Writeregister=(RegDst==2'b00)?Instruction[20:16]: (RegDst==2'b01)?
Instruction[15:11]:5'b11111;

意义：

00--I 型指令$rt，01—R 型指令的$rd，10—jal 会把 pc 地址存在$31

MemRead 控制的 mux；

assign Read_data = MemRead? RAM_data[Address[RAM_SIZE_BIT + 1:2]]: 32'h00000000;

意义：

是否进行 memory 的读取，1-是，0-否

MemWrite 并不是控制 mux

MemtoReg[1:0]控制的 mux

assign writedata=(MemtoReg==2'b00)?aluout:(MemtoReg==2'b01)?memreaddata:pc_4;

意义：把什么内容写回寄存器 00—alu 计算的结果，01—从储存器中读出来的数据，10—pc+4

ALUScr1 控制的 mux

assign aluin1=ALUSrc1?{17'h00000,Instruction[10:6]}:readdata1;
意义：Alu 的第一个计算对象是扩展的常数还是寄存器读出的数据

ALUScr2 控制的 mux
assign aluin2=ALUSrc2?luires:readdata2;
意义：alu 的第二个计算对象是 lui 的结果还是寄存器读出的数据

ExtOp 并非是控制 mux

LuOp 控制的 mux
assign luires=LuOp?{Instruction[15:0],16'h0000}:extendresult;
意义：后续的计算是用 lui 的值还是符号扩展的值

（c）汇编程序分析-1

阅读 instructionmemory 的指令代码，结合注释理解存储器中的指令程序，对应的 MIPS

```
addi $a0, $zero, 12123          #(0x2f5b)
// addiu $a1, $zero, -12345 #(0xcfc7)
// sll $a2, $a1, 16
// sra $a3, $a2, 16
// beq $a3, $a1, L1
// lui $a0, 22222 #(0x56ce)
// L1:
// add $t0, $a2, $a0
// sra $t1, $t0, 8
// addi $t2, $zero, -12123 #(0xd0a5)
// slt $v0, $a0, $t2
// sltu $v1, $a0, $t2
// Loop:
// j Loop
```

分析过程：
A0=12123;　　(0x2f5b)
A1=-12345;　　（0xffffcfc7）
A2=a1<<16;　　(0xcfc70000)
A3=a2/2^16;　　(0xffffcfc7)
If a3==a1:　　//是相等的
　　T0=a2+a0;　　0xcfc72f5b 809029797
　　T1=t0*2^8;　　0xffcfc72f
　　T2=-12123;　　0xffffd0a5
　　If a0<t2:
　　　　V0=1;
　　Else
　　　　V0=0;　走这个 v0=0
　　If |a0|<|t2|:
　　　　V0=1;走这个 v1=1

Else

V0=0;

一直 loop

Loop 的数据不会再变化了

最终的数据:

$a0:0x00002f5b

$a1:0xffffcfc7

$a2:0xcfc70000

$a3:0xffffcfc7

$t0:0xcfc72f5b

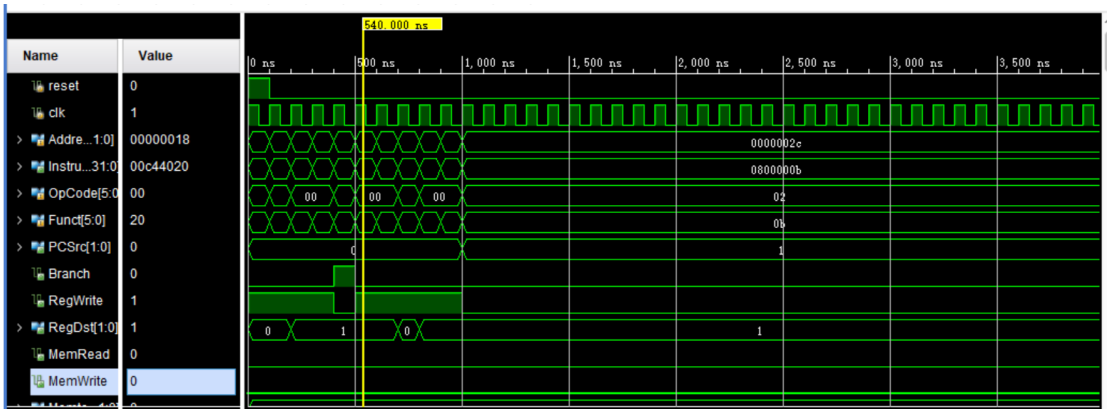
$t1:0xffcfc72f

$t2:0xffffd0a5

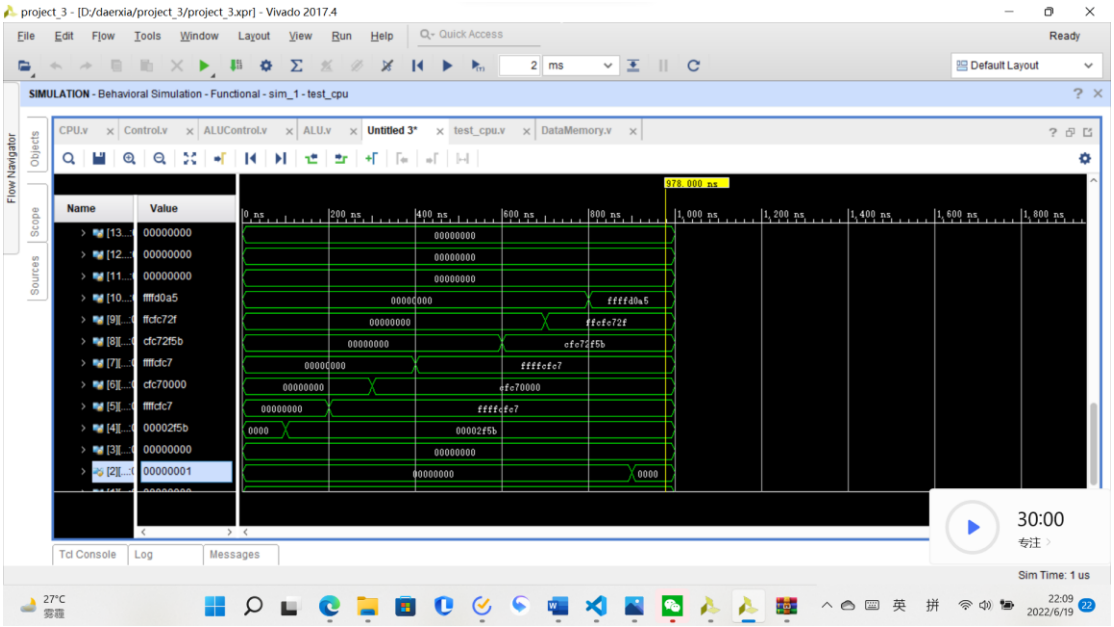
$v0:0

$v1:1

仿真：



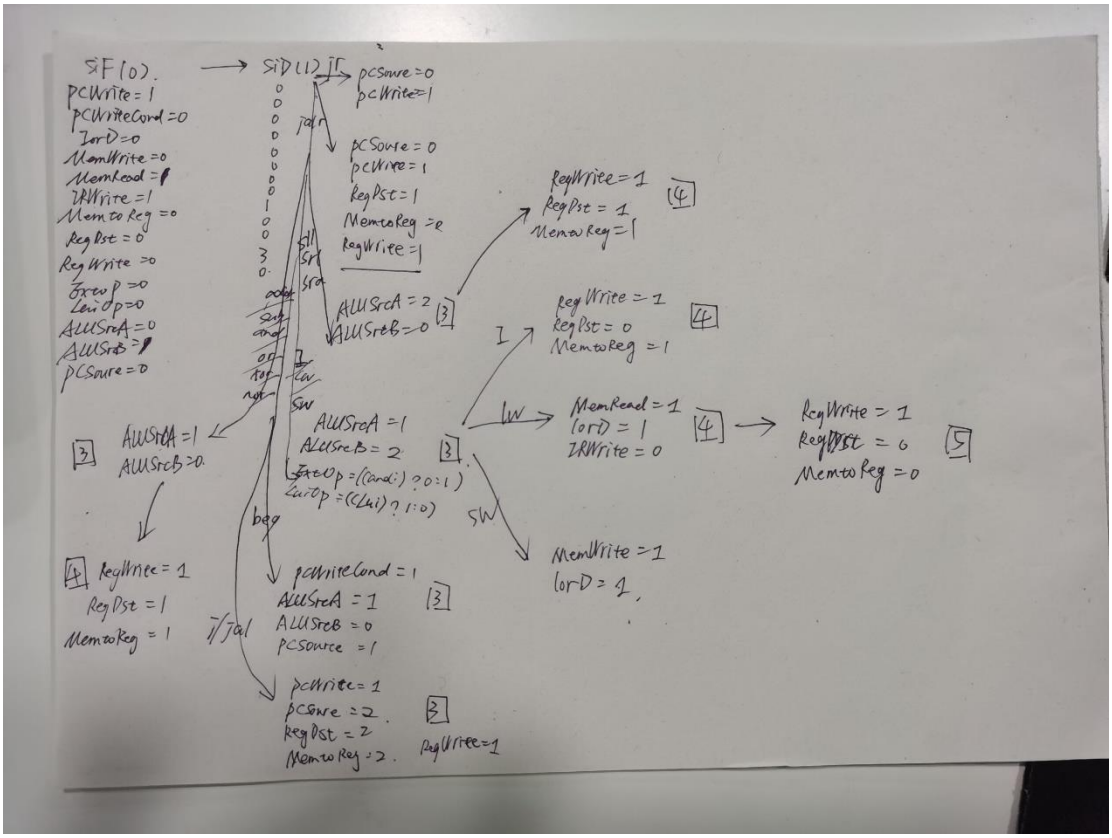
控制信号部分



寄存器结果

2.MIPS 多周期 CPU 的实现

(a) 多周期状态机控制器：



(b) 多周期 CPU 的 ALU 控制逻辑与功能实现

我们补全 controller 的代码利用 OpCode 和 Funct 获得 ALUOp

补全 ALUCtrl 通过 aluop 和 funct 获得 aluconf

在 alu 中，aluconf'控制着 alu 的功能

每个"状态"都会更新 alu 的数据

和单周期的差别不是太大

lui,jr,jalr 三种指令和单周期有差别

（c）

3.（a）

```
0       addi $a0, $zero, 5          //$a0=5
1       xor $v0, $zero, $zero       //$v0=0
2       jal sum                     //跳转到 sum 保存当前 pc
    Loop:
3       beq $zero, $zero, Loop      //死循环
    sum:
4       addi $sp, $sp, -8           //$sp=$sp-8
5       sw $ra, 4($sp)              //MEM[$sp+4]=$ra
6       sw $a0, 0($sp)              //MEM[$sp+0]=$a0
7       slti $t0, $a0, 1            //$a0=5>1, $t0=0;    $a0=0<1,$t0=1
8       beq $t0, $zero, L1          //$t0=0,跳到 L1
```

```
9       addi $sp, $sp, 8          //移栈
10      jr $ra                    // 返回
    L1:
11      add $v0, $a0, $v0         //$v0=$a0+$v0
12      addi $a0, $a0, -1         //$a0=$a0-1
13      jal sum                   //跳转到 sum 保存当前 pc
14      lw $a0, 0($sp)            // $a0= MEM[$sp]    从 1 到 N
15      lw $ra, 4($sp)            // $ra= MEM[$sp+4]
16      addi $sp, $sp, 8          //$sp=$sp+8
17      add $v0, $a0, $v0         //$v0=$a0+$v0
18      jr $ra                    // 返回
```

1.算从 1 到 N 的和算两遍
2.loop？死循环
3.当 a0 还大于 1 的时候，跳到 L1
4.计算 1~N 的和

机器码翻译：

```
0       addi $a0, $zero, 5        0x20040002
1       xor $v0, $zero, $zero     0x00001026
2       jal sum                   0x0c100004
    Loop:
3       beq $zero, $zero, Loop    0x1000ffff
    sum:
4       addi $sp, $sp, -8         0x23bdfff8
5       sw $ra, 4($sp)            0xafbf0004
6       sw $a0, 0($sp)            0xafa40000
7       slti $t0, $a0, 1          0x28880001
8       beq $t0, $zero, L1        0x11000002
9       addi $sp, $sp, 8          0x23bd0008
10      jr $ra                    0x03e00008
    L1:
11      add $v0, $a0, $v0         0x00821020
12      addi $a0, $a0, -1         0x2084ffff
13      jal sum                   0x0c100004
14      lw $a0, 0($sp)            0x8fa40000
15      lw $ra, 4($sp)            0x8fbf0004
16      addi $sp, $sp, 8          0x23bd0008
17      add $v0, $a0, $v0         0x00821020
18      jr $ra                    0x03e00008
```