计算机组成原理-P3 Logisim 单周期实验报告

一、模块规格

1. IFU

取指令模块，内部包含寄存器 PC 以及指令存储器 IM

端口定义表

信号名	方向	描述
clk	I	时钟信号
reset	I	重置信号
Jump	I	选择 PC 的值来源: PC[31:28] || JAddress[25:0] || 0^2
Branch	I	选择 PC 的值来源: PC + 4 + (branchOffset || 0^2)
branchOffset[15:0]	I	分支偏移量, b 系列指令
JAddress[25:0]	I	跳转目标地址，针对 J 指令
Instr[31:0]	O	当前 PC 所指指令
PC+4[31:0]	O	当前 PC+4 的值

功能定义表

序号	功能名称	功能描述
1	重置	reset 置 1 时，PC 寄存器复位
2	顺序执行	Jump == 0 且 Branch == 0 时，Instr 为 PC 所指处指令，PC <= PC + 4
3	分支（branch）	Branch == 1 时，Instr 为 PC 所指处指令，PC <= PC + 4 + branchOffset
4	跳转（J）	Jump == 1 时，Instr 为 PC 所指处指令，PC <= (PC + 4)[31:28] || JAddress || 0^2

2. GRF

端口定义表

信号名	方向	描述
clk	I	时钟信号
reset	I	重置信号
RegWEnable	I	写使能信号
RAddr1[4:0]	I	读功能所读寄存器地址 1
RAddr2[4:0]	I	读功能所读寄存器地址 2
RegWAddr[4:0]	I	写功能所写寄存器地址
RegWData[31:0]	I	写功能写入数据
RD1[31:0]	O	读寄存器数据 1
RD2[31:0]	O	读寄存器数据 2

功能定义表

序号	功能名称	描述
1	重置	reset == 1 时重置 GRF，所有寄存器归零
2	读取寄存器数据	clk 接时钟信号时，RD1/2 输出寄存器编号为 RAddr1/2 的寄存器的值
3	写入寄存器	WEnable == 1 时，将 WD 处数据在下一个时钟上升沿存入编号为 WAddr 的寄存器

3. ALU

端口定义表

信号名	方向	描述
Operand1[31:0]	I	操作数 1
Operand2[31:0]	I	操作数 2（具体由 ALUOp2Datapath 元件确定）
ALUOperation[2:0]	I	具体操作，0：加， 1：减， 2: 或， 其他待定
Result[31:0]	O	运算结果
isZero	O	结果是否为 0

功能定义表

序号	功能名称	描述
1	运算	根据 ALUOperator 指定的操作进行运算，结果输出至 Result。具体操作：0：加， 1：减， 3: 或， 其他待定
2	判断结果是否为 0	运算结果为 0 时，isZero 置 1

4. DM

端口定义表

信号名	方向	描述
clk	I	时钟信号
reset	I	重置数据，清零 RAM
Address[31:0]	I	操作地址
MemWData[31:0]	I	待写入数据
MemWEnable	I	是否写入数据
MemMode[1:0]	I	操作位宽，0 --- word, 1 --- half, 3 --- byte
MemRdata[31:0]	O	操作地址处的数据

功能定义表

序号	功能名称	描述
1	读取	以指定位宽读取指定位置数据
2	写入	以指定位宽在指定位置写入数据

注：对于不是以字为宽度的 IO 操作，输出采用符号延拓

5. **EXT**

端口定义表

信号名	方向	描述
EXTInput[15:0]	I	输入数据
EXTMode	I	操作模式： 0 --- 无符号延拓 32， 1 --- 有符号延拓 32
EXTOutput[31:0]	I	输出数据

功能定义表

序号	功能名称	描述
1	无符号延拓 32 位	EXTMode == 0 时，将 EXTInput 做 32 位无符号延拓
2	有符号延拓 32 位	EXTMode == 1 时，将 EXTInput 做 32 位有符号延拓

二、控制器设计

1. **概述**

该部分介绍 Controller 及各 MUX 的详细信息。Controller 由两部分连接成：确定指令的 InstructionDecode 及确定对应控制信号的 ControlSignalGenerator。

总体端口定义如下，具体信号功能请参见 ControlSignalGenerator 部分描述：

信号名	方向	描述
OpCode[5:0]	I	指令的 OpCode 部分
Funct[5:0]	I	指令的 Funct 部分
BranchEqual	O	相等时分支
Jump	O	跳转指示
RegWEnable	O	寄存器写使能信号
RegWAddrpath_sel[1:0]	O	寄存器写地址选择
RegWDatapath_sel[1:0]	O	寄存器写数据选择
ALUOperation[2:0]	O	ALU 操作信号
ALUOp2Datapath_sel[1:0]	O	ALU 操作数 2 选择信号
MemWEnable	O	内存写使能
MemMode[1:0]	O	操作位宽
EXTMode	O	Extender 模式
Link	O	将 PC + 4 存入$ra 寄存器
JumpReg	O	跳转到寄存器地址

2. **相关各 MUX 控制信号说明**

a) RegWAddrpath: 根据 RegWAddrpath_sel 的选择输出对应数据

端口定义表

信号名	方向	描述
RegWAddrpath_sel[1:0]	I	选择信号：0 --- rd， 1 --- rt，2 --- $ra，其他待定
RegWAddrpath_rdIn[4:0]	I	rd
RegWAddrpath_rtIn[4:0]	I	rt
RegWAddrpath_out[4:0]	O	输出

b) RegWDatapath：根据 RegWDatapath_sel 的选择输出对应数据

端口定义表

信号名	方向	描述
RegWDatapath_sel[1:0]	I	选择信号：0 --- alu 结果， 1 --- mem 数据，2 --- PC，其他待定
RegWDatapath_aluIn[31:0]	I	ALU 结果
RegWDatapath_memIn[31:0]	I	MEM 结果
RegWDatapath_out[31:0]	O	输出

c) ALUOp2Datapath: 根据 ALUOp2Datapath_sel 的选择输出对应数据

端口定义表

信号名	方向	描述
ALUOp2Datapath_sel[1:0]	I	选择信号，0 --- RegData2，1 --- EXTender，2 --- 移位后的 immediate(用于 lui)，3 --- 待定
ALUOp2Datapath_regIn[31:0]	I	reg 数据输入
ALUOp2Datapath_extIn[31:0]	I	ext 数据输入
ALUOp2Datapath_shiftIn[31:0]	I	移位后 immediate 输入
ALUOp2Datapath_out[31:0]	O	输出

3. **InstructionDecoder**

端口定义表

信号名	方向	描述
OpCode[5:0]	I	指令的 OpCode 部分
Funct[5:0]	I	指令的 Funct 部分
ID_addu	O	指令为 addu 时为 1
ID_subu	O	指令为 subu 时为 1
ID_ori	O	指令为 ori 时为 1
ID_lw	O	指令为 lw 时为 1
ID_sw	O	指令为 sw 时为 1
ID_beq	O	指令为 beq 时为 1
ID_lui	O	指令为 lui 时为 1

功能：译码 --- 当检测到某指令时输出其对应信号

OpCode	Funct	ID_addu	ID_subu	ID_ori	ID_lw	ID_sw	ID_beq	ID_lui
000000	100001	1	0	0	0	0	0	0
000000	100011	0	1	0	0	0	0	0
001101	xxxxxx	0	0	1	0	0	0	0
100011	xxxxxx	0	0	0	1	0	0	0
101011	xxxxxx	0	0	0	0	1	0	0
000100	xxxxxx	0	0	0	0	0	1	0
001111	xxxxxx	0	0	0	0	0	0	1

4. **ControlSignalGenerator** 端口定义表及信号介绍

信号名	方向	描述
ID_addu	I	指令为 addu
ID_subu	I	指令为 subu
ID_ori	I	指令为 ori
ID_lw	I	指令为 lw
ID_sw	I	指令为 sw
ID_beq	I	指令为 beq
ID_lui	I	指令为 lui
BranchEqual	O	为 1 代表 ALU 结果为 0 时分支
Jump	O	跳转指示，为 1 时进行跳转
RegWEnable	O	寄存器写使能信号
RegWAddrpath_sel[1:0]	O	寄存器写地址选择，见 RegWAddrpath 模块中同名输入信号
RegWDatapath_sel[1:0]	O	寄存器写数据选择，见 RegWDatapath 模块中同名输入信号
ALUOperation[2:0]	O	ALU 操作信号，见 ALU 模块
ALUOp2Datapath_sel[1:0]	O	ALU 操作数 2 选择信号，见 ALUOp2Datapath 模块
MemWEnable	O	内存写使能，为 1 时内存可写
MemMode[1:0]	O	操作位宽，见 DM 模块中同名输入信号
EXTMode	O	Extender 模式，0 --- 无符号延拓；1 --- 有符号延拓
Link	O	将 PC + 4 存入$ra 寄存器
JumpReg	O	跳转到寄存器地址

功能：根据输入指令信号输出对应控制信号

信号	addu	subu	ori	lw	sw	beq	lui
BranchEqual	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	0
RegWEnable	1	1	1	1	0	0	1
RegWAddrpath_sel[1:0]	0	0	1(b01)	1(b01)	xx	xx	1(b01)
RegWDatapath_sel[1:0]	0	0	0	1(b01)	xx	xx	0
ALUOperation[2:0]	0	1(b001)	2(b010)	0	0	1(b001)	2(b010)
ALUOp2Datapath_sel[1:0]	0	0	1(b01)	1(b01)	1(b01)	0	2(b10)
MemWEnable	0	0	0	0	1	0	0
MemMode[1:0]	xx	xx	xx	0	0	xx	xx
EXTMode	x	x	0	1	1	1	0
Link	0	0	0	0	0	0	0
JumpReg	0	0	0	0	0	0	0

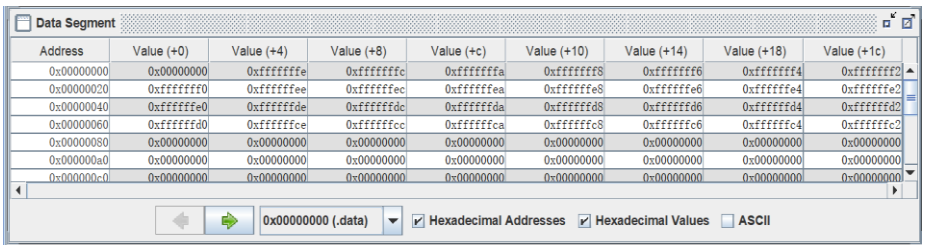
三、测试程序

以下为测试程序及期待结果，使用时机器码需去掉前两行

```
li $gp, 0
li $sp, 0
# Used for simulating i
n MARS
begin:
ori $18, $18, 3
ori $20, 0x3456
sw $20, 1($18)
ori $18, 4
lui $30, 0xaaaa
ori $30, 0xbbbb
lw $30, -3($18)
beq $20, $20, jumo
beq $zero, $zero, begin
jumo:
sw $zero, 4($zero)
start:
ori $3, 0x0120
lui $3, 0xf568
lui $4, 0x4127
addu $5, $4, $0
lui $7, 0
lui $8, 0
lui $9, 0
lui $10, 0
ori $8, 0x0004
ori $9, 0x0002
lui $23, 0
ori $23, 0x80
loop:
sw $10, ($7)
addu $7, $7, $8
subu $10, $10, $9
beq $7, $23, end_loop
beq $0, $0, loop
end_loop:
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0xf5680000
$a0	4	0x41270000
$a1	5	0x41270000
$a2	6	0x00000000
$a3	7	0x00000080
$t0	8	0x00000004
$t1	9	0x00000002
$t2	10	0xffffffc0
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000007
$s3	19	0x00000000
$s4	20	0x00003456
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000080
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00000000
$sp	29	0x00000000
$fp	30	0x00003456
$ra	31	0x00000000
pc		0x00003074
hi		0x00000000
lo		0x00000000



四、思考题

1. L0.T2.1 **若 PC（程序计数器）位数为 30 位，试分析其与 32 位 PC 的优劣。**

   优势：因为指令是在字边缘上对其的，所以 30 位 PC 足以寻址所有的指令，节省了两位寄存器的空间

   劣势：外围电路更加复杂，地址需要在输入 PC 寄存器前移位，PC 寄存器输出的地址需要经过拓展才能输入 RAM。且非标准的 30 位寄存器可能会使成本增高

2. L0.T2.2 **现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。**

   DM 和 GRF 部分合理，DM 需要大容量、可修改的存储器，GRF 需要速度快的存储器，容量需求有限。IM 部分应该也使用 RAM，因为程序运行过程中可能需要动态加载代码/修改代码区域的功能。动态链接库技术以及操作系统的管理都需要这样的功能。

3. L0.T3.1 **结合上文给出的样例真值表，给出 RegDst， ALUSrc， MemtoReg，RegWrite，nPC_Sel，ExtOp 与 op 和 func 有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）**

See MIPS Green Sheet							
	func	10 0000	10 0010	n/a			
	op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100
		add	sub	ori	lw	sw	beq
Control Signals	RegDst	1	1	0	0	X	X
	ALUSrc	0	0	1	1	1	0
	MemtoReg	0	0	0	1	X	X
	RegWrite	1	1	1	1	0	0
	MemWrite	0	0	0	0	1	0
	nPC_sel	0	0	0	0	0	1
	ExtOp	X	X	0	1	1	X
	ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract
		All Supported Instructions					

   约定o_5~o_0代表 Opcode 第 5~0 位，f_5~f_0代表 Funct 第 5 位~第 0 位

   RegDst=~o5·~o4·~o3·~o2·~o1·f5·~f4·~f3·~f2·~f0

   ALUSrc=(~o5·~o4·o3·o2·~o1·o0) + (o5·~o4·~o3·~o2·o1·o0) + (o5·~o4·o3·~o2·o1·o0)

MemtoReg=o5·~o4·~o3·~o2·o1·o0

RegWrite=(~o5·~o4·~o3·~o2·~o1·f5·~f4·~f3·~f2·~f0) + (~o5·~o4·o3·o2·~o1·o0) + (o5·~o4·~o3·~o2·o1·o0)

nPC_Sel=~o5·~o4·~o3·o2·~o1·~o0

ExtOp=o5·~o4·~o2·o1·o0

4. L0.T3.2 **充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式，请给出化简后的形式。**

RegDst = ~o0

ALUSrc = o0

MemtoReg = o5

RegWrite = (~o3·~o2) + (o3·o2)

MemWrite = o5·o3

nPC_sel = ~o3·o2

ExtOp = o5

5. L0.T3.3 **事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。**

nop 指令在与逻辑（InstructionDecoder）中不会被解析成已知信号，则代表各指令的信号输出均为 0，在或逻辑中，各控制信号均为 0，CPU 只会将 PC 加 4，不会产生其他可观测影响。

6. L0.T4.1 **前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM 片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。**

假设 MARS 内存配置为从 0 开始，compact（数据从 0x3000 开始）。DM 中在输入 RAM 的地址信号前添加一个 MUX，MUX 的两个输入分别为输入的的 Address 和 Address – 0x00003000，MUX 选择信号设为将地址的高 20 位与 0x00003 比较的结果，相等则选择 Address – 0x00003000，否则选择 Address。

7. L0.T4.2 **除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)"了解相关内容后，简要阐述相比与测试，形式验证的优劣。**

所谓形式验证，是指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的功能。形式验证方法分为等价性验证、模型检验和定理证明等。

相比测试，形式验证可以对指定描述的所有可能的情况进行验证，涵盖了所有输入场景，还可以检测边角案例异常，且无需开发测试数据，对大型电路速度一般比仿真快。但是形式验证也有其劣势，形式验证中的等价性验证、模型检验均需要有一个功能完全正常的电路或工作模型，而定理证明需要人工干预及操作人员的专业知识，且在失败的情况下无法给出错误样例。此外，形式验证无法执行任何延时和功耗检查。