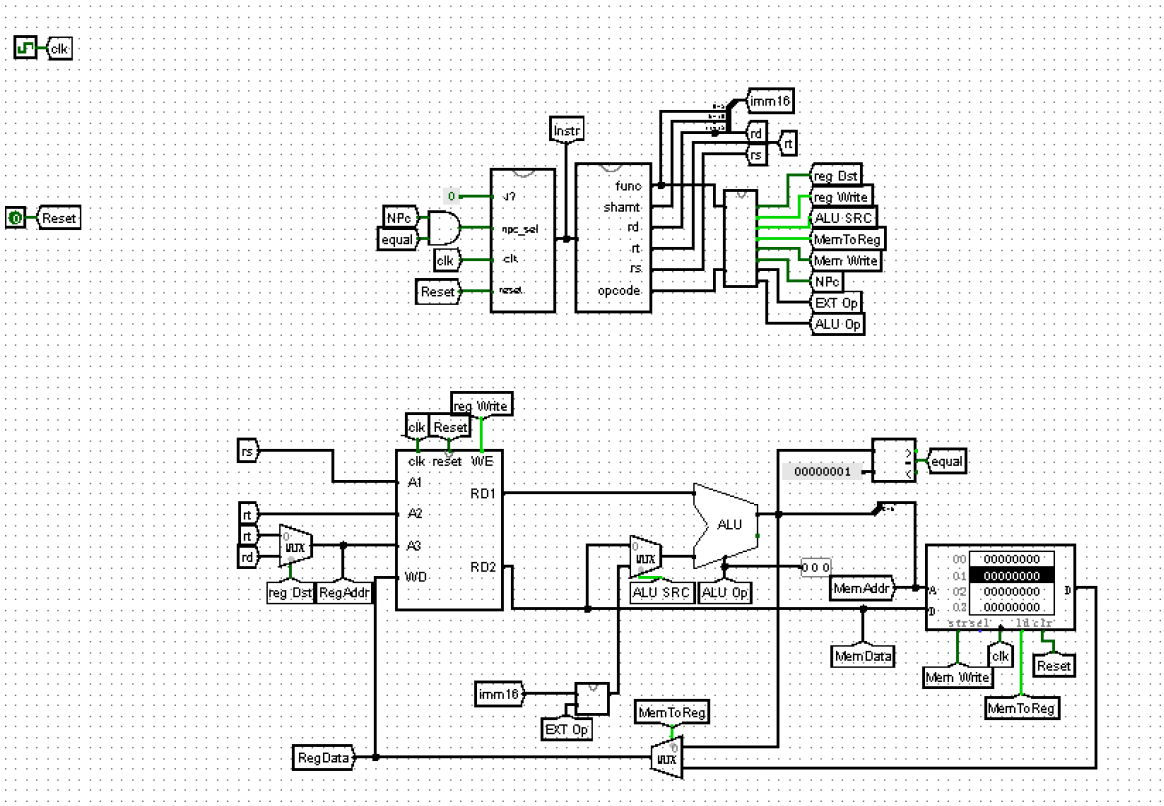
Logisim 单周期 CPU 实验报告

一、 设计说明

本处理器支持 MIPS 指令集中的 addu, subu, ori, lw, sw, beq, lui, nop，j 指令，其中 addu 和 subu 不支持进位功能。

二、Logisim 单周期 CPU 顶层设计



三、模块规格

1. IFU 模块

（1） 基本描述：

内部包括 PC 和 IM 级相关逻辑；PC 以 32 位的寄存器实现，设置复位信号，IM 中存有 32 位的指令，PC 中是 IM 中指令的地址，选择 2-6 位作为 IM 中的选择信号；

（2） 模块定义：

信号名	方向	描述
J	I	J 指令跳转标志
NPc_sel	I	Beq 指令跳转标志
Reset	I	复位信号
Clk	I	时钟信号
PC	O	当前地址
OPcode	O	从 IM 选择的代码

2. ALU 模块

（1）基本描述：

算术逻辑运算单元 ALU 用 32 加法器加上逻辑门实现了加、减、与以及判断是否相等的逻辑算术运算功能，有两个 32 位操作数，一个 32 位输出，一个进/退位输出（暂时未使用）和一个 3 位的控制 ALUop 和 equal 单位。

（2）模块定义：

信号名	方向	描述
SrcA	I	A 操作数 32 位
SrcB	I	B 操作数 32 位
ALUop	I	ALU 运算控制代码 001 ： A + B 001 ： A - B 010 ： A or B 100 ： A == B ？ 1 : 0
Equal	O	相等标志符
Result	O	运算结果 32 位
Mark	O	进/退位标志（未使用）

3. DM 模块

（1） 基本描述：

数据存储器从ＡＬＵ中计算出结果得出存储的地址，将 Memdata 的数据写入ＤＭ中，并把数据写出，有时钟和 MemWrite、MemToReg 和 clk、reset 作为控制信号，用 32 位的 RAM 实现，将ＤＭ做成利用地址将选定寄存器实现数据存储。

（2） 模块定义：

信号名	方向	描述
A	I	指令存储器中寄存器的地址
Memdata	I	写入ＤＭ中的３２位数据
MemToReg	I	控制内存是否有输出
Clk	I	时钟信号
Reset	I	清零信号
MemWrite	I	写使能信号
D	O	从 DM 读出的数据

4. GPR 模块

（1）基本描述：

寄存器堆 GPR 有 32 个存储 32 位数据的寄存器,从 IFU 中获得的 32 位指令，通过拆分 opcode 获得寄存器地址，在 GPR 中选择相应的寄存器，从 RD1 和 RD2 中读出操作数，或从 WD 向 A3 寄存器存值。

（2）模块定义：

信号名	方向	描述
A3	I	写入数据时候写入的地址
WD	I	写入 Wreg 指向的地址的寄存器中写入 32 位数据
Clk	I	时钟信号
WE	I	写使能信号
A1	I	存储在 RF 中的第一个数据的地址
A2	I	存储在 RF 中的第二个数据的地址
Reset	I	清零信号
RD1	O	从 RF 中读出的第一个数据
RD2	O	从 RF 中读出的第二个数据

5. Control 模块

（1）基本描述

将 32 位指令按规定格式转化成各个单元的控制信号，达到利用指令编码控制硬件和执行程序的目的。

（2）模块定义

信号名	方向	描述
func	I	特定操作说明位
op	I	Opcode 操作码
RegDst	O	寄存器选择信号 0: rt 1: rd
ALUSrc	O	控制 ALU 操作 0 为 R 类型运算　从 RD2 中取出数据运算 1 为 Lw/sw 运算　从 32Imm 选择数据
MemtoReg	O	控制数据从 ALU 读出还是 DM 中读出 0 选择 ALUresult　R 类型 1 选择 ReadData　支持 lw
RegWrite	O	寄存器堆的写使能信号
MemWrite	O	DM 的写使能信号 1 向 DM 写入数据
Npc_sel	O	Beq 分支指令的控制端
ALUctr<2:0>	O	ALU 的三位控制信号，控制 ALU 的操作 000 +　001 -　010 or　110 ==
J	O	j 跳转标志

四、控制器设计

控制器真值表：

func	000000	000000	001101	100011	101011	000100	001111	000010
opcode	100001	100011						
op	addu	subu	ori	lw	sw	beq	lui	J
nPC_Sel	0	0	0	0	0	1	0	X
RegDst(写寄存器目标)1-rd	1	1	0	0	x	x	0	X
RegWrite（写寄存器使能端）	1	1	1	1	x	x	1	X
MemWrite（数据写入指定存储器单元）	0	0	0	0	1	0	0	0
MemtoReg（写入寄存器的数据来源）	0	0	0	1	x	x	x	X
ALUSrc（第二个 alu 操作数来源）	0	0	1	1	1	0	1	X
Jump	0	0	0	0	0	0	0	1
ALUOp[2:0]（控制 ALU 运算）	000	001	010	000	000	011	000	XXX
ExtOp[2:0] （扩展方式）	x	x	000	001	001	x	010	XXX

五、测试程序

```
.data
array:      .space 20
.text
addu $t1, $0, 4
addu $t2, $0, 4
sw $t1, array($t2)
beq $t1, $t2, Loop_end
addu $t1, $t1, 1
sw $t0, array($t2)
Loop_end:
lw $t0, array($t2)
```

addu 加立即数会被 mars 编译为 lui、ori 和 addu 三个指令，可以同时测试这三条指令的正确性。使用 sw 将 array 的第 2 项赋值为 4，如果 beq 生效，则会将$t0 寄存器赋值为 4，如果无效则为 0（$t0 为 0）.

六、问答题

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

答：

Pc 寄存器减少后两位等效于 pc 除 4，即在后面只需要执行 pc+1 运算即可。并且所能存储的地址与按字节存储的 32 位 pc 相同均为 2^30 条。

但是，如果遇到 jal，需要向外输出完整的 pc，30 位寄存器需要末尾补 00。

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。

答：

合理。IM 中的指令集只需要读取而不需要输入，并且不随 reset 清零，又因为 ifu 中只需要读取指令所以使用 rom 合适。DM 中的储存器需

要可以完成读和取 2 个功能，所以选用 ram。Grf 中若使用 ram 则无法处理在一个周期内多个地址的任务，但是若使用 grf 则可以。

3.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。）

答：

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					

Func5 个端口分别为： f5,f4,f3,f2,f1

Op5 个端口分别为： o5,o4,o3,o2,o1

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.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。

答：

name	add	sub	ori	lw	sw	beq
Op5	0	0	0	1	1	0
Op4	0	0	0	0	0	0
Op3	0	0	1	0	1	0
Op2	0	0	1	0	0	1
Op1	0	0	0	1	1	0
Op0	0	0	1	1	1	0
f1	0	1	x	x	x	x
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	0	0	1	0	0	0
ALUOp[2:0]	000	010	111	000	000	010

RegDst = !o5&&!o4&&!o3&&!o2&&!o1

ALUSrc = o0

MemtoReg=o5&&&o1&&o0

RegWrite=(!o5&&!o4&&!o3&&!o2&&!o1)||(!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

5.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。

答：

nop 是 0X00000000,只是做了 PC=PC+4,对逻辑电路电路中的元件进行任何操作。

6.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。

答：

假设 DM 有 256MB 容量,并且映射在 0x3000_0000-0x3FFF_FFFF 区间,那么只需要把高 4 位地址与 0x3 进行比较,结果就是 DM 的片选信号。之前的 DM 存满后就从 0x3000_0000~0x3FFF_FFFF 存储数据。（不太会）

由于 5 位地址，所以这次的 DM 最多存到 0x0000_007f,所以不需要片选信号。

7.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。

答：

形式验证时要确定电路在哪一级电路上的测试是正确的，使用模型检验的方法看两个电路在描述上是否一致。

1．组合逻辑电路的逻辑验证

对组合逻辑来说，不存在状态寄存器，其输出值 Z[t]不依赖于前面的输入值 X[t-i](1≤i≤t)。这时只要对每个输入向量证明其输出向量相同。在组合逻辑验证领域有以下两类方法。

(1)转换为单一抽象模型比较。通过对单一表示的结构进行比较，得出其功能等价的结论。在最坏的情况下，布尔函数为正，表示随输入个数指数增加，其过大的内存需求限制了一般布尔函数的验证能力。

(2)利用测试输入向量进行验证。探寻使两个电路具有不同输出的输入测试向量，若不存在这样的测试向量，则电路在功能上等价。在最坏情况下，这种方法需要穷举所有可能的输入测试矢量，运行时间又成为

一个主要问题。

2．时序逻辑电路的验证

对一个时序电路而言，可以把它看成一个有限状态机(FSM，finite-state machine)。电路功能的等价可以用有限状态机的等价来判断。假定有两个状态机 A 和 B，要对它们进行比较。直观的说，当 A 和 B 有相同的接口，而且从相同的初始状态出发，两者对有效输入值序列产生相同的输出值序列，则可以说 A 和 B 等价。

形式验证的优点如下：

(1)形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了 100%。

(2)形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试激励。

(3)形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

形式验证主要验证数字 IC 设计流程中的各个阶段的代码功能是否一致，包括综合前 RTL 代码和综合后网表的验证，因为如今 IC 设计的规模越来越大，如果对门级网表进行动态仿真，会花费较长的时间，而形式验证只用几个小时即可完成一个大型的验证。另外，因为版图后做了时钟树综合，时钟树的插入意味着进入布图工具的原来的网表已经被修改了，所以有必要验证与原来的网表是逻辑等价的。