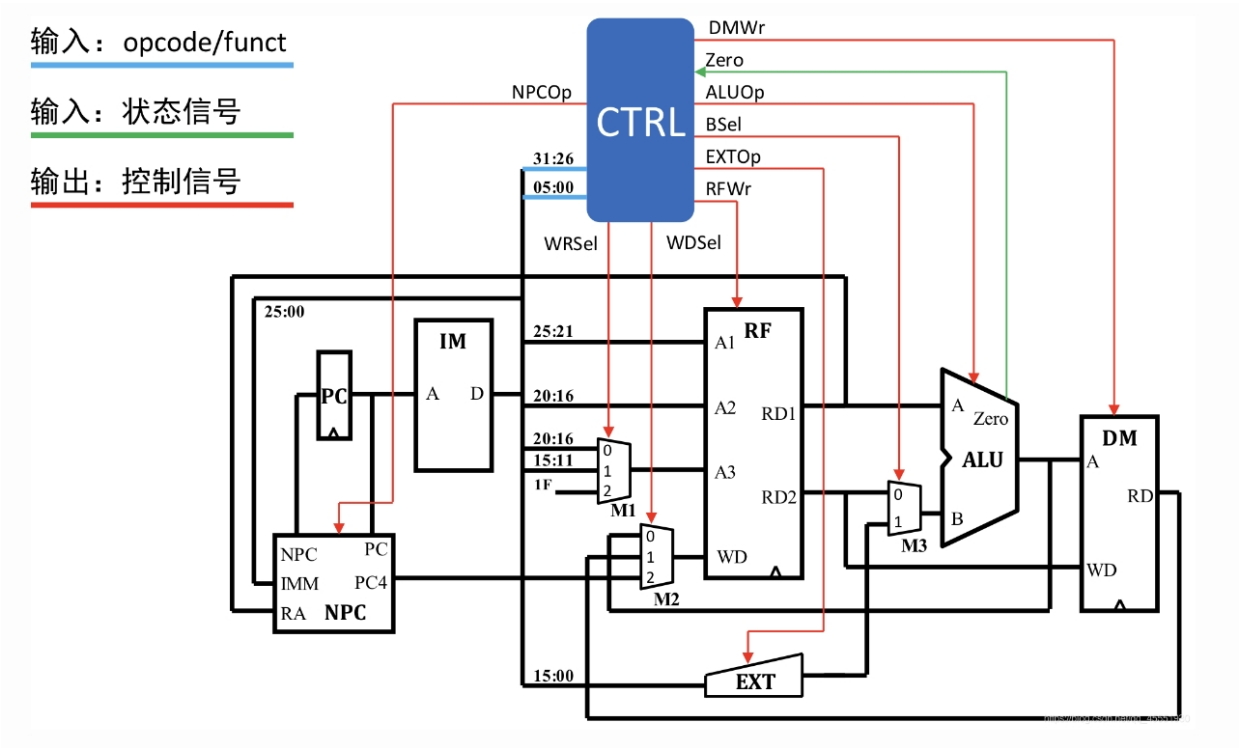
Logisim单周期CPU设计文档

注：表格中空着的两行均是为以后的扩展而留出的

设计要求

- 设计一个32位单周期cpu
- cpu支持`add, sub, ori, lw, sw, beq, lui, nop`操作
- `add, sub`按无符号加减法处理（不考虑溢出）

顶层设计（参考课件）



模块定义

1.IFU（取指令单元）

利用fsm的知识，将其分为三个模块NPC、PC、IM分别对应状态转移，状态储存，和状态输出

NPC端口说明：

序号	信号名	方向	描述
1	PC[31:0]	I	当前指令地址
2	NPCop[2:0]	I	输入控制信号
3	Ra[31:0]	I	输入$ra保存的地址
4	Zero	I	判断`$rs`和`$rt`是否相等，相等为1，否则为0

序号	信号名	方向	描述
5	NPC[31:0]	O	输出下条指令的地址
6	Imm[25:0]	I	包括beq的16位指令的扩展以及j和jr指令的26位立即数
7	PC+4[31:0]	o	输出pc+4的值

控制信号	功能
3’b000	输出PC+4
3’b001	执行beq的跳转
3’b010	执行j/jal的指令
3’b011	执行jr的指令

PC：用寄存器即可

IM端口说明：

序号	信号名	方向	描述
1	NPC[31:0]	I	需要取出的指令的地址
2	Instr[31:0]	O	需要执行的指令

2.Controller（控制器）

参考教程，我将控制器分为两个模块，分别为AND和OR，AND用于判断指令类型，OR用于执行具体的指令

序号	信号名	方向	描述
1	opcode[5:0]	I	输入opcode
2	funct[5:0]	I	输入funct
3	ALUop[3:0]	O	ALU控制信号
4	ALUsel	O	选择立即数/寄存器值进行运算

序号	信号名	方向	描述
5	DMwr	O	DM使能信号
6	NPCop[2:0]	O	NPC控制信号
7	EXTop	O	EXT控制信号
8	WE	O	GRF使能信号
9	SelWr[1:0]	O	待写入寄存器地址选择
10	SelWd[1:0]	O	写入寄存器数据选择
11	DMop[1:0]	O	DM控制信号

1.SelWr功能说明

控制信号值	功能
2’b00	选择待写入寄存器地址为15-11
2’b01	选择待写入寄存器地址为20-16
2’b10	选择待写入寄存器为 `$31`（`$ra`）

2.SelWd功能说明

控制信号值	功能
2’b00	选择写入寄存器的数据来自DM
2’b01	选择写入寄存器的数据来自ALU的计算结果
2’b10	选择写入寄存器的数据来自PC+4

3.ALUsel功能说明

控制信号值	功能
1‘b0	选择寄存器的值进行运算
1’b1	选择立即数进行运算

控制信号值	功能

3.GRF

利用P0的课下即可

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号，将 32 个寄存器中的值全部清零 1：复位 0：无效
WE	I	写使能信号 1：可向 GRF 中写入数据 0：不能向 GRF 中写入数据
A1	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD1
A2	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
A3	I	5 位地址输入信号，指定 32 个寄存器中的一个作为写入的目标寄存器
WD	I	32 位数据输入信号
RD1	O	输出 A1 指定的寄存器中的 32 位数据
RD2	O	输出 A2 指定的寄存器中的 32 位数据

模块功能定义如下：

序号	功能名称	描述
1	复位	reset信号有效时，所有寄存器存储的数值清零，其行为与logisim自带部件register的reset接口完全相同
2	读数据	读出 A1,A2 地址对应寄存器中所存储的数据到 RD1,RD2
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中。

4.DM（数据存储器）

序号	信号名	方向	描述
1	addr[31:0]	I	待操作的内存地址
2	WD[31:0]	I	待输入的值
3	DMwr	I	使能信号
4	clk	I	时钟信号
5	reset	I	复位信号
6	RD[31:0]	O	memory[addr]的值
7	DMop[1:0]	I	控制信号

控制信号值	功能
2’b00	sw和lw
2’b01	sh和lh
2’b10	sb和lb

5.ALU

序号	信号名	方向	描述
1	A[31:0]	I	运算数A
2	B[31:0]	I	运算数B
3	C[31:0]	O	运算结果C
4	ALUop[3:0]	I	控制信号
5	Zero	O	判断A==B?
6	LessZero	O	判断A-B<0?

控制信号：

控制信号值	功能
4’b0000	加法运算
4’b0001	减法运算
4’b0010	or运算
4’b0011	lui运算
4’b0100	and运算

注：and运算为自行扩展

6.EXT

将16位立即数扩展成32位

序号	信号名	方向	描述
1	Imm[15:0]	I	待扩展的16位数
2	extOut	O	扩展后的32位数
3	EXTop	I	控制信号

控制信号值	操作
1’b0	进行零扩展
1’b1	进行符号扩展

7.万能分线器（Splitter）

序号	信号名	方向	描述
1	Code[31:0]	I	输入指令
2	opcode[5:0]	O	

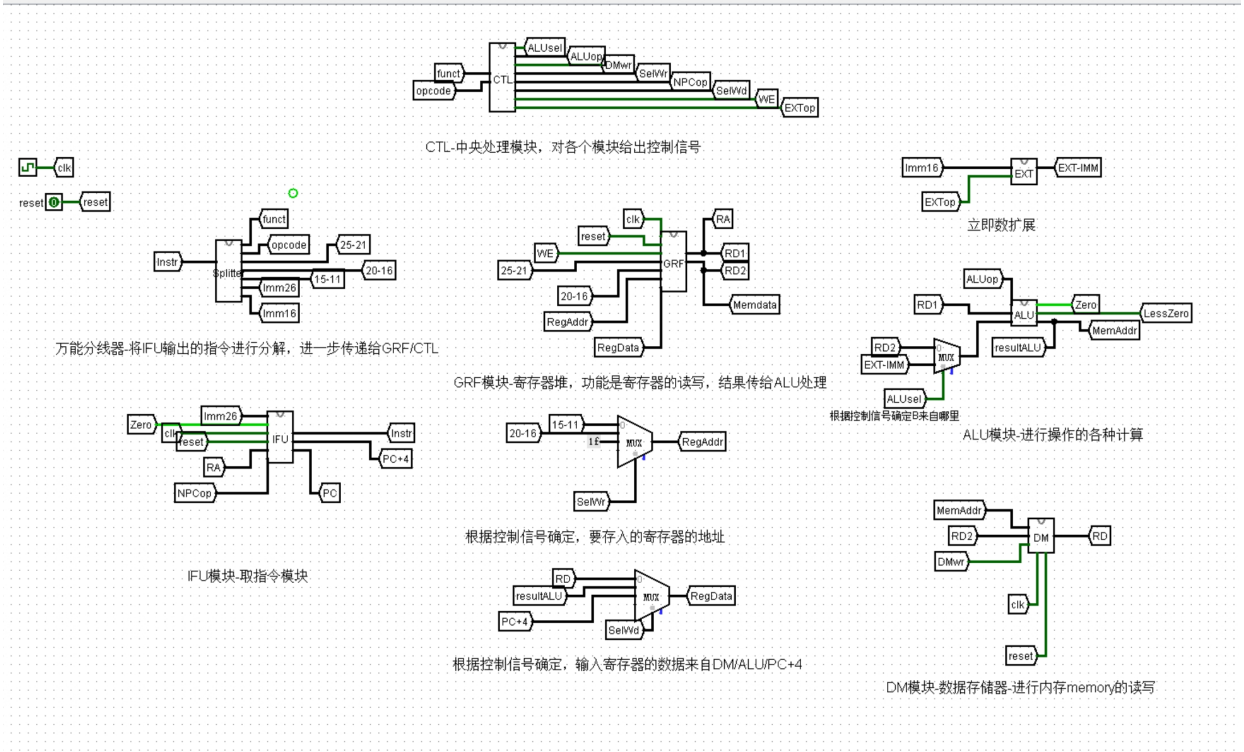
序号	信号名	方向	描述
3	funct[5:0]	O	
4	25-21	O	25-21位
5	20-16	O	20-16位
6	15-11	O	15-11位
7	Imm16[15:0]	O	16位立即数
8	Imm26[25:0]	O	26位立即数

8.数据通路分析

指令	ALUsel	ALUop	DMwr	SelWr	SelWd	NPCop	EXTop	WE
add	0	0000	0	00	01	000	X	1
sub	0	0001	0	00	01	000	X	1
ori	1	0010	0	01	01	000	0	1
lw	1	0000	0	01	00	000	1	1
sw	1	0000	1	01	00	000	1	0
beq	0	0000?	0	X	X	001	X	0
lui	1	0011	0	01	01	000	X	1
j	X	X	0	X	X	010	X	0
jal	X	X	0	10	10	010	X	1
jr	X	X	0	X	X	011	X	0
lh	1	0000	0	01	00	000	1	1

注：j/jal/jr是我自行扩展的，不属于题目要求的范围内

整理风格，模块化



测试

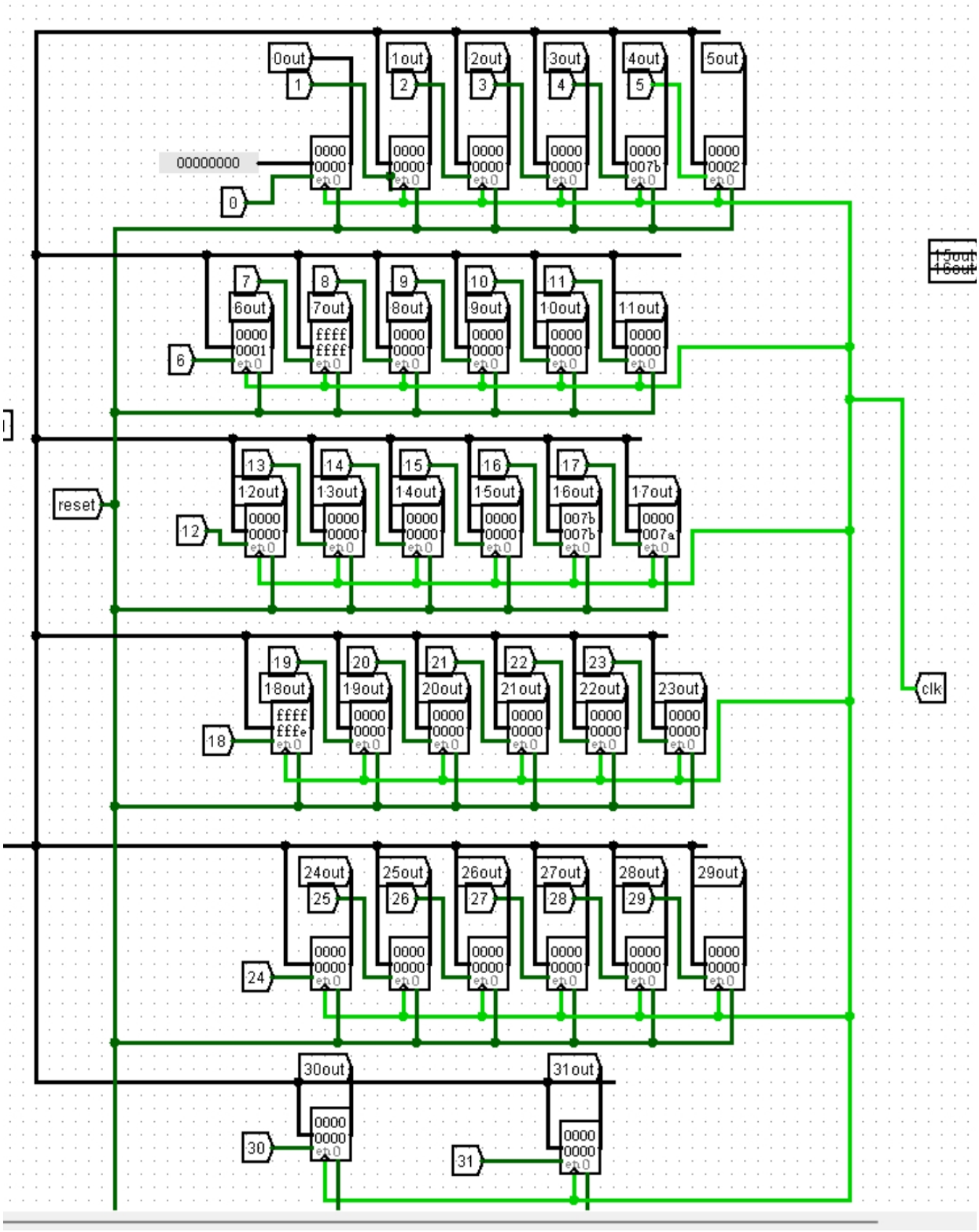
测试代码：

```
ori $a0, $0, 123
ori $a1, $a0, 456
lui $a2, 123             # 符号位为 0
lui $a3, 0xffff          # 符号位为 1
ori $a3, $a3, 0xffff     # $a3 = -1
add $s0, $a0, $a2      # 正正
add $s1, $a0, $a3      # 正负
add $s2, $a3, $a3      # 负负
ori $t0, $0, 0x0000
sw $a0, 0($t0)
sw $a1, 4($t0)
sw $a2, 8($t0)
sw $a3, 12($t0)
sw $s0, 16($t0)
sw $s1, 20($t0)
sw $s2, 24($t0)
lw $a0, 0($t0)
lw $a1, 12($t0)
sw $a0, 28($t0)
sw $a1, 32($t0)
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
beq $a0, $a1, loop1     # 不相等
beq $a0, $a2, loop2     # 相等
loop1:sw $a0, 36($t0)
loop2:sw $a1, 40($t0)
```

导出16进制文件：

```
v2.0 raw
3404007b
348501c8
3c06007b
3c07ffff
34e7ffff
00868020
00878820
00e79020
34080000
ad040000
ad050004
ad060008
ad07000c
ad100010
ad110014
ad120018
8d040000
8d05000c
ad04001c
ad050020
34040001
34050002
34060001
10850001
10860001
ad040024
ad050028
```

该cpu运行结果：




```
v2.0 raw
7b 1fb 7b0000 ffffffff 7b007b 7a fffffffe 7b
ffffffff 0 2
```

mars中的结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000001
$a1	5	0x00000002
$a2	6	0x00000001
$a3	7	0xffffffff
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x007b007b
$s1	17	0x0000007a
$s2	18	0xfffffffe
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000306c
hi		0x00000000
lo		0x00000000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x0000007b	0x000001fb	0x007b0000	0xffffffff	0x007b007b	0x0000007a	0xfffffffe	0x0000007b
0x00000020	0xffffffff	0x00000000	0x00000002	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000001a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

通过对比发现完全一致，通过该样例测试

思考题

1. **上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能**

`IFU`、`EXT`、`ALU`、`Controller`充当了FSM中的状态转移功能，`GRF`、`DM`发挥了状态存储功能。

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

合理。首先让我们明确各个模块需要完成的功能。`IM`模块只需要被读取，`DM`既需要读取也需要写入，`GRF`需要读写，另外需要指出的是其与`ALU`相连，对其读写速度要求较高。明确功能后我们来看看各个元件的特点。`ROM`只能读取，符合`IM`的功能所需；`RAM`可以读写，并且**Separate load and store ports**的性质可以较好地满足读写的需求，当然我们也可以用同样具有读写功能的`Register`完成，但是这样就需要大量的`Register`，造成一定程度的资源浪费；`Register`可以进行高速读写，满足`GRF`的功能所需。综上所述，这样的使用方法是合理的。

3. **在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路**

是的。我额外设计了`Splitter`模块，我称之为万能分线器模块，将指令分解成我所需的各个part，例如`funct`、`opcode`、`25-21`等，模块内利用简单的分线器即可实现对应功能。

4. **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？**

因为nop指令实际上执行的是`sll $0, $0, 0`，机器码为`0x0000_0000`，事实上执行的就是`pc<=pc+4`的指令，由于我们用或门实现信号控制，加或不加没有差别。

5. **阅读 Pre 的 “MIPS 指令集及汇编语言” 一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处**

- 首先，这里的测试样例对`ori`指令进行了测试，关注到了与`$0`进行或运算，但是还应该考虑`$0`作为目标寄存器的情况；同时也可以考虑16 位无符号数边界附近的数，例如65533,65534,6553565533,65534,65535。
- add指令的测试充分考虑到了 正正、正负、负负的三种情况。
- sw指令的测试中，考虑到了`$base`寄存器中的值为正数、为负数，还应该考虑寄存器中的值为零的情况；`offset`方面考虑到了正数和零，还应该考虑到为负数的情况。
- `lw`指令的测试，可注意测试目标寄存器是`$0`的情况。
- `beq`的测试，可利用内存中的结果进行检查。
- 指令的覆盖方面，有部分指令没得到测试，例如`sub`指令

总之，题目所指测试样例有着较好的测试强度，但再指令覆盖率和各个指令的分支覆盖率方面仍有改进的空间。