单周期 CPU

一、 设计目的与说明

使用 logisim 搭建一个 32 位 8 指令的单周期 CPU

处理器应支持的指令集为：{addu,subu,ori,lw,sw,beq,lui,nop}；

Nop 的机器码为 0x00000000，是空指令，不进行有效行为。

Addu,subu 不支持溢出；

采用模块化和层次设计，顶层有效的驱动信号要求有且仅仅包括 reset；

Clk 是内置时钟。

二、 单周期数据通路设计

指令	Adder		PC	IM.A	GRF				ALU		DM		EXT	Nadd	
	A	B			RA1	RA2	WA	WD	A	B	A	WD		A	B
R 型指令	PC	4	Adder	PC	Rs	Rt	Rd	ALU	RF.RD1	RF.RD2					
ORI	PC	4	Adder	PC	Rs		Rt	ALU	RF.RD1	EXT			IMM16		
LUI	PC	4	Adder	PC	$0		Rs	ALU	0	EXT			IMM16		
LW	PC	4	Adder	PC	Rs		Rt	DM.RD	RF.RD1	EXT	ALU		IMM16		
SW	PC	4	Adder	PC	Rs	Rt			RF.RD1	EXT	ALU	RF.RD2	IMM16		
BEQ	PC	4	Adder|Nadd	PC	Rs	Rt			RF.RD1	RF.RD2			IMM16	Adder	Shift

Rs=Instr(25:21)

Rt=Instr(20:16)

Rd=Instr(15:11)

Imm16=Instr(15:0)

“Adder|Nadd”表示需要一个MUX,ALU的判零输出端Zero可直接作为该MUX的选择控制。

三、 模块规格

1、IFU（取指令单元）

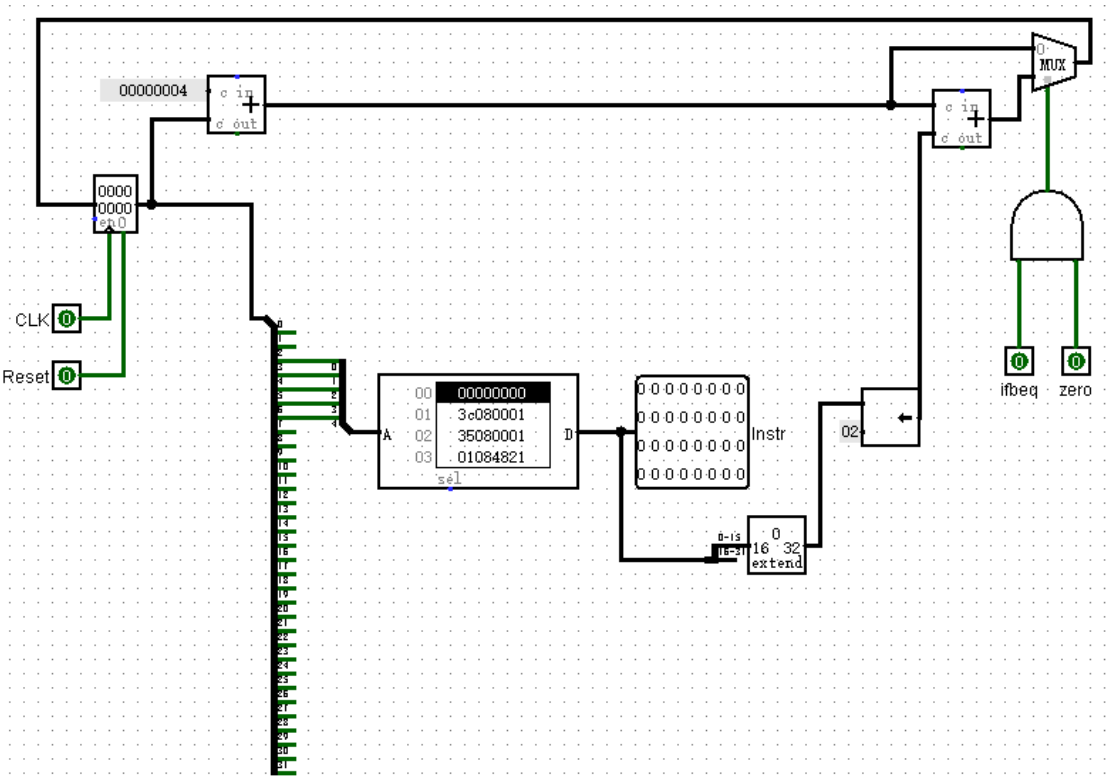
包含PC（程序计数器）,IM（指令存储器）及相关逻辑。

PC用寄存器实现，应具有复位功能。

起始地址：0x00000000.

IM用ROM实现，容量为32bit * 32。

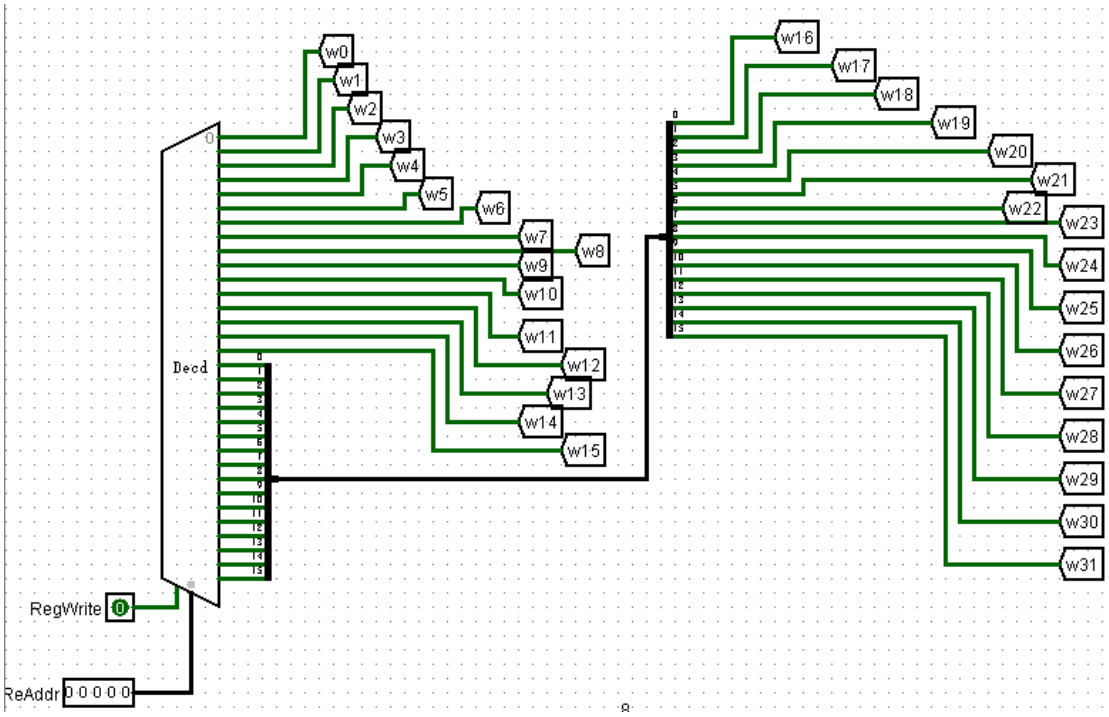
因IM实际地址宽度仅为5位，故需使用恰当的方法将PC中储存的地址同IM联系起来。



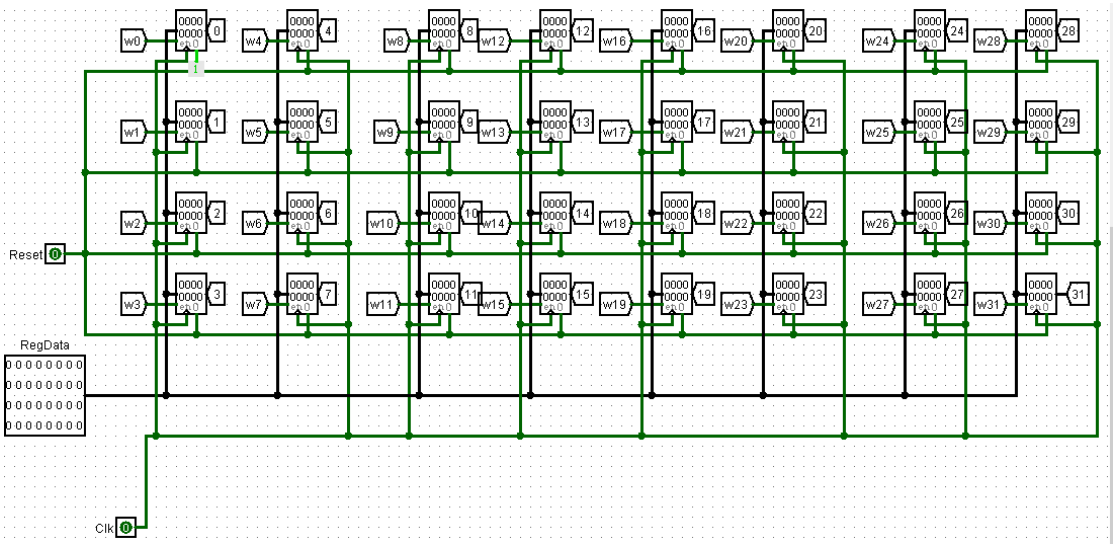
2、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）

用具有写使能的寄存器实现，寄存器总数为32个

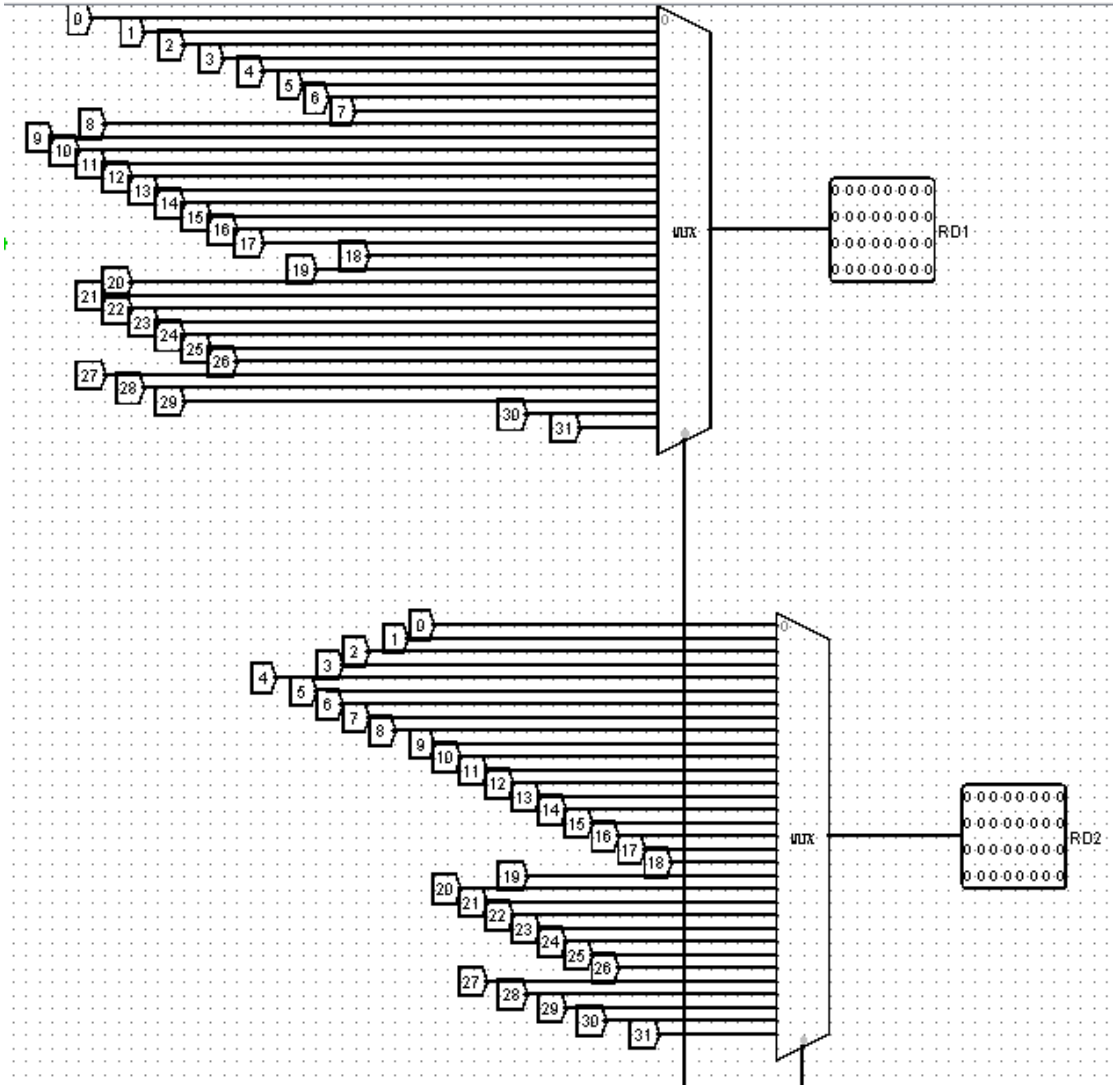
0号寄存器的值保持为0



说明：用译码器来控制寄存器的使用



说明：寄存器输入由时钟信号，译码器控制，输出由多路选择器控制。



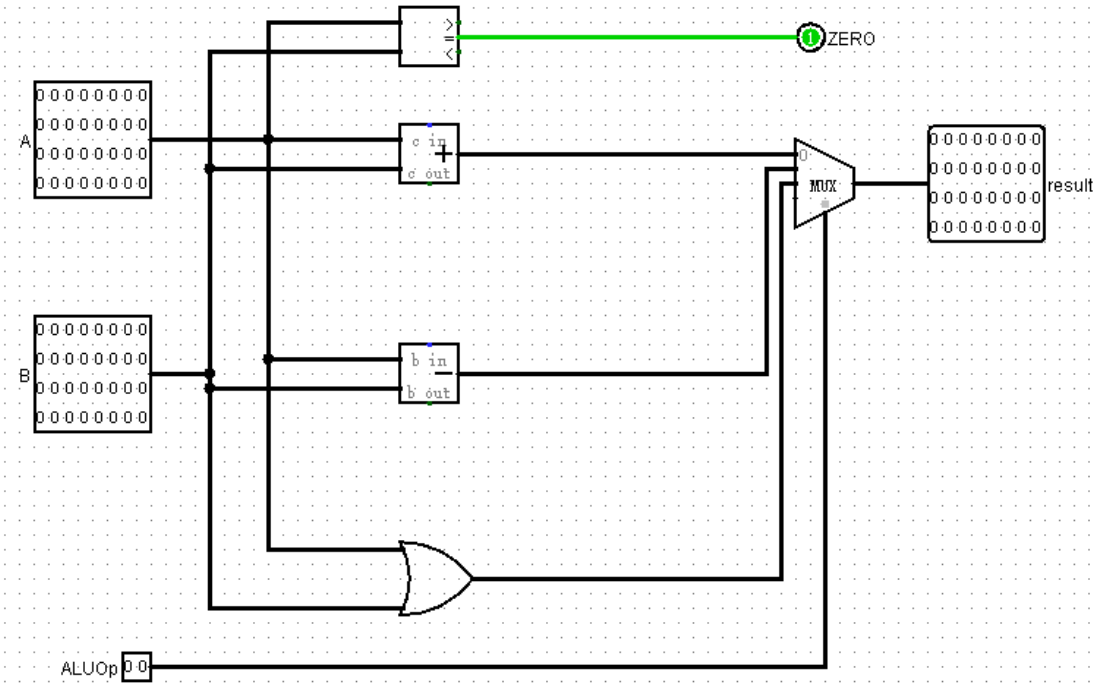
R1 连第一个 MUX，R2 连第二个 MUX。

说明：根据输入的地址决定要输出的寄存器

3、ALU（算数逻辑单元）

提供 32 位加、减、或运算及大小比较功能

可以不支持溢出（不检测溢出）。



说明：支持 32 位加、减、按位或运算及比较是否相等

ALU

信号	功能
A	第一个运算的数
B	第二个运算的数
ALUOP[1:0]	控制信号
Result	运算结果
ZERO	判断 A 与 B 是否相等，相等为 1

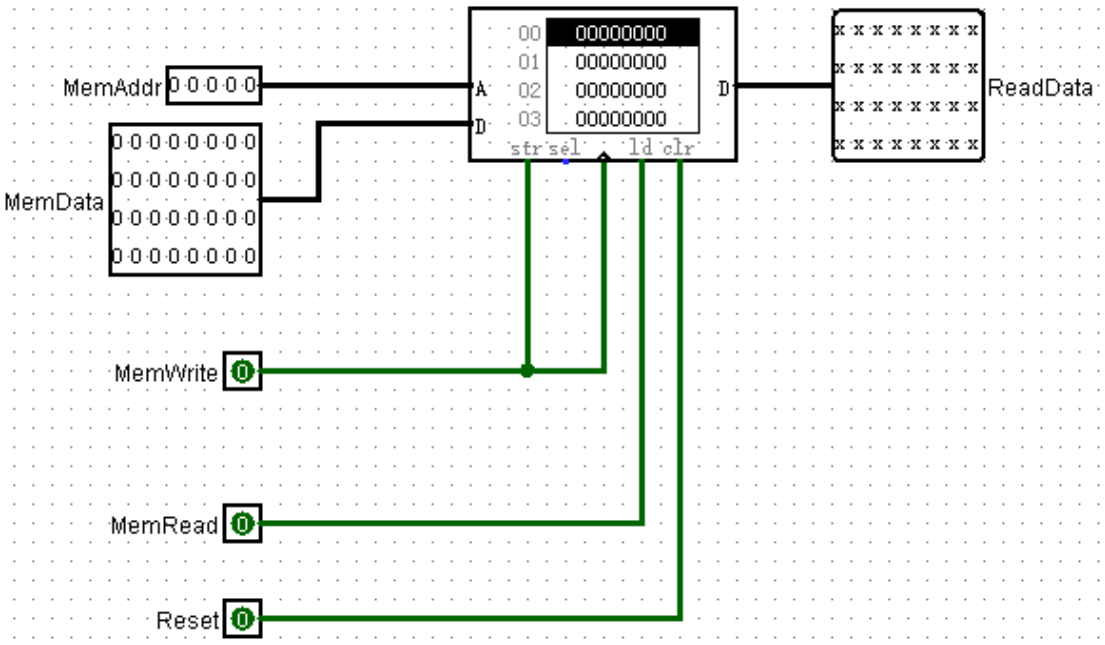
ALUOP	功能
00	加
01	减
10	或

4、DM（数据存储器）

使用 RAM 实现，容量为 32bit * 32。

起始地址：0x00000000.

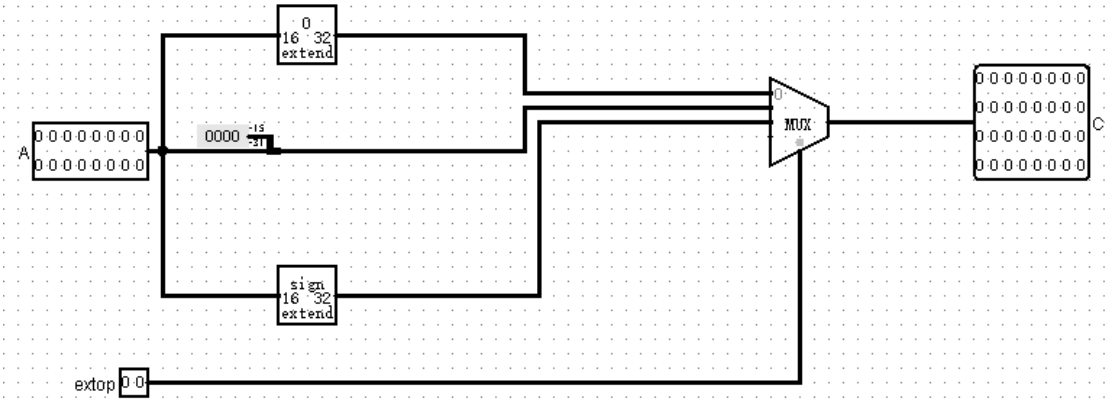
RAM 应使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports.



说明：由外界提供 32 位地址，由分线器将第三位到第七位作为 RAM 的地址。

5、EXT

可以使用 logisim 内置的 Bit Extender.



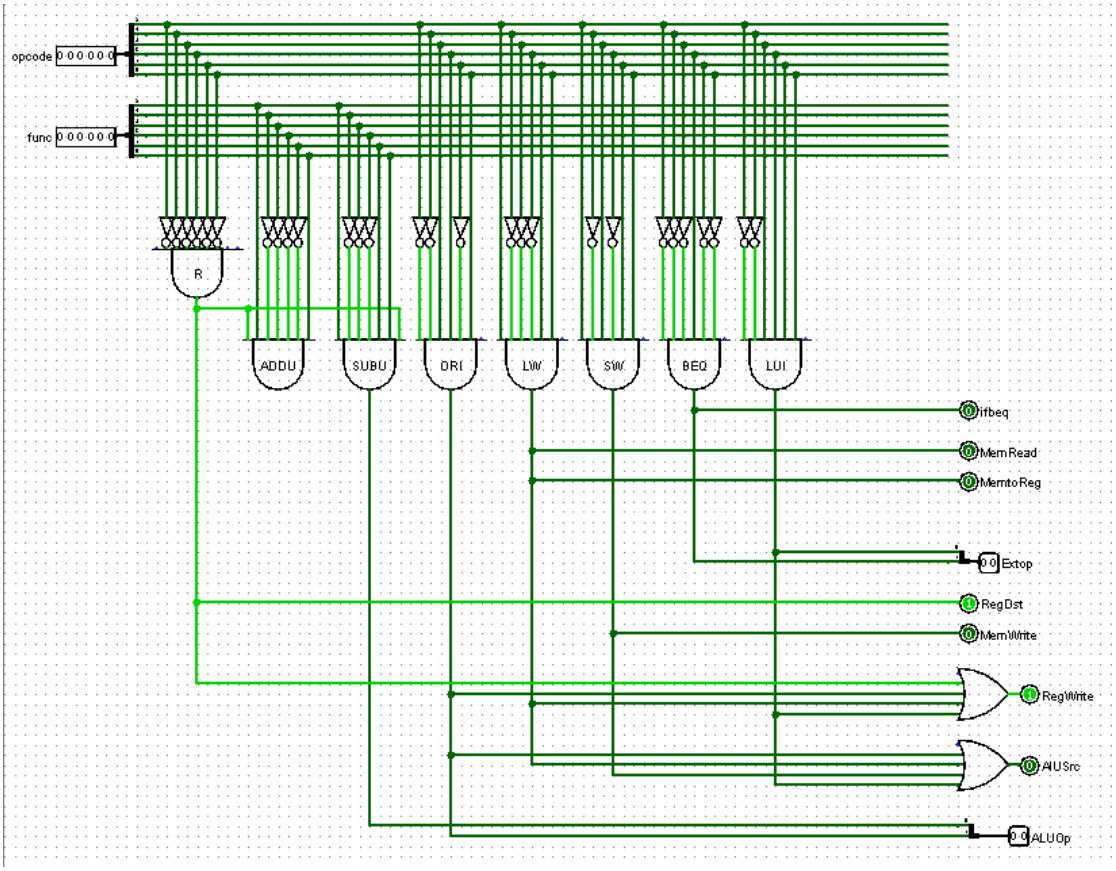
四、Controller（控制器）

控制信号	失效时作用	有效时作用
RegDst	寄存器堆写入端地址来选择Rt字段	寄存器堆写入端地址选择 Rd字段
RegWrite	无	把数据写入寄存器堆中对应寄存器
ALUSrc	ALU输入端B选择寄存器堆输出R[rt]	ALU输入端B选择Signext输出
PCSrc	PC输入源选择 PC+4	PC输入选择beq指令的目的地址
MemRead	无	数据存储器DM读数据（输出）
MemWrite	无	数据存储器DM写数据（输入）
MemtoReg	寄存器堆写入端数据来自ALU输出	寄存器堆写入端数据来自DM输出

使用与或门阵列构造控制信号。

我们把解码逻辑分解为和逻辑和或逻辑两部分：和逻辑的功能是识别，将输入的机器码识别为相应的指令；或逻辑的功能是生成，根据输入的指令的不同，产生不同的控制信号。

Func	100001	100011	n/a				
Op	000000	000000	001101	100011	101011	000100	001111
	addu	subu	ori	lw	sw	beq	Lui
nPC_sel	0	0	0	0	0	1	0
MemRead	0	0	0	1	0	0	0
MemtoReg	0	0	0	1	x	x	0
EXTOP	x	x	0	0	0	0	1
RegDst	1	1	0	0	x	x	x
MemWrite	0	0	0	0	1	0	0
RegWrite	1	1	1	1	0	0	1
ALUSrc	0	0	1	1	1	0	1
ALUOp	add	substract	or	add	add	cmp	add



说明：将输入的机器码识别为相应的指令

五、测试程序

00000000　　nop

```
3c080001        lui $8,1        $8==0x00010000
35080001        ori $8,$8,1     $8 与 0x00000001 或->$8=0x00010001
01084821        addu $9,$8,$8 $9=0x00020002
01284823        subu $9,$9,$8 $9=0x00010001
00000000        nop
00000000        nop
00000000        nop
11290000        beq $9,$9,0     if($9==$9)->0
ac090000        sw $9,0($0)     save $9 into $0
8c0a0000        lw $10,0($0)  load $10 from $0
1000ffff        beq $0,$0, 65535  end
00000000        nop
```

六、思考题

1、在上个学年的计组课程中，PC（程序计数器）位数被规定为 30 位，试分析其与 32 位 PC 的优劣。

可处理的位数更多，但是运行慢。

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。

合理。

对于指令存储器来说，指令是一开始输入好的，中间不会有改变，所以只用 ROM 就好。对于数据存储器而言，数据在不断变化着，故 RAM 合适。对于寄存器组而言，应当用寄存器来组成寄存器组。

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（表达式中只能用“与”，“或”，“非”3 种基本逻辑运算）。

RegDst=^(func);

ALUSrc=ori^lw^sw^lui;

MemtoReg=lw^sw^beq;

RegWrite=addu^subu^ori^lw^lui

nPC_Sel=beq;

ExtOp=lui;

4、充分利用真值表的 x 可以将以上的控制信号化简为最简单的表达式，请给出化简后的形式。

RegDst=^(func);

ALUSrc=ori^lw^sw^lui;

MemtoReg=lw;

RegWrite=addu^subu^ori^lw^lui

nPC_Sel=beq;

ExtOp=lui;

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。

Nop 不进行任何操作。

当真值表中的信号全为 0 则不进行操作，即代表此刻为 nop 指令。

6、前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需要再增加一个 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。

数据存储器起始地址 0x00000000 与指令存储器起始地址 0x00003000 不一样。片选信号无效则此时地址指向 IM 中的某条指令，而片选信号有效则取决于 DM 起始地址与 0x00000000 比较产生的偏移量。

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”了解相关内容后，简要阐述相比与测试，形式验证的优劣。

形式验证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。测试是通过反复试验试图查明缺陷，花相当多的时间尝试所有可能的组合，因此永远不会完整。形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验。形式验证可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现和改正电路设计中的错误，有可能缩短设计周期。