P3. logisim 设计单周期 CPU

一、实验目的

使用 logisim 设计一个能执行七条指令的单周期 CPU。

二、设计要求

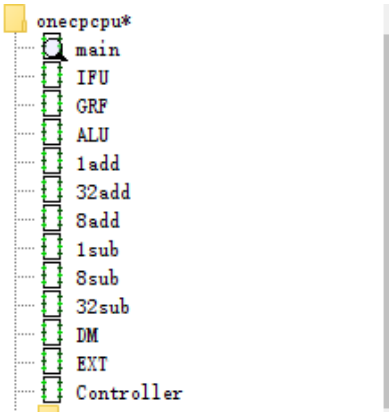
1.CPU 使用 logisim 设计。

2.CPU 能支持 addu、subu、ori、lw、sw、beq、lui、nop 指令。addu、subu 可以不支持实现溢出。

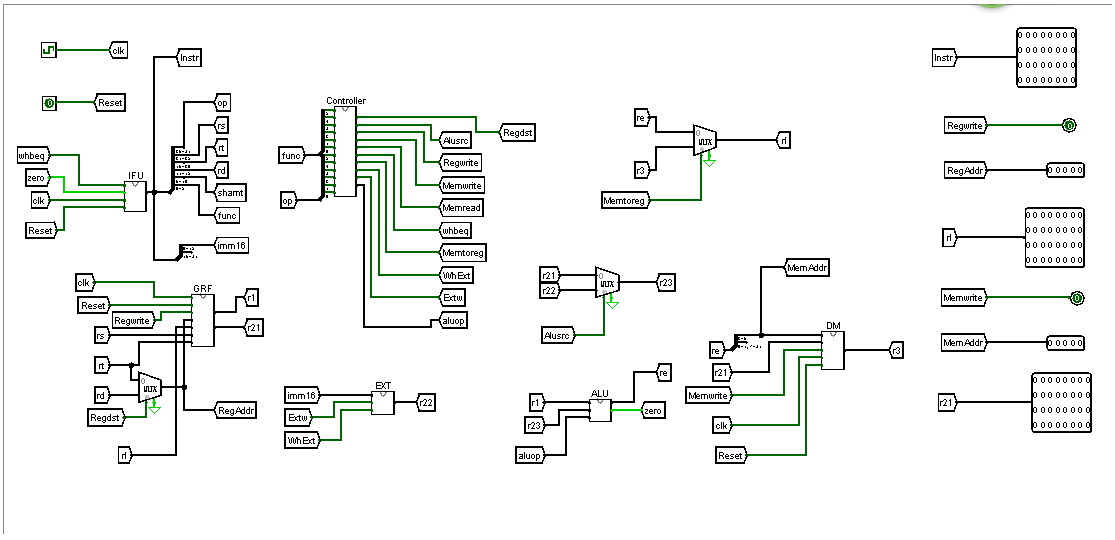
3.CPU 为单周期设计

三、实验设计

模块定义：

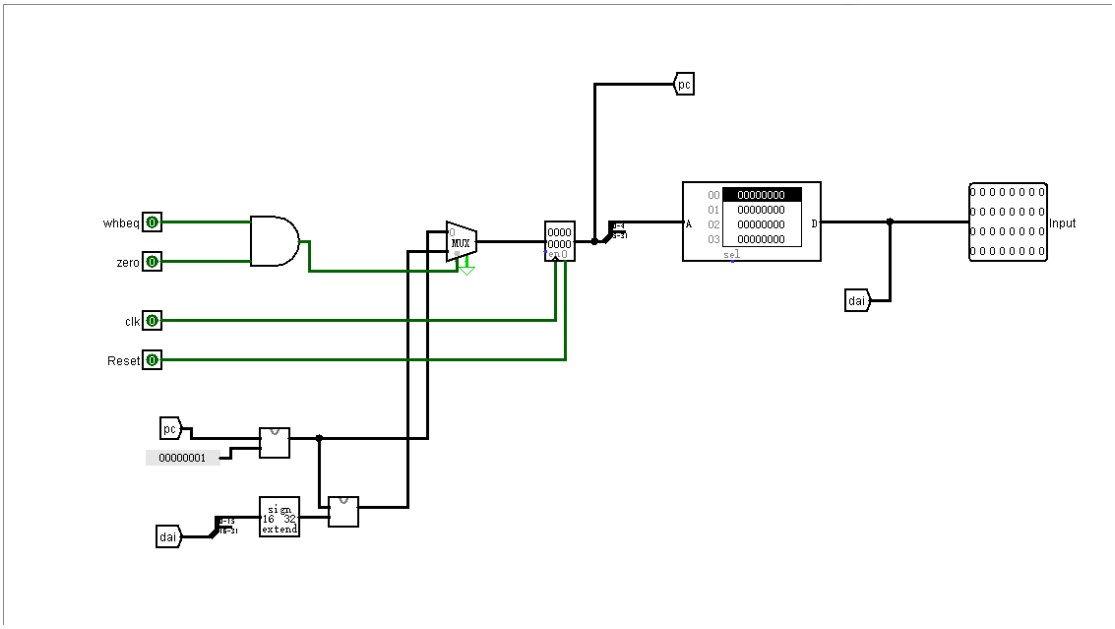


主电路：



四、各模块介绍

1. IFU 模块



（1）基本描述

IFU 主要功能是完成取指令功能，IFU 包括 PC（程序计数器）和 IM（指令存储器）以及其他逻辑部件，IFU 除了可以顺序执行取指令外，还可以根据 BEQ 指令的执行情况决定顺序取指令还是转移取指令。

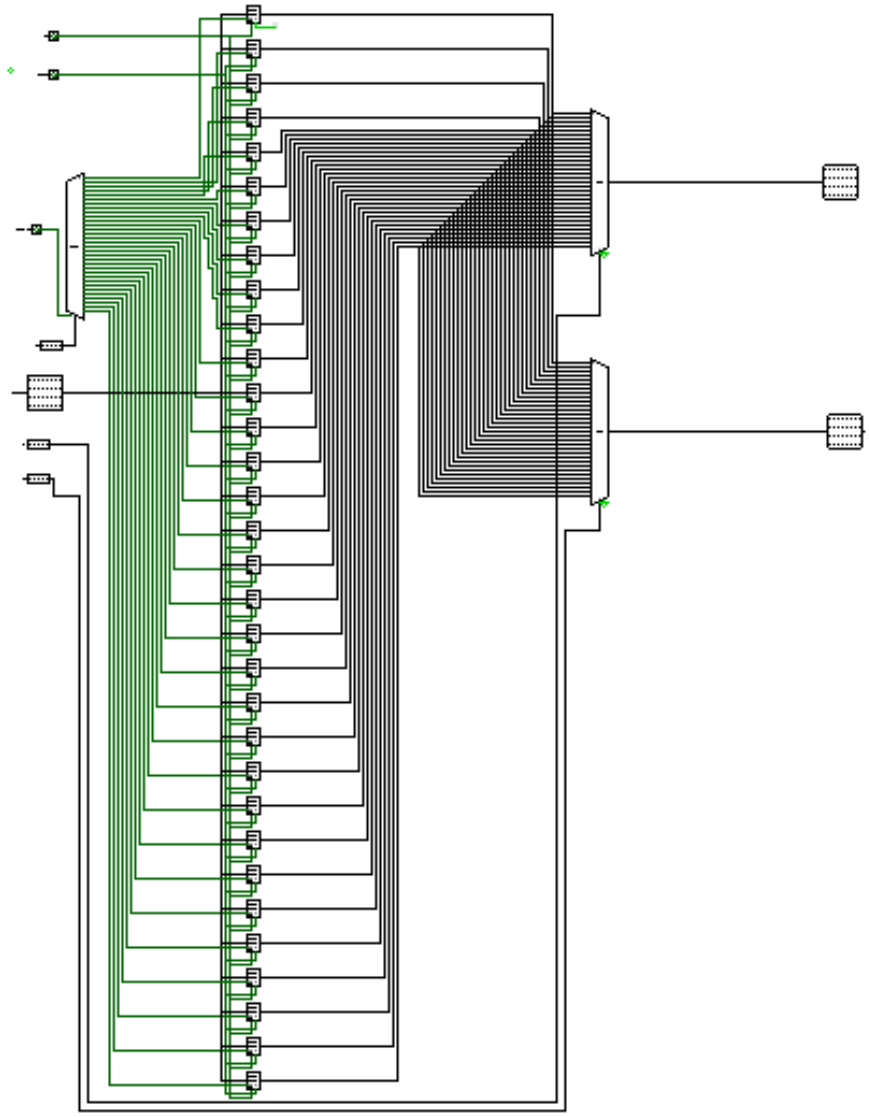
（2）模块接口

信号名	方向	描述
whbeq	I	判断当前指令是否为 beq 指令。 1：当前指令是 beq。 0：当前指令不是 beq。
zero	I	判断当前 ALU 计算结果是否为 0。 1：计算结果为 0。 0：计算结果不是 0。
clk	I	时钟信号。
Reset	I	复位信号。 1：复位信号有效。 0：复位信号无效。
Input[31:0]	O	当前的 32 位 MIPS 指令。

（3）功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，PC 被设置为 0x00000000。
2	取指令	根据 PC 中的数字取出指令。
3	计算下一条指令地址	如果当前指令不是 BEQ 指令，或者当前指令是 BEQ 指令并且 Zero 为 0，则 PC←PC+1。 如果当前指令是 BEQ 指令并且 Zero 为 1，则 PC←PC+1+sign_ext[当前指令 15：0]。 [注]：PC 取地址为 4 字节，低两位地址可以去除。

2. GRF 模块



（1）基本描述

GPR 主要功能是利用寄存器以实现对数据的取出和存入操作。通过一个 32 位 MIPS 指令对指令中的指定寄存器的值进行读或写操作，以实现对这些数据的后续操作

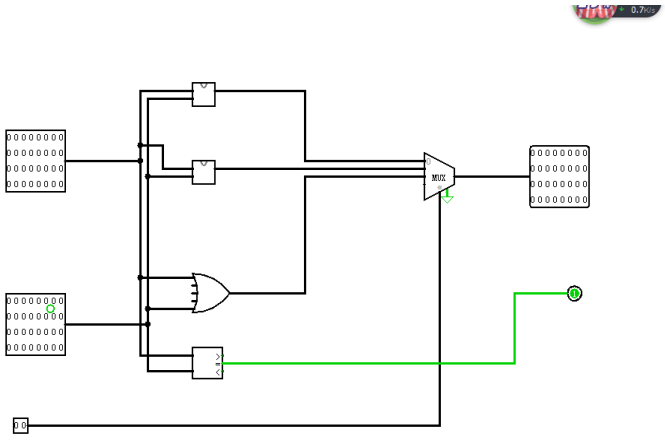
（2）模块接口

信号名	方向	描述
Regwrite	I	判断当前是否向寄存器堆写入值。 1：向寄存器里存值。 0：不向寄存器里存值。
wd[4:0]	I	写寄存器地址。
clk	I	时钟信号。
Reset	I	复位信号。 1：复位信号有效。 0：复位信号无效。
Input[31:0]	I	写入数据的输入。
Rd1	I	读入数据的寄存器地址 1。
Rd2	I	读入数据的寄存器地址 2。
Output1[31:0]	O	地址中的数据输出 1。
Output2[31:0]	O	地址中的数据输出 2。

（3）功能定义

序号	功能名称	功能描述
1	读数	取出指定寄存器中存储的数据
2	写数	将指定的 32 位数据写入指定的寄存器中。

3. ALU 模块



（1）基本描述

ALU 的功能是对指定的两个 32 位数进行无符号加、无符号减、或以及对两个数进行大小比较。

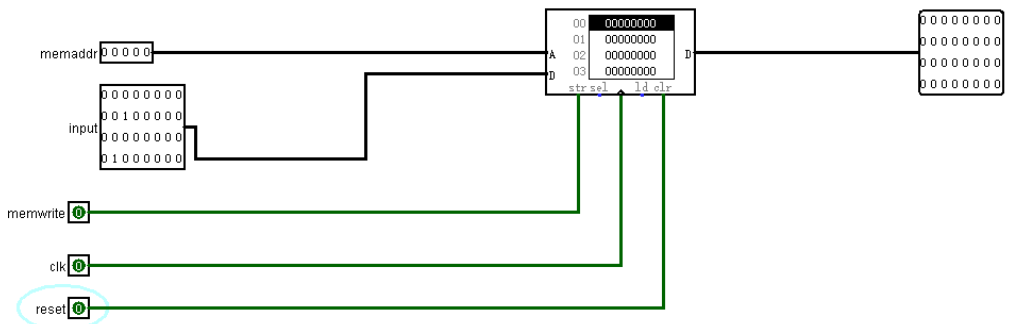
（2）模块接口

信号名	方向	描述
Input1[31:0]	I	第一个 32 位数据的输入。
Input2[31:0]	I	第二个 32 位数据的输入。
Aluop[1:0]	I	输出的控制信号。 00：输出加法结果。 01：输出减法结果。 10：输出或运算结果。
Zero	O	判断两个数是否相等。
output[31:0]	O	写入数据的输入。

（3）功能定义

序号	功能名称	功能描述
1	加法运算	对两个 32 位数据进行无符号加法运算。
2	减法运算	对两个 32 位数据进行无符号减法运算。
3	或运算	对两个 32 位数据进行或运算。
4	判断是否相等	判断两个 32 位数据是否相等

4.DM 模块



（1）基本描述

DM（数据存储器）的功能是对指定的存储器地址里的数据进行操作。并根据信号判断进行读操作还是写操作。

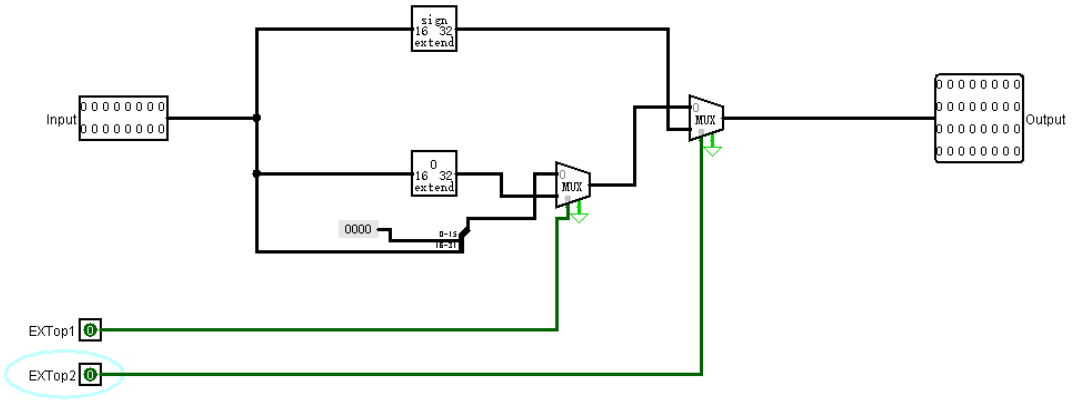
（2）模块接口

信号名	方向	描述
Input [31:0]	I	执行写操作所需的 32 位数据。
memaddr[4:0]	I	存储 32 位数据的地址。
clk	I	时钟信号
Reset	I	复位信号。
memwrite	I	判断执行写操作还是读操作。 0：读操作 1：写操作
Output[31:0]	O	读入指定地址的 32 位数据

（3）功能定义

序号	功能名称	功能描述
1	读数	Memwrite 信号为 0 时，读出指定地址中的数据。
2	写数	Memwrite 信号为 1 时，将 32 位写入指定地址中。

5. EXT 模块



（1）基本描述

EXT 的功能是对一个十六位二进制数进行扩展，并根据两个扩展信号判断进行符号扩展还是零扩展。

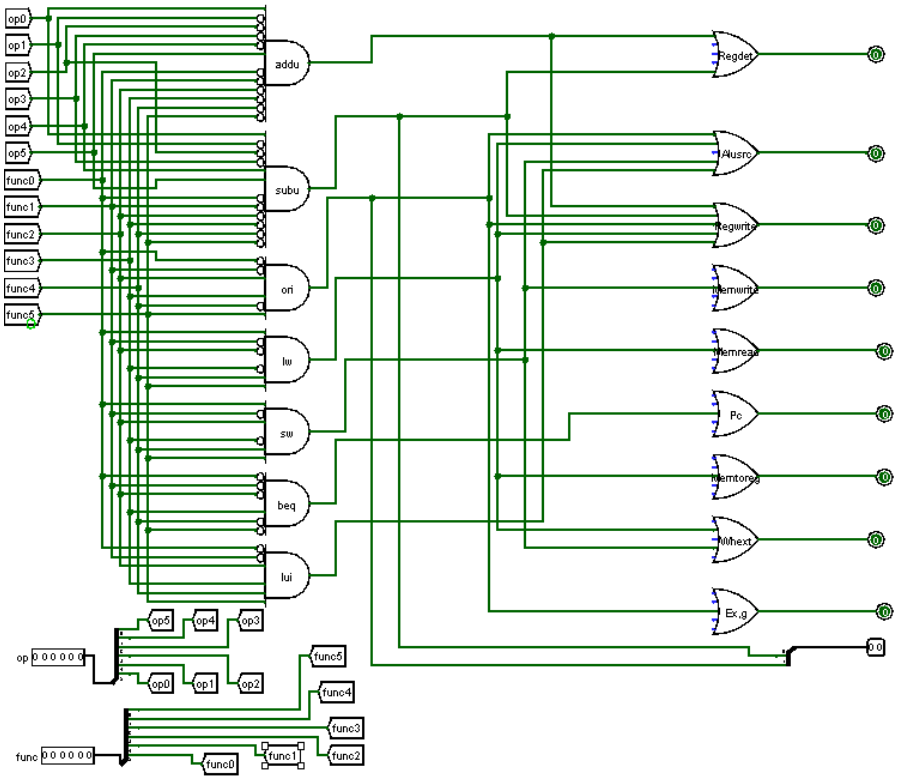
（2）模块接口

信号名	方向	描述
Input [15:0]	I	需要扩展的 16 位数据
memaddr[4:0]	I	存储 32 位数据的地址。
EXTop1	I	判断哪里补零。 0：低位补零。 1：高位补零。
EXTop2	I	选择需要输出的 32 位数据 0：已经补零的数据。 1：已经符号扩展的数据。
Output[31:0]	O	输出扩展完成的 32 位数据。

（3）功能定义

序号	功能名称	功能描述
1	低位补零	EXTop1 信号为 0 时且 EXTop2 信号为 0 时，进行低位补零。
2	高位补零	EXTop1 信号为 1 时且 EXTop2 信号为 0 时，进行高位补零。
3	符号位扩展	EXTop2 信号为 1 时，进行符号位扩展。

6. 控制器



（1）基本描述

控制器的功能是根据输入的信号对其他部件的运算进行控制。

（2）模块接口

信号名	方向	描述
Op[5:0]	I	六位 op
func[5:0]	I	六位 func。
Regdet	O	写地址控制。
Alusrc	O	ALU 操作数控制。
Regwrite	O	GRF 写控制。
Mnemwirte	O	DM 写控制。
Memread	O	DM 读控制。
Whbeq	O	判断指令是否为 beq 指令。
Memtoreg	O	GRF 写入数据控制。
WhExt	O	判断位扩展的方式。
Extw	O	判断是高位扩展还是低位扩展。
Aluop	O	判断 alu 执行的操作。

（3）功能定义

序号	功能名称	功能描述
1	总体控制	根据输入信号，判断需要控制的数据，从而得到我们想要的结果。

（4）控制器真值表

func	100001	100011					
Op	000000	000000	001101	100011	101011	000100	001111
指令	Addu	Subu	Ori	lw	sw	beq	lui
Regdet	1	1	0	0	x	x	0
Alusrc	0	0	1	1	1	0	1
Regwrite	1	1	1	1	0	0	1
Mnemwirte	0	0	0	0	1	0	0
Memread	0	0	0	1	0	0	0
Whbeq	0	0	0	0	0	1	0
Memtoreg	0	0	0	1	x	x	x
WhExt	x	x	0	1	1	x	0
Extw	x	x	1	x	x	x	0
Aluop	00	01	10	00	00	x	x

五、Mips 程序测试

lui $t0,0x36a0　#立即数 0x36a0 加载至 t0 寄存器高位。

lui $t1,0x002a　#立即数 0x002a 加载至 t1 寄存器高位。

ori $s0,$zero,0x0003　#zero 寄存器中的数与立即数 0x00000003 进行或运算，结果储存在 t3 寄存器中。

ori $s1,$zero,0x0001　#zero 寄存器中的数与立即数 0x00000001 进行或运算，结果储存在 s1 寄存器中。

ori $s2,$zero,0x0002　#zero 寄存器中的数与立即数 0x00000002 进行或运算，结果储存在 s2 寄存器中。

ori $t4,$zero,0x0004　#zero 寄存器中的数与立即数 0x00000004 进行或运算，结果储存在 t4 寄存器中。

loop:sw $t0,0($t3)　#把 t0 寄存器中的数存储到 t3 寄存器中的数存再加上偏移量 0 所指向的 RAM 中。

subu $t0,$t0,$t1　#t0 寄存器中的值减去 t1 寄存器中的值后存到 t0 寄存器中。

addu $t3,$t3,$t4　#t4 寄存器中的值加上 t3 寄存器中的值后存到 t3 寄存器中。

addu $s2,$s2,$s1　#s1 寄存器中的值加上 s2 寄存器中的值后存到 s2 寄存器中。

beq $s2,$s0,loop　#判断 s0 的值和 s2 的值是否相等，相等转 loop。

loop2:subu $t3,$t3,$t4　#t3 寄存器中的值减去 t4 寄存器中的值后存到 t3 寄存器中。

lw $t5,0($t3)　#把 t3 寄存器的值加偏移量 4 当作地址读取存储器中的值存入 t5。

ori $t6,$t5,0xc020　#zero 寄存器中的数与立即数 0x00000003 进行或运算，结果储存在 t3 寄存器中。

sw $t6,4($t3)　#把 t6 寄存器中的数存储到 t3 寄存器中的数存再加上偏移量 4 所指向的 RAM 中。

subu $s2,$s2,$s1　#s2 寄存器中的值减去 s1 寄存器中的值后存到 s2 寄存器中。

beq $s2,$s1,loop2　#判断 s2 的值和 s1 的值是否相等，相等转 loop2。

addu $t3,$t3,$t4　#t4 寄存器中的值加上 t3 寄存器中的值后存到 t3 寄存器中。

lui $s7,0x33a4　#立即数 0x33a4 加载至 s7 寄存器高位。

sw $s7,4($t3)　#把 s7 寄存器中的数存储到 t3 寄存器中的数存再加上偏移量 4 所指向的 RAM 中。

addu $s1,$s1,$s1　#s1 寄存器中的值加上 s1 寄存器中的值后存到 s1 寄存器中。

lw $s6,0($t3)　#把 t3 寄存器的值加偏移量 0 当作地址读取存储器中的值存入 s6。

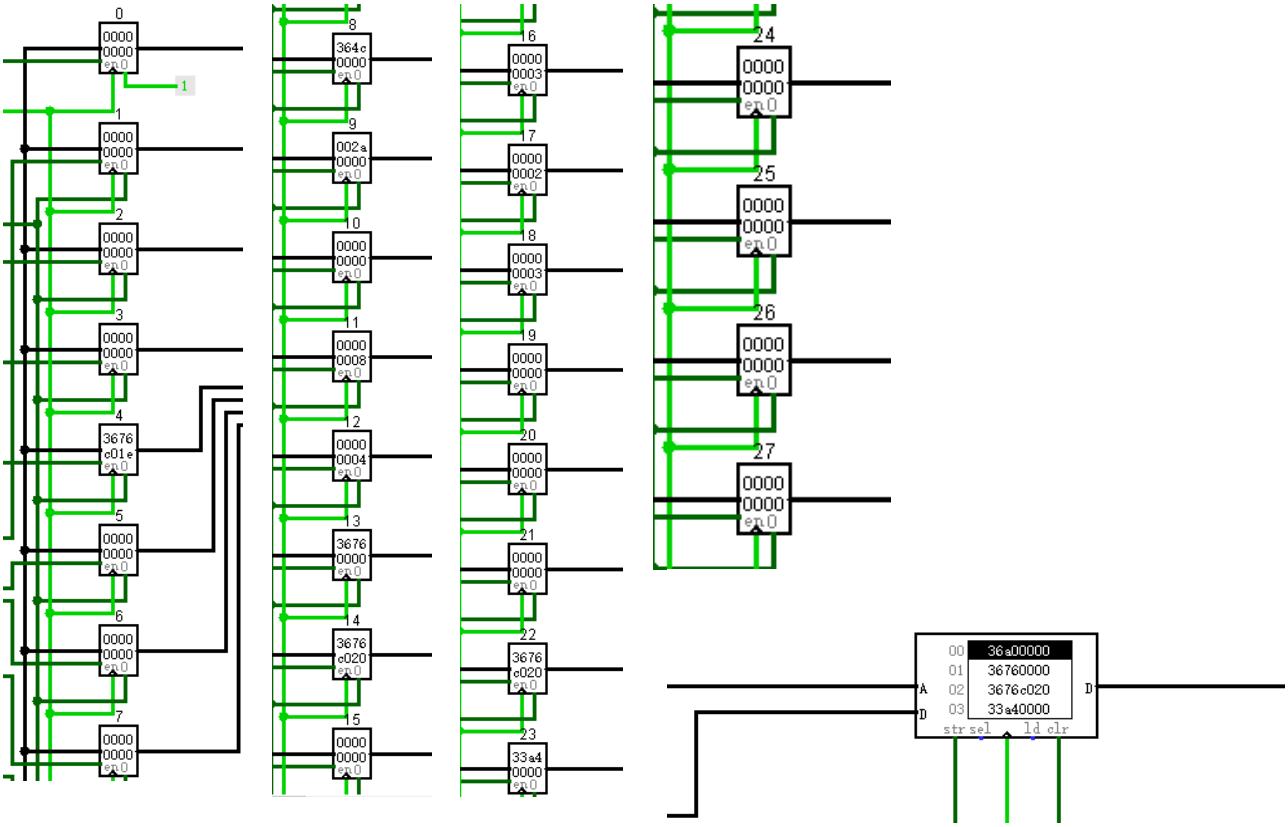
subu $a0,$s6,$s1　#s6 寄存器中的值减去 s1 寄存器中的值后存到 a0 寄存器中。

期待结果：

$zero	0	0x00000000
$at	1	0x000ac020
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x367ec01e
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x364c0000
$t1	9	0x002a0000
$t2	10	0x00000000
$t3	11	0x00000008
$t4	12	0x00000004
$t5	13	0x36760000
$t6	14	0x367ec020
$t7	15	0x00000000
$s0	16	0x00000003
$s1	17	0x00000002
$s2	18	0x00000003
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x367ec020
$s7	23	0x33a40000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x00000000	0x36a00000	0x36760000	0x3676c020	0x33a40000

实际结果：



六、思考题

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

30 位 pc 在计算下一地址时需要先扩展位数，而 32 位不需要扩展就可以直接加。

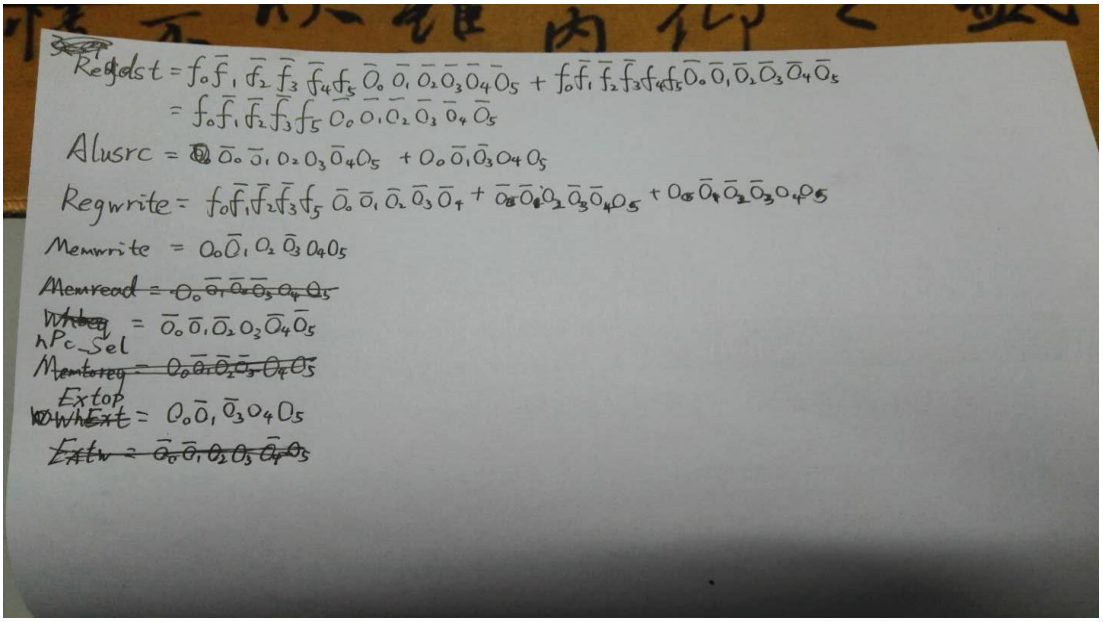
2.现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

合理，IM 只需要存储多条指令，需要只能实现读功能的 ROM 即可。

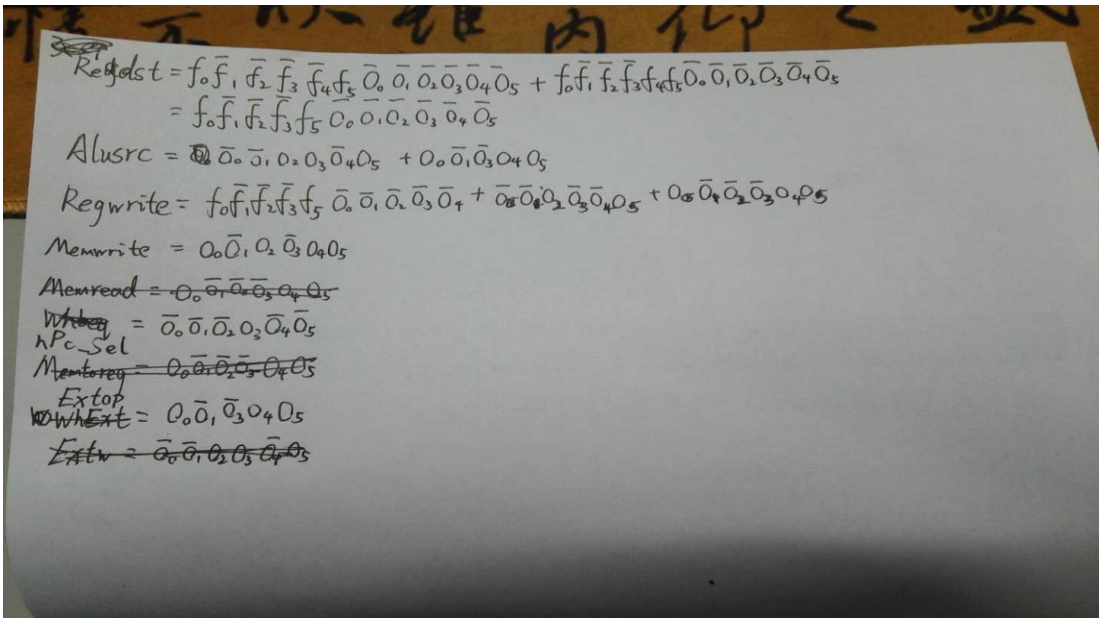
DM 需要存储多条数据，并且还要实现读功能和写功能，需要用到 RAM。

GRF 只需要能存储 32 个数，使用 32 个寄存器就可以实现。

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。）



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。



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 指令对任何元件没有操作，对电路没有影响。

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。

将地址的高四位与 0x3 进行比较，得到片选信号，当信号有效时进行存储。无效时不存储。

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。

优势：能对所有情况进行验证、能够快速发现电路中的错误，减少错误几率。

缺点：无法具体分析电路的性能。