Porject 3 logisim设计单周期cpu实验报告

一、整体介绍

1.设计的为32位单周期cpu

2.支持7条指令，分别为{addu, subu, ori, lw, sw, beq, lui, nop}

3. nop机器码为0x00000000， 即空指令，不进行任何有效行为（修改寄存器等）。

4. addu,subu可以不支持溢出。

5. 处理器为单周期设计。

5. 需要采用模块化和层次化设计。顶层有效的驱动信号要求包括且仅包括：reset (clk 请使用内置时钟模块).

二、模块介绍

1.GRF

寄存器堆IO说明

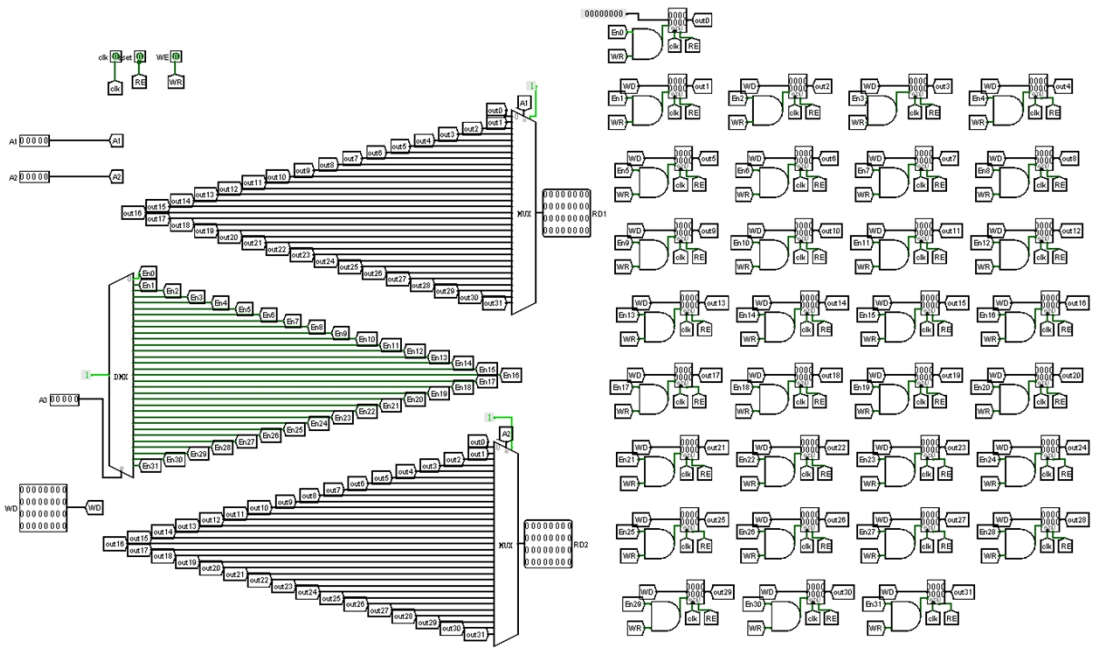
信号名	方向	功能描述
Clk	I	时钟信号
Reset	I	复位信号：1：复位 0：无效
RegWrite	I	是否可以写入寄存器堆的控制信号 其中，1：可以写入 0：不可写入
RA1[4:0]	I	读操作寄存器地址1
RA2[4:0]	I	读操作寄存器地址2
WA[4:0]	I	写操作寄存器地址

RegData [31:0]	I	写入“写入寄存器”的内容
RD1[31:0]	O	读入寄存器 1 中的内容
RD2[31:0]	O	读入寄存器 2 中的内容

GRF 功能

序号	功能名称	功能描述
1	复位	当 reset=1 时，寄存器被置为 0x00000000
2	读寄存器	根据读入寄存器地址，读出寄存器内容
3	写寄存器	根据写入寄存器地址和写入信号，向寄存器内写入内容

实现图片



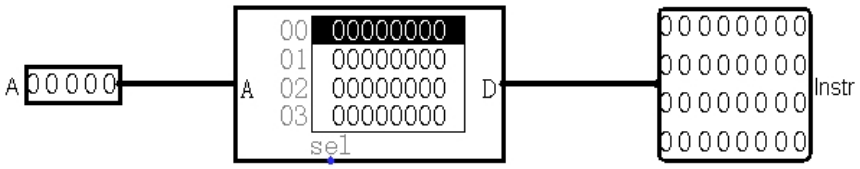
2. IM

IM 接口说明

信号名	方向	功能描述
A[4:0]	I	读入的 5 位地址
Instr[31:0]	O	输出的 32 位指令

IM 功能

序号	功能名称	功能描述
1	取指令	通过 pc 传入的 5 位地址，取出 32 位指令



3. ALU

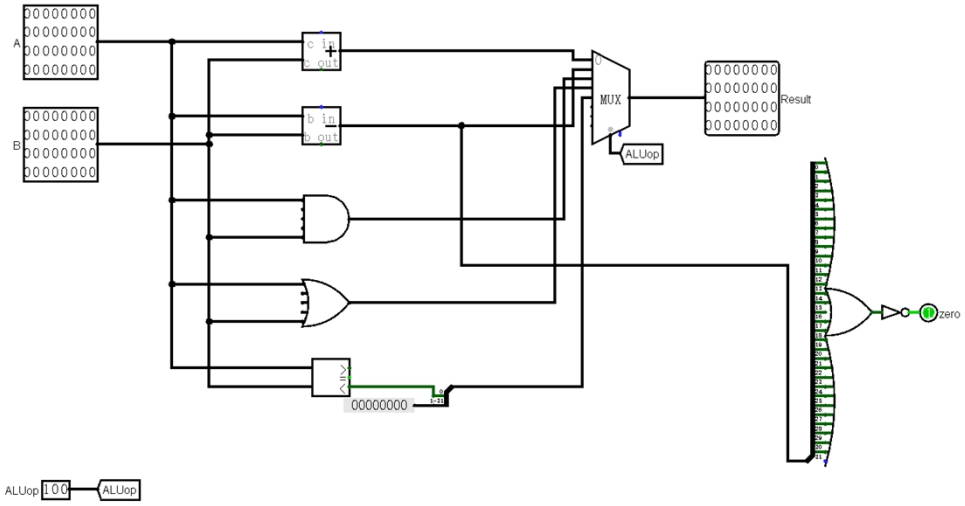
ALU 接口说明

信号名	方向	功能描述
A[31:0]	I	输入数据 1
B[31:0]	I	输入数据 2
ALUop[2:0]	I	输入选择信号 000:做加运算 001:做减运算 010:做与运算 011:做或运算 100:做比较运算
Zero	O	当 zero = 0 时，表示 A = B 当 zero = 1 时，表示 A != B
Result[31:0]	O	输出 A 与 B 做运算的 32 位结果

ALU 功能

序号	功能名称	功能描述
1	加法	Result = A + B

2	减法	Result = A - B
3	与	Result = A & B
4	或	Result = A | B
5	判断大小	若 A = B,则 zero = 1, Result = 1 若 A < B, 则 zero = 0, Result = 1 若 A > B, 则 zero = 0, Result = 1



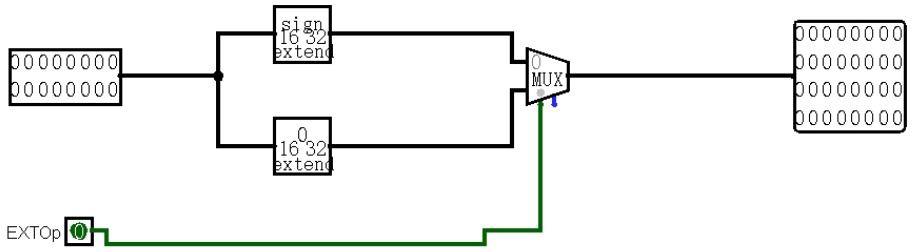
4. EXT

EXT 接口说明

信号名	方向	功能描述
in[15:0]	I	读入的 16 位待扩展数
ExtOp	I	扩展方式选择 0:有符号扩展 1:无符号扩展
Out[31:0]	O	输出的 32 位扩展结果

EXT 功能

序号	功能名称	功能描述
1	进行有符号扩展	将 In[15]从 Out[15]扩展到 Out[31]
2	进行无符号扩展	将 Out[16]到 Out[31]补 0



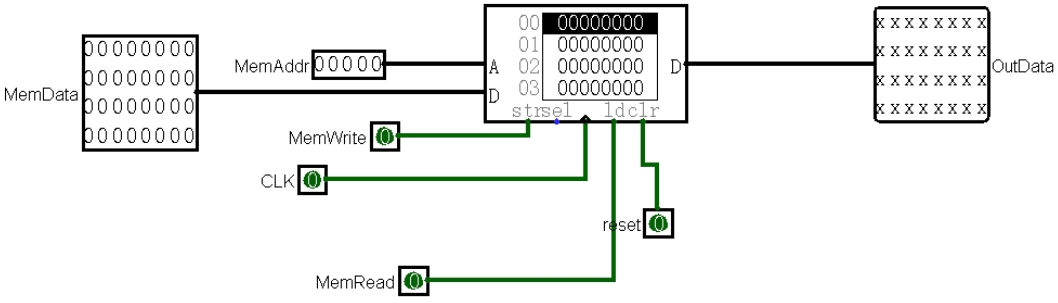
5. DM

DM 接口说明

信号名	方向	功能描述
Clk	I	时钟信号
Reset	I	复位标志
MemAddr[4:0]	I	输入数据存储器的地址
MemWrite	I	写入控制信号
MemRead	I	执行读操作
MemData[31:0]	I	写入数据存储器的内容
OutData[31:0]	0	输出的 32 位数据

DM 功能

序号	功能名称	功能描述
1	复位	将 DM 数据设置为 0X00000000
2	读	读入 DM 内数据
3	写	向 DM 内写入 32 位数据



6. IFU

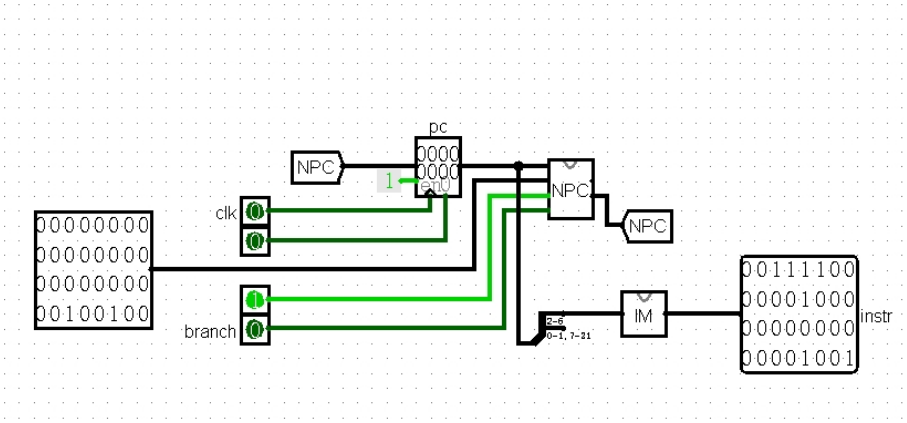
IFU 接口说明

信号名	方向	功能描述
clk	I	时钟信号
Imm[31:0]	I	32 位立即数
Zero	I	Alu 传入的 zero 信号
branch	I	branch 信号，判断是否执行 beq 指令
Re	I	重置信号
Instr[31:0]	O	从 IM 中读取指令

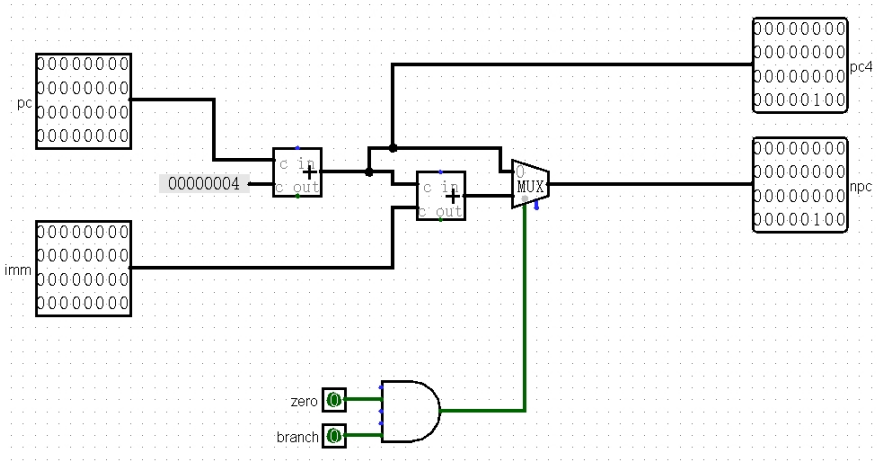
IFU 功能

序号	功能名称	功能描述
1	计算地址	计算 pc 下一个地址
2	读取指令	读取 IM 中存储的指令

图为 IFU 模块



图为 IFU 中 NPC 模块



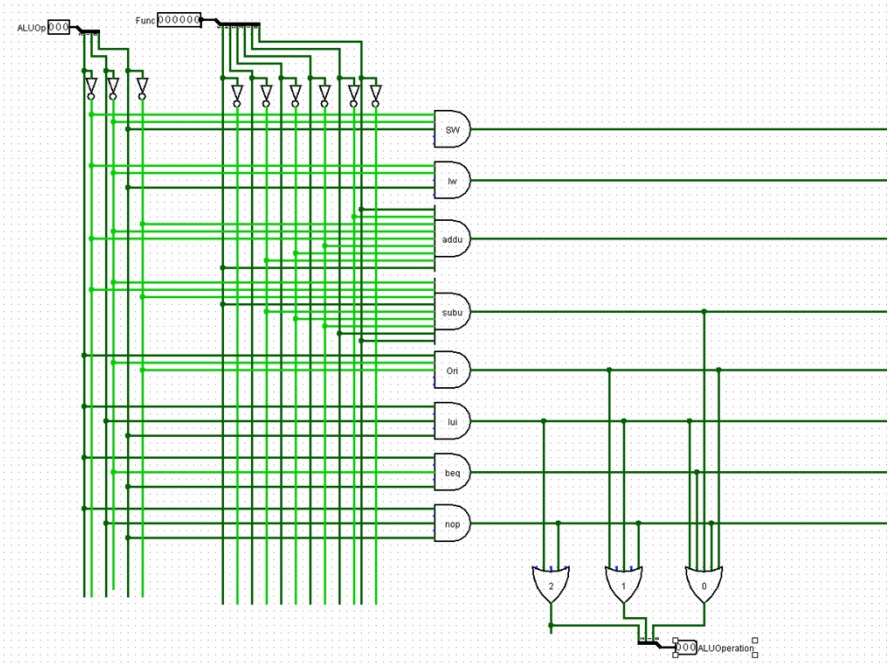
7. ALUControl

ALUControl 接口说明

信号名	方向	功能描述
ALUOp[2:0]	I	输入 3 位控制信号
Func[2:0]	I	输入 6 位 func 信号
ALUOperation[2:0]	O	计算出 ALUOperation 的 3 位信号

ALUControl 功能

序号	功能名称	功能描述
1	计算 ALUOperation 值	结合 ALUOp 和 Func 计算出 ALUOperation 的值



7.Control

信号名	方向	功能描述
Op[5:0]	I	Op 信号
RegDst	O	写地址控制 选择 RT,RD
Branch	O	判断是否为 beq 指令，是则设置为 1
MemtoReg	O	GRF 写入的选择信号
MemWrite	O	DM 写入信号
MemRead	O	DM 读入信号
ALUOp[2:0]	O	传递给 ALUcontrol 的控制信号
ALUsrc	O	ALU 第二操作数的选择信号
RegWrite	O	GRF 写入控制信号
ExtOp	O	控制 Ext 有符号/无符号扩展信号

三、控制模块设计

1.ALUControl 设计思路

	ALUOp	Func	功能	ALUOperation
Sw	001	xxxxxx	加法	000
lw	001	xxxxxx	加法	000
Addu	000	100001	加法	000
subu	000	100011	减法	001
Ori	011	xxxxxx	或	011
lui	111	xxxxxx	无	111
beq	010	xxxxxx	减法	001
nop	111	xxxxxx	无	xxx

2. Control 设计思路

1.对 RegDst：1)R 型指令，GRF 的 WR 接口选择 Rd，所以 RegDst=1

2) lw,lui,ori 指令，WR 接口选择 Rt， 故 RegDst = 0，其余时候都可以

2.branch 信号：只有指令操作为 beq 时，为 1；其余时候为 0

3.MemtoReg，当指令为 addu/subu 时候，向 GRF 传入 ALU 的值，MemtoReg=00。

当指令为 lw,向 GRF 传入 DM 的值，MemtoReg = 01。

当指令为 lui,向 GRF 传入立即数，MemtoReg=10，其余时候不做要求

4.MemWrite：只有 sw 时，需要向 DM 内写入值，MemWrite = 1

其余时候，MemWrite = 0.

5.MemRead：当指令为 lw 时候，需要读入 DM 的值，MemRead =1

其余时候，MemRead = 0

6.ExtOp： 当指令为 ori 时候，需要进行符号扩展，ExtOp =0，进行符号扩展

其余时候：ExtOp = 1，进行无符号扩展

7.RegWrite：当指令为 addu/subu/lw/loi/lui 时，需要向 GRF 写入数据，此时 RegWrite 为 1，其余时候为 0

8.AluOp：当指令为 addu/subu 时候，ALUOp 为 000

当指令为 lw/sw 需要用到 alu 加法计算地址时，ALUOp 为 001

当指令为 beq 需要 alu 进行减法运算时，ALUOp 为 010

当指令为 ori 需要 ALU 进行或运算时，ALUOp 为 011

9.ALUSrc:当指令为 lw/sw/ori 需要涉及立即数运算时，ALUSrc 为 1，其余时候为 0

	addu	subu	lw	sw	beq	lui	ori	nop
Op	000000	000000	100011	101011	000100	001111	001101	000000
RegDst	1	1	0	x	x	0	0	x
branch	0	0	0	0	1	0	0	x
MemtoReg	00	00	01	xx	xx	10	00	x
MemWrite	0	0	0	1	0	0	0	x

MemRead	0	0	1	0	0	0	0	x
ALUOp	000	000	001	001	010	111	011	111
ALUSrc	0	0	1	1	0	x	1	x
RegWrite	1	1	1	0	0	1	1	x
ExtOp	0	0	0	0	0	0	1	x

四、测试程序设计

```
.text

  lui $t0, 100 #t0 高位赋值 100

  ori $t1, $t0, 234 #T1 = t0 | 234

  addu $t2, $t1, $t0 # t2 = t1 + t0

  addu $t3, $t2, $t2 # t3 = t2 + t2

  subu $t4, $t3, $t0 #t4 = t3 - t0

  lui $a0, 100 #a0 高位赋值 100

  lui $a1, 100

  lui $v0, 400

  beq $v0, $a0, end1 #若正常运行，则此处不应跳转
```

```
  sw $v0, 4($0)#向 0x4 存入 v0 的值
  sw $v0, 8($0)
  sw $a1,12($0)
  lw $v1, 4($0)#从 0x00000004 取出 v1 的值
  lw $s5, 8($0)
  nop

  beq  $a0, $a1, end #若正常，则 s0 不应被赋值
  lw $s0,4($a0)

end1:
end:
  lui $v0, 200
```

16 进制机械码：

v2.0 raw

3c080064

350900ea

01285021

014a5821

01686023

3c040064

3c050064

3c020190

10440008

ac020004

ac020008

ac05000c

8c030004

8c150008

00000000

10850001

8c900004

3c0200c8

Mars 运行后结果：

寄存器区：

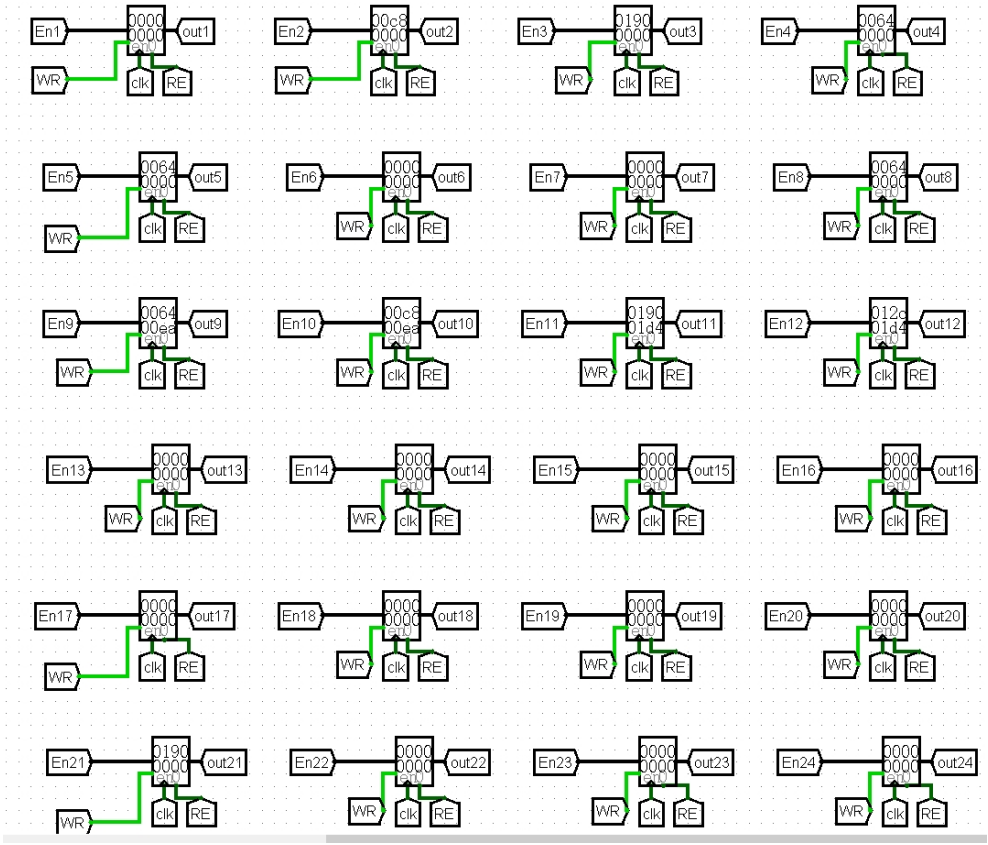
$zero	0	0
$at	1	0
$v0	2	13107200
$v1	3	26214400
$a0	4	6553600
$a1	5	6553600
$a2	6	0
$a3	7	0
$t0	8	6553600
$t1	9	6553834
$t2	10	13107434
$t3	11	26214868
$t4	12	19661268
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	26214400
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	6144
$sp	29	12284
$fp	30	0
$ra	31	0
pc		12360
hi		0
lo		0

内存区：

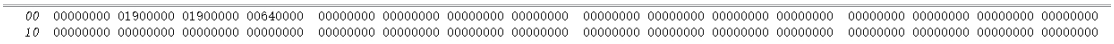
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x00000000	0	26214400	26214400	6553600	0	0	0
0x00000020	0	0	0	0	0	0	0
0x00000040	0	0	0	0	0	0	0
0x00000060	0	0	0	0	0	0	0
0x00000080	0	0	0	0	0	0	0
0x000000a0	0	0	0	0	0	0	0
0x000000c0	0	0	0	0	0	0	0
0x000000e0	0	0	0	0	0	0	0
0x00000100	0	0	0	0	0	0	0
0x00000120	0	0	0	0	0	0	0

Logisim 运行后的结果

寄存器区：



内存区：



五、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。

答：1.30 位 pc 会使存储容量变小四倍，2.执行 pc = pc + 4 时候，进行

拓展。

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。

我认为合理，ROM 为只读存储器，RAM 为内存中存储器，可读可写，力求范围大，对速度要求不高，RAM 比较适合。最后 GRF 存储，对范围要求少，对速度要求高，所以适合用寄存器

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：将 Op，func 分为 Op5,Op4,Op3,Op2,Op1,Op0 和 f5,f4,f3,f2,f1,f0

RegDst = !Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0

ALUSrc = (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

MemtoReg = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

RegWrite = (!Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0) | (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

MemWrite = (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

nPC_sel = !Op5 & !Op4 & !Op3 & Op2 & !Op1 & !Op0

ExtOp = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4

& Op3 & !Op2 & Op1 & Op0)

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。

即，为了简化可将 X 取为 0/1，在本题中将 X 均取为 0，化简后为:

RegDst = !Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0

ALUSrc = (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

MemtoReg = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

RegWrite = (!Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0) | (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

MemWrite = (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

nPC_sel = !Op5 & !Op4 & !Op3 & Op2 & !Op1 & !Op0

ExtOp = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

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 的机械码为 0X00000000，即什么都不执行，只执行 pc = pc + 4 操作，所以不需要加入真值表，因为对任何控制信号都无需求。

4. 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。

5. 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。

优点：

1.形式验证的覆盖率达到 100%，对所有可能出现的情况进行模拟

2.形式验证是利用数学上的方法，将待验证电路和功能描述进行比较，不用仿真模拟

3.形式验证是进行逻辑形式的比较，由于逻辑结构区别不大，因此验证时间短，可以快速发现错误

缺点：

不能有效地测试电路的性能，如功耗等。