Logisim 开发 MIPS 单周期处理器

一、总体设计

单周期 CPU 顶层设计图

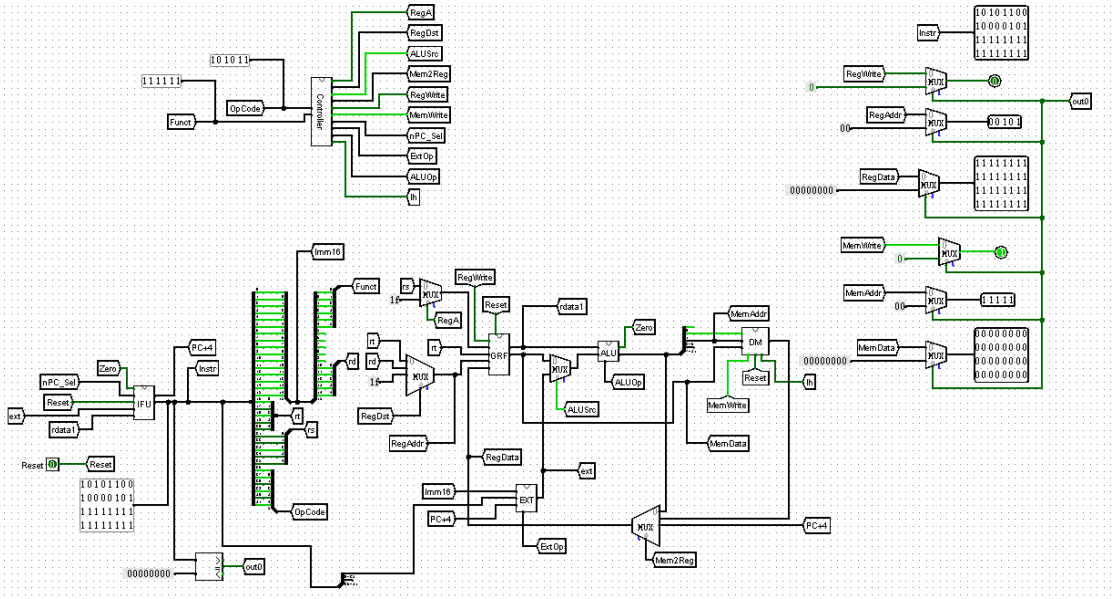

图 1 CPU 顶层设计图

表 1 工程化设计

Ins	Adder		PC	LM Add	Registers				ALU		DM		Signext	Signext2	Nadder		Shift<<2
	A	B			Reg1	Reg2	Wreg	Wdata	A	B	Add.	Wdata					
addu	PC	4	Adder	PC	Rs	Rt	Rd	ALU	Rdata1	Rdata2							
subu	PC	4	Adder	PC	Rs	Rt	Rd	ALU	Rdata1	Rdata2							
ori	PC	4	Adder	PC	Rs		Rt	ALU	Rdata1	Signext			imm16				
lw	PC	4	Adder	PC	Rs		Rt	DM	Rdata1	Signext	ALU		imm16				
sw	PC	4	Adder	PC	Rs	Rt			Rdata1	Signext	ALU	Rdata2	imm16				
beq	PC	4	dder/Nadd	PC	Rs	Rt			Rdata1	Rdata2			imm16		Adder	Shift	imm16
lui	PC	4	Adder	PC			Rt	Signext					imm16				
jal	PC	4	Signext	PC			31	Adder					imm16				
jr			Rdata1	PC	31												
j			Signext	PC									imm16				
xori	PC	4	Adder	PC	Rs		Rt	ALU	Rdata1	Signext			imm16				
lh	PC	4	Adder	PC	Rs		Rt	Signext2	Rdata1	Signext	ALU		imm16	DM			

二、模块定义

1、IFU

（1）基本描述

IFU 主要功能是完成取指令功能。IFU 内部包括 PC、IM（指令存储器）以及其他相关逻辑。IFU 除了能执行顺序取指令外，还能根据 beq 指令的执行情况决定顺序取指令还是转移取指令。

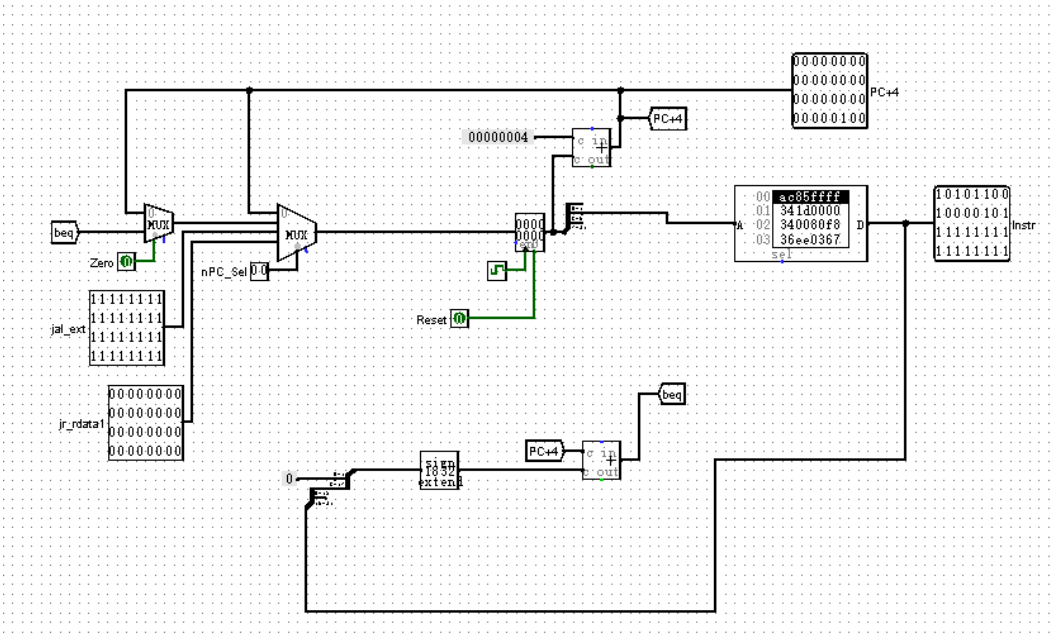

图 2 IFU

（2）模块接口

表 2 IFU 模块接口

信号名	方向	功能描述
nPC_Sel	I	判断读入指令类型 01：当前指令为 beq 10：当前指令为 jal 或 j 11：当前指令为 jr 00：其他指令
Zero	I	ALU 的输出是否等于 0 1：计算结果为 0 0：计算结果非 0
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
ext	I	Ext 的输出
rdata1	I	寄存器组的 32 位数据输出 1
PC+4	O	当前 PC 加 4
Instr[31:0]	O	32 位 MIPS 指令

（3）功能定义

表 3 IFU 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 0x00000000
2	取指令	根据 PC 从 IM 中取出指令

3	计算下一条指令地址	如果当前指令不是 beq 指令，则 PC<-PC+4 如果当前指令是 beq 指令，并且 zero 为 1，则 PC<-PC+sign_ext

2、GRF

（1）基本描述

通用寄存器可用于传送和暂存数据，也可参与算术逻辑运算，并保存运算结果。

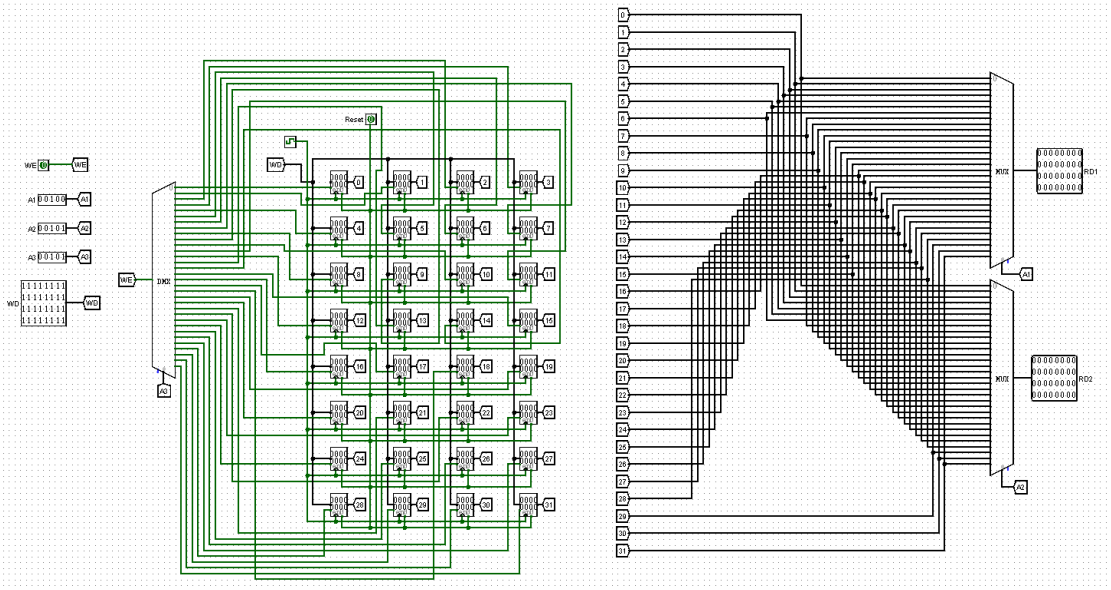

图 3 GRF

（2）模块接口

表 4 GRF 模块接口

信号名	方向	功能描述
A1[4:0]	I	读寄存器地址 1
A2[4:0]	I	读寄存器地址 2
A3[4:0]	I	写寄存器地址
WD[31:0]	I	写入数据的输入
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
WE	I	写使能信号 1：写操作 0：读操作
RD1[31:0]	O	32 位数据输出 1
RD2[31:0]	O	32 位数据输出 2

（3）功能定义

表 5 GRF 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有寄存器被设置为 0x00000000
2	读寄存器	根据输入的寄存器地址读出数据
3	写寄存器	根据输入的地址，把输入的数据写进选中的寄存器

3、ALU

（1）基本描述

ALU 是算数逻辑单元，是能实现多组算术运算和逻辑运算的组合逻辑电路。

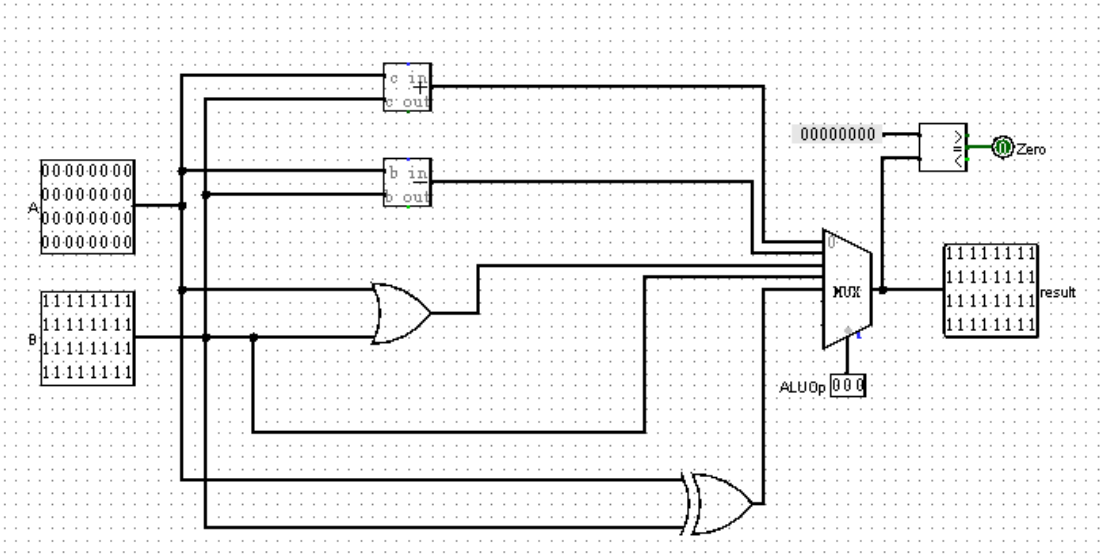

图 4 ALU

（2）模块接口

表 6 ALU 模块接口

信号名	方向	功能描述
A[31:0]	I	32 位输入数据 1
B[31:0]	I	32 位输入数据 2
ALUOp[2:0]	I	控制信号 000：加法 001：减法 010：或运算 011：32 位输入数据 2 100：异或运算
C[31:0]	O	32 位输出数据
Zero	O	ALU 的计算结果是否为 0

（3）功能定义

表 7 ALU 功能定义

序号	功能名称	功能描述
1	加	A + B
2	减	A - B
3	或	A | B
4	异或	A ⊕ B

4、DM

（1）基本描述

数据存储器是用于存放程序运行的中间处理数据的，可随程序运行而随时写入或读出数据存储器的内容。

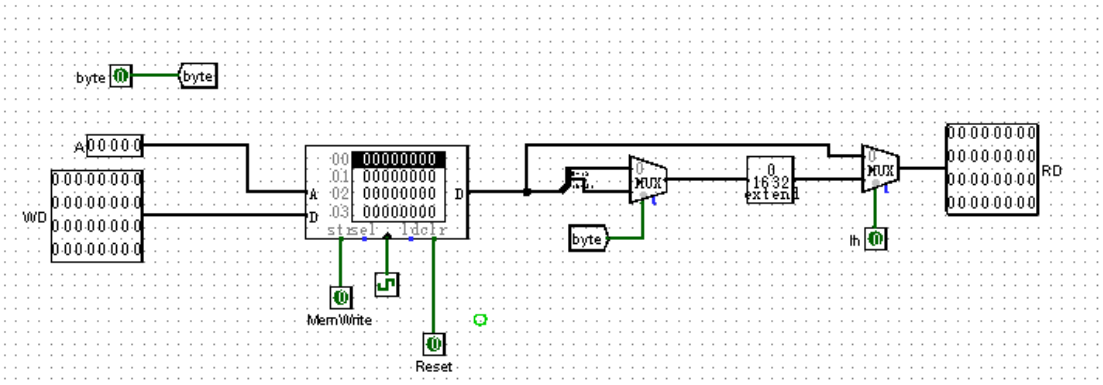

图 5 DM

（2）模块接口

表 8 DM 模块接口

信号名	方向	功能描述
A[4:0]	I	写入寄存器地址
WD[31:0]	I	32 位输入数据
byte	I	半字地址
Clk	I	时钟信号
MemWrite	I	写入控制信号 0：读操作 1：写操作
lh	I	是否为 lh 指令
Reset	I	复位信号 1：复位 0：无效
RD[31:0]	O	32 位输出数据

（3）功能定义

表 9 DM 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有数据被设置为 0x00000000

2	读	根据输入的寄存器地址读出数据
3	写	根据输入的地址，把输入的数据写入

5、EXT

（1）基本描述

对数据进行需要的位扩展，分为高位扩展和低位扩展。

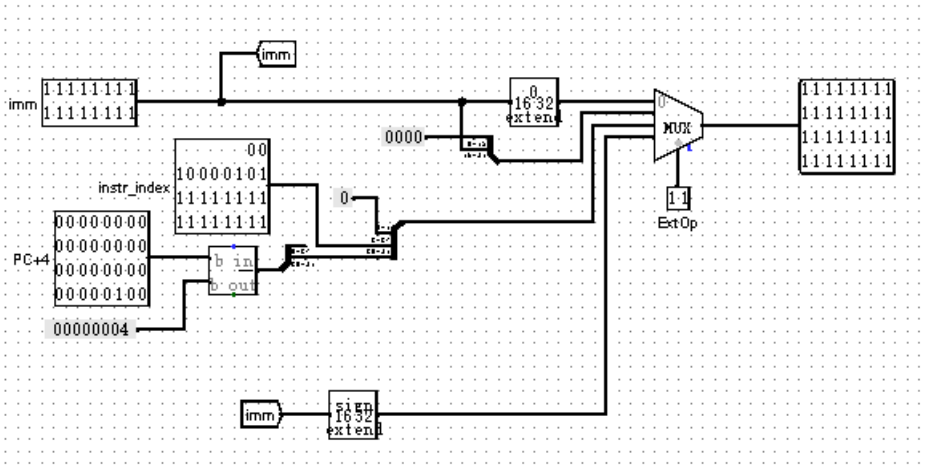

图 6 EXT

（2）模块接口

表 10 EXT 模块接口

信号名	方向	功能描述
A[15:0]	I	16 位输入数据
ExtOp[1:0]	I	控制信号 0：高位补 0 1：低位补 0
PC+4	I	IFU 输出的 PC+4
instr[25:0]	I	指令第 25 位到第 0 位
B[31:0]	O	32 位输出数据

（3）功能定义

表 11 EXT 功能定义

序号	功能名称	功能描述
1	高位扩展	高 16 位补 0
2	低位扩展	低 16 位补 0

6、Controller

（1）基本描述

控制器将每一条机器指令中包含的信息，转化为给 CPU 各部分的控制信号。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。

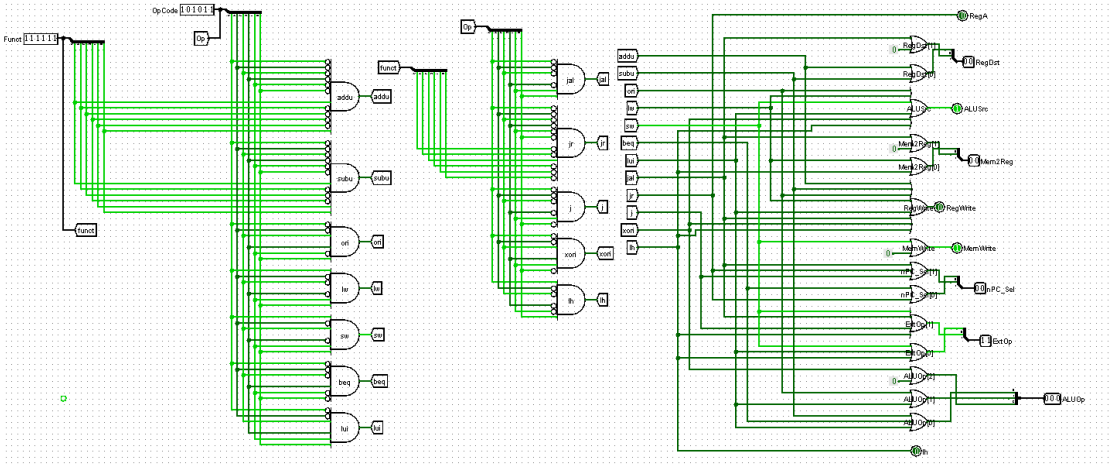

图 7 Controller

（2）模块接口

表 12 Controller 模块接口

信号名	方向	功能描述
OpCode[5:0]	I	6 位 op
Funct[5:0]	I	6 位 funtion
RegA	O	读寄存器地址 1 的选择输入
RegDst[1:0]	O	写地址控制信号
ALUSrc	O	CPU 第二操作数选择控制
Mem2Reg[1:0]	O	WD 读入数据控制信号
RegWrite	O	GRF 读写控制信号
MemWrite	O	DM 写控制中写入 GPR 的数据选择信号
nPC_Sel[1:0]	O	指令判断信号
ExtOp[1:0]	O	Ext 控制信号
ALUOp[2:0]	O	ALU 控制信号
lh	O	lh 指令判断信号

三、控制器设计

表 13 控制器真值表设计_1

Funct[5:0]	100001	100011				
OpCode[5:0]	000000		001101	100011	101011	000100
	addu	subu	ori	lw	sw	beq
RegA	0	0	0	0	0	0
RegDst	01	01	00	00	x	x
ALUSrc	0	0	1	1	1	0

Mem2Reg	00	00	00	01	x	x
RegWrite	1	1	1	1	0	0
MemWrite	0	0	0	0	1	x
nPC_Sel	00	00	00	00	00	01
ExtOp	x	x	00	00	11	x
ALUOp	000	001	010	000	000	001
lh	0	0	0	0	0	0

表 13 控制器真值表设计_2

Funct[5:0]			001000			
OpCode[5:0]	001111	000011	000000	000010	001110	100001
	lui	jal	jr	j	xori	lh
RegA	x	x	1	x	0	0
RegDst	00	10	x	x	00	00
ALUSrc	1	x	x	x	1	1
Mem2Reg	00	10	x	x	00	01
RegWrite	1	1	0	x	1	1
MemWrite	x	x	x	x	x	0
nPC_Sel	00	10	11	10	00	00
ExtOp	01	10	x	10	00	11
ALUOp	011	x	x	x	100	000
lh	0	0	0	0	0	1

四、处理器测试

#测试 ori 指令

```
ori $a0,$0,123 #test reg0 && reg4
ori $a1,$a0,456 #test return of $a0 && reg5
ori $0,$a1,789 #load reg0
ori $t0,$0,0 #test reg0
```

#测试 lui 指令

```
lui $a2,123        #符号位为 0
lui $s0,0xffff     #符号位为 1
ori $s1,$s0,0xffff # $s0 = -1 && test reg16 && reg17
```

#测试 addu 指令

```
ori $t0,$0,123 #test reg8
ori $t1,$0,456 #test reg9
addu $t2,$t0,$t1 #test ++ && reg10
```

#测试 subu 指令

```
ori $t3,$0,123 #test reg11
ori $t4,$0,456 #test reg12
```

```
subu $t5,$t3,$t4 #test reg13 && - - = negative
subu $t6,$t4,$t3 #test reg14 && - - = positive

#测试 sw 指令 && reg15 - reg23
ori $t0,$0,0x0000 #base && test 15
sw $s2,0($t0)
sw $s3,4($t0)
sw $s4,8($t0)
sw $s5,12($t0)
sw $s6,16($t0)
sw $s7,20($t0)


#测试 lw 指令 && test reg24 && reg25
lw $t8,0($t0)
lw $t9,12($t0)
sw $a0,28($t0) #retest sw
sw $a1,32($t0)

#测试 beq 指令
ori $a0,$0,1
ori $a1,$0,2
ori $a2,$0,1
beq $a0,$a1,loop1 #unequal
beq $a0,$a2,loop2 #equal

loop1: sw $a0,36($t0)
nop #测试 nop 指令
jr $31 #测试 jr 指令
loop2: sw $a1,40($t0)

#测试 jal 指令
jal loop1

#测试 j 指令
j next1
#测试 xori 指令
next2:
xori $t7,$a0,2
xori $t7,$a0,3

#测试 lh 指令
next1:
lh $t2,($t0)  #后半字
lh $t2,2($t0) #前半字
```

反馈：经以上指令测试，该单周期 CPU 均能输出正确的结果。

五、思考题

（1）模块规格

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：在简单的 MIPS 单周期处理器中，用 30 位的 PC 和用 32 位的 PC 并没有本质的差别。

缺点：但 MIPS 是一个 32 位体系结构，一般都采用 32 位的数据路径。在需要对该处理器进行复杂化的时候，32 位的 PC 会有更好的可移植性，而 30 位 PC 的可移植性相较之就会差一些。

优点：30 位对总线的需求要小于 32 位 PC，在成本上可能会稍低一些。

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。根据 IM、DM、GRF 的需求，其中 IM 只需要从地址中读取指令；DM 用来向内存中存储数据，需要足够的储存空间，同时需要能写入数据；GRF 做为寄存器堆，需要较快的存储速度。ROM 是只读存储器，恰好满足 IM 的要求；RAM 是随机存取存储器，可以随时读写，能够满足 DM 的需求；寄存器能高速存储，适用于 GRF。

（2）控制器设计

① 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：

$$\text{RegDst} = \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,\overline{f_4}\,f_5 + \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,f_4 f_5$$

$$= \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,f_5$$

$$\text{ALUSrc} = \overline{o_1}\,\overline{o_2}\,o_3 o_4\,\overline{o_5}\,o_6 + o_1\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,o_5 o_6 + o_1\,\overline{o_2}\,o_3\,\overline{o_4}\,o_5 o_6 + \overline{o_1}\,\overline{o_2}\,o_3 o_4 o_5 o_6$$

$$= \overline{o_1}\,\overline{o_2}\,o_3 o_4 o_6 + o_1\,\overline{o_2}\;\overline{o_4}\,o_5 o_6$$

$$\text{MemtoReg} = o_1\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,o_5 o_6$$

$$\text{RegWrite} = \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,\overline{f_4}\,f_5 + \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,f_4 f_5$$

$$+ \overline{o_1}\,\overline{o_2}\,o_3 o_4\,\overline{o_5}\,o_6 + o_1\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,o_5 o_6 + \overline{o_1}\,\overline{o_2}\,o_3 o_4 o_5 o_6$$

$$= \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,f_5 + \overline{o_1}\,\overline{o_2}\,o_3 o_4\,o_6 + o_1\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,o_5 o_6$$

$$\text{nPC\_Sel} = \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,o_4\overline{o_5}\,\overline{o_6}$$

$$\text{ExtOp} = \overline{o_1}\,\overline{o_2}\,o_3 o_4 o_5 o_6$$

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。

答：

$$\text{RegDst} = \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,f_5$$

$$\text{ALUSrc} = \overline{o_1}\,\overline{o_2}\,o_3 o_4 o_6 + o_1\,\overline{o_2}\,\overline{o_4}\,o_5 o_6$$

$$\text{MemtoReg} = o_1\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,o_5 o_6$$

$$\text{RegWrite} = \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,\overline{o_5}\,\overline{o_6}\,f_1\,\overline{f_2}\,\overline{f_3}\,f_5 + \overline{o_1}\,\overline{o_2}\,o_3 o_4\,o_6 + o_1\,\overline{o_2}\,\overline{o_3}\,\overline{o_4}\,o_5 o_6$$

$$\text{nPC\_Sel} = \overline{o_1}\,\overline{o_2}\,\overline{o_3}\,o_4\overline{o_5}\,\overline{o_6}$$

$$\text{ExtOp} = \overline{o_1}\,\overline{o_2}\,o_3 o_4 o_5 o_6$$

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。

答：

因为 nop 空指令并不会改变任何一个模块，控制信号不会对其操作产生影响。

（3）测试 CPU

① 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：

由于在 MIPS 中指令存储器是从地址 0x00003000 开始储存，如果我们的数据地址达到 0x00003000 时，而在我们的处理器中会存在数据地址和指令的存储地址重合的现象。加入一个比较数据地址的前四位和 0x00003000 的大小的片选

信号便能解决需要手工修改指令码的问题。当接收到的地址小于 0x00003000 时选择原路径，大于或等于是进行数据偏移。

② 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：

形式验证的优点：

a.由于形式验证技术是借用数学上的方法将待验证电路的功能描述或参考设计直接进行比较，因此测试者不必考虑如何获得测试向量。

b.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，因此有效地克服了测试验证的不足。

c.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。

形式验证缺点：

形式验证到目前为止仍然不能有效的验证电路的性能，如电路的时延和功耗等。