P3 实验设计报告

18373085 张海渝

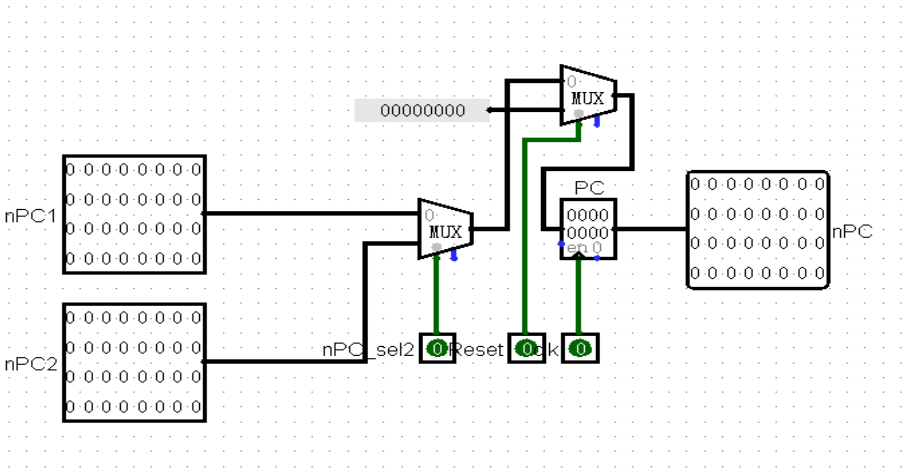
一. 模块规格

1. NPC：（程序计数器）

序号	功能名称	方向	功能描述
1	nPC1[31:0]	I	为 PC+4
2	nPC2[31:0]	I	为 PC+4+偏移量
3	nPC_sel2[31:0]	I	选择下一 PC 值
4	Reset	I	同步复位
5	Clk	I	时钟信号
6	NPC[31:0]	O	下一 PC 值

模块解释：

1. 通过选择器进行先选择下一 PC 值。
2. 如果 Reset 信号有效时，用选择器将 PC 值复位。



图一：NPC 模块图

2. IM：（指令获取）

序号	功能名称	方向	功能描述
1	PC[4:0]	I	指令地址

2	Instr	O	输出的机器码

模块解释：

1. PC[4:0]是当前 PC 值的第 6-2 位(因为 ROM 地址为 5 位，并且 PC 每次加 4，相当于 ROM 加 1)。

2. 用 ROM 存储机器码。

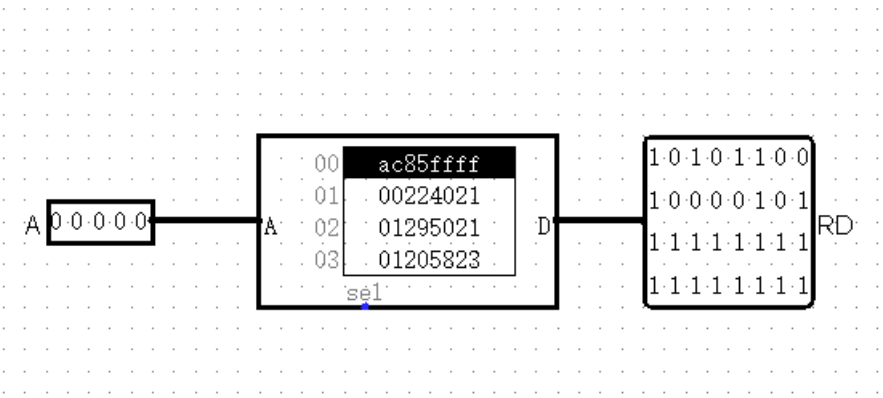


图 2：IM 模块图

3. GRF：（通用寄存器组，也称寄存器文件、寄存器堆）

序号	功能名称	方向	功能描述
1	A1[4:0]	I	机器码的第 25-21 位，寄存器编号
2	A2[4:0]	I	机器码的第 20-16 位，寄存器编号
3	A3[4:0]	I	需要改变的寄存器
4	WD3[31:0]	I	寄存器需要改变为的值
5	Reset	I	同步复位信号
6	Clk	I	时钟信号
7	RegWrite	I	是否写入寄存器信号
8	RD1[31:0]	O	A1 寄存器中的值
9	RD2[31:0]	O	A2 寄存器中的值

模块解释：

1.0 号寄存器始终为 0，不改变。

2.一共有 32 个带有使能端的寄存器。

4. ALU：（算数逻辑单元）

序号	功能名称	方向	功能描述

1	A[31:0]	I	进行操作的第一个数
2	B[31:0]	I	进行操作的第二个数
3	ALUctr[2:0]	I	进行操作的方式
4	ALUresult[31:0]	O	计算后的结果

模块解释：

1. ALUctr=0：输出 0 值

   ALUctr=1：加法

   ALUctr=2：减法

   ALUctr=3：或（or）操作

   ALUctr=4:输出 B 的值

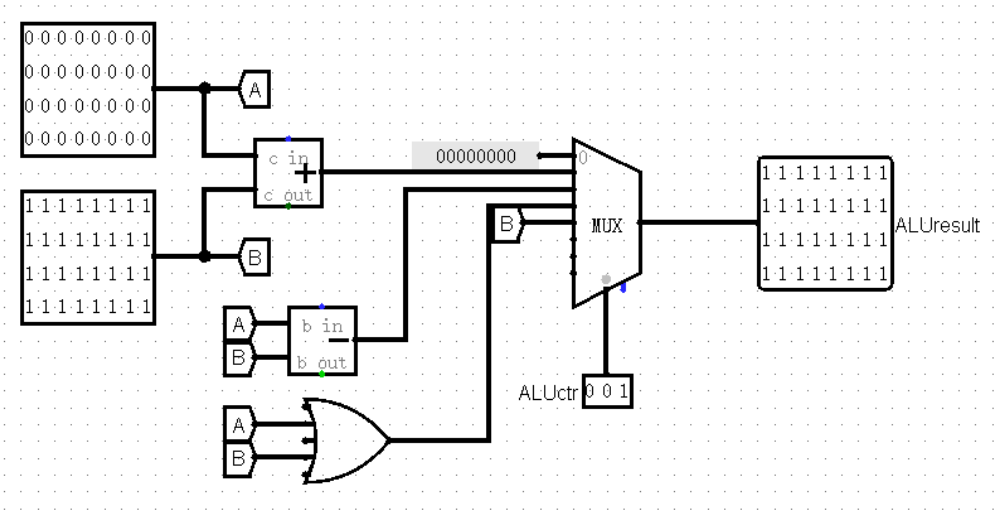


图 3：ALU 模块

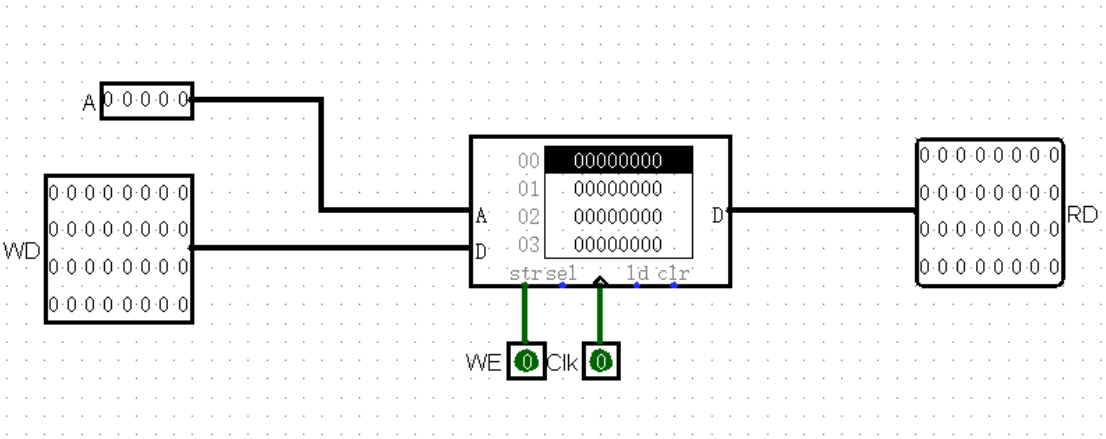
5. DM：（数据存储器）

序号	功能名称	方向	功能描述
1	A[4:0]	I	进行操作的数据的地址
2	WD[31:0]	I	数据
3	WE	I	控制是否 RAM 是否工作
4	Clk	I	时钟信号
5	RD	O	读出的数据

模块解释：

1.此模块通过 RAM 存储数据。

2.RAM 的地址也为 5 位，所以进行操作的地址为 ALU 计算结果的 6-2 位。



图四：DM 模块图

6.EXT：（数据扩展器）

序号	功能名称	方向	功能描述
1	IMM[15:0]	I	进行扩展的数
2	ExtOp[1:0]	I	扩展的方式
3	ExtIMM[31:0]	O	扩展后的数据

模块解释：

1. ExtOp=0：前 16 位 0 扩展

   ExtOp=1：符号扩展至 32 位

   ExtOp=2：后 16 位补 0 扩展

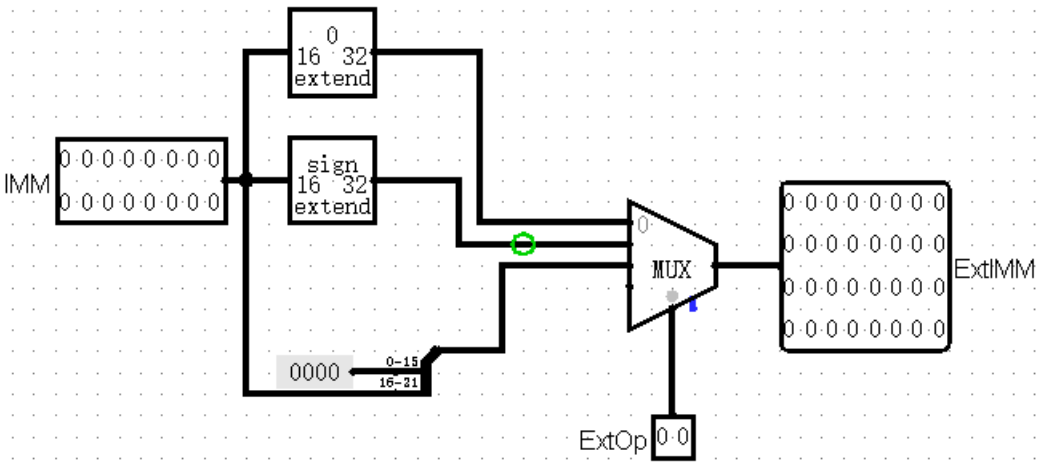


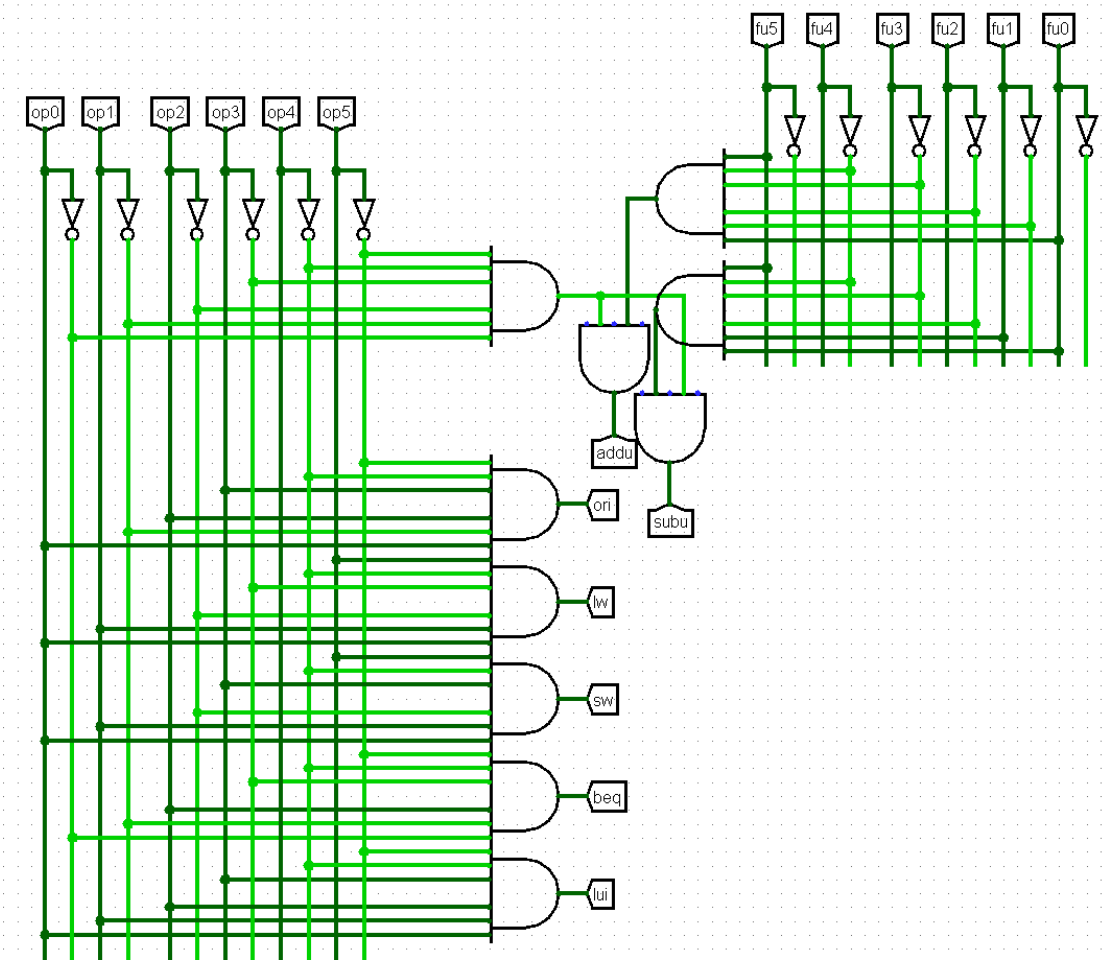
图 5：EXT 模块图

7. Control_Unit：（控制器）

序号	功能名称	方向	功能描述
1	opcode[5:0]	I	Special
2	funct[5:0]	I	Function
3	RegDst	O	选择需要在 GRF 中写入数据的寄存器
4	ALUSrc	O	选择进行 ALU 操作的第二个操作数
5	MemtoReg	O	选择存入 GRF 中寄存器的数据
6	RegWrite	O	是否在 GRF 中写入控制信号
7	MemWrite	O	DM 中的 RAM 使能端控制信号
8	nPC_sel	O	选择下一 PC 值信号
9	ExtOp[1:0]	O	数据扩展方式
10	ALUctr[2:0]	O	ALU 进行操作的方式

模块解释：

1. 首先通过 opcode 与 funct 选择出此时进行操作的指令方式(与)。



2. 选择出指令的方式以后用真值表输出相应的信号值。

funct	100000	100010					
opcode	000000	000000	001101	100011	101011	000100	001111
	add	sub	ori	lw	sw	beq	lui
RegDst	1	1	0	0	x	x	0
ALUScr	0	0	1	1	1	0	1
MemtoReg	0	0	0	1	x	x	0
RegWrite	1	1	1	1	0	0	1
MemWrite	0	0	0	0	1	0	0
nPC_sel	0	0	0	0	0	1	0
ExtOp[1:0]	x	x	0	1	1	x	2
ALUctr[2:0]	1	2	3	1	1	2	4

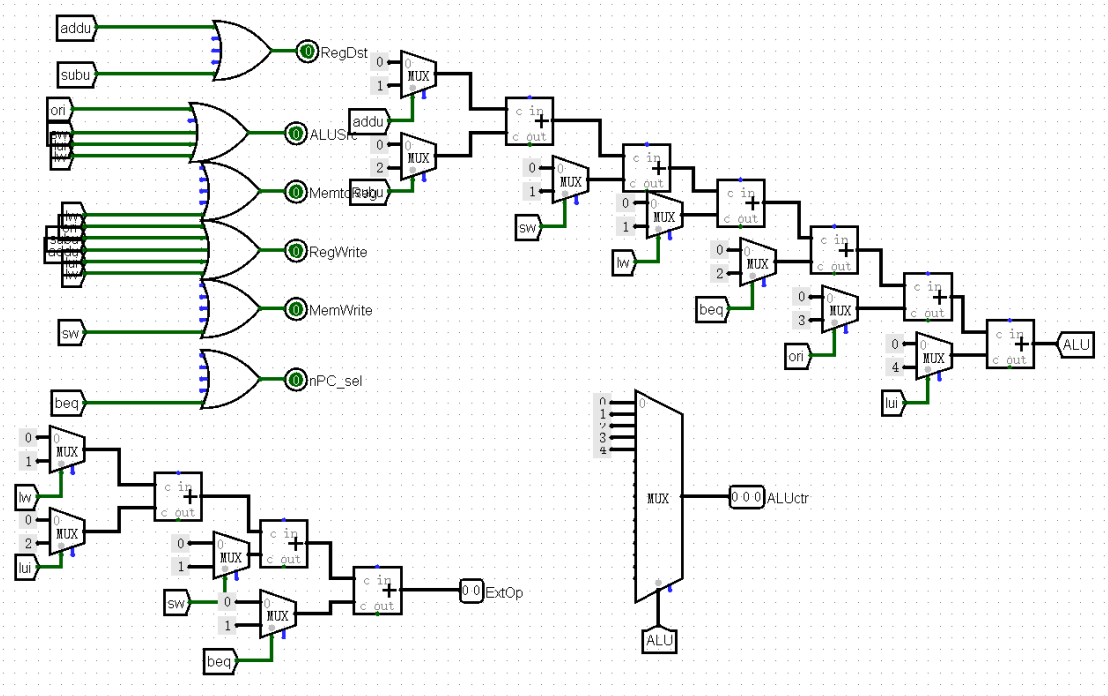


图 6：信号输出

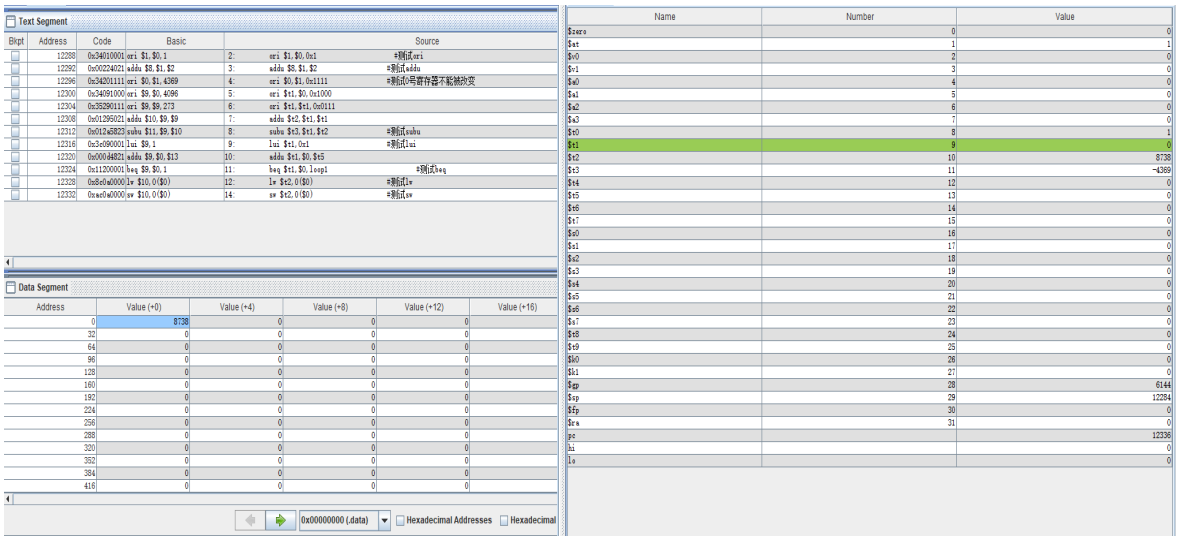
二． 测试代码

MIPS 测试：

代码：

```
.text
        ori $1,$0,0x1                   #测试ori
        addu $8,$1,$2                   #测试addu
        ori $0,$1,0x1111                #测试0号寄存器不能被改变
        ori $t1,$0,0x1000
        ori $t1,$t1,0x0111
        addu $t2,$t1,$t1
        subu $t3,$t1,$t2                #测试subu
        lui $t1,0x1                     #测试lui
        addu $t1,$0,$t5
        beq $t1,$0,loop1                        #测试beq
        lw $t2,0($0)                    #测试lw
loop1:
        sw $t2,0($0)                    #测试sw
```

结果：



Text Segment

Bkpt	Address	Code	Basic		Source
	12288	0x34010001	ori $1, $0, 1	2: ori $1, $0, 0x1	#测试ori
	12292	0x00224021	addu $8, $1, $2	3: addu $8, $1, $2	#测试addu
	12296	0x34201111	ori $0, $1, 4369	4: ori $0, $1, 0x1111	#测试0号寄存器不能被改变
	12300	0x34091000	ori $9, $0, 4096	5: ori $t1, $0, 0x1000	
	12304	0x35290111	ori $9, $9, 273	6: ori $t1, $t1, 0x0111	
	12308	0x01295021	addu $10, $9, $9	7: addu $t2, $t1, $t1	
	12312	0x012a5823	subu $11, $9, $10	8: subu $t3, $t1, $t2	#测试subu
	12316	0x3c090001	lui $9, 1	9: lui $t1, 0x1	#测试lui
	12320	0x00044821	addu $9, $0, $13	10: addu $t1, $0, $t5	
	12324	0x11200001	beq $9, $0, 1	11: beq $t1, $0, loop1	#测试beq
	12328	0x8e0a0000	lw $10, 0($0)	12: lw $t2, 0($0)	#测试lw
	12332	0xae0a0000	sw $10, 0($0)	14: sw $t2, 0($0)	#测试sw

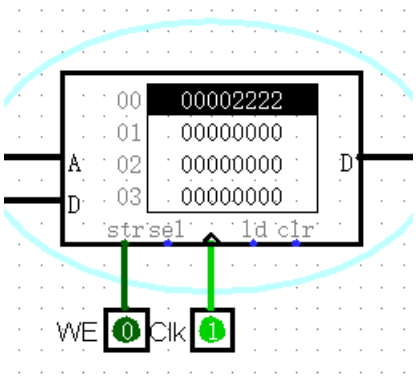
Data Segment

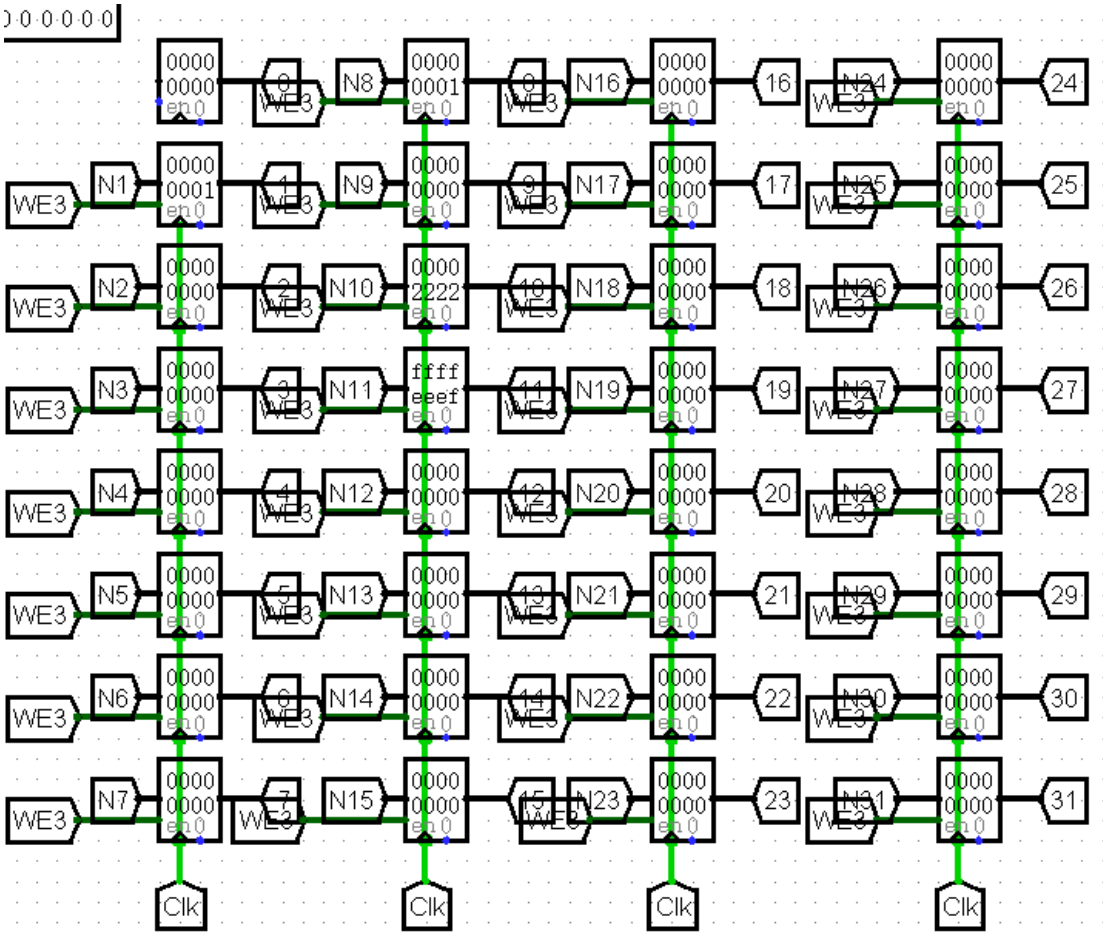
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)
0	8738	0	0	0	
32	0	0	0	0	
64	0	0	0	0	
96	0	0	0	0	
128	0	0	0	0	
160	0	0	0	0	
192	0	0	0	0	
224	0	0	0	0	
256	0	0	0	0	
288	0	0	0	0	
320	0	0	0	0	
352	0	0	0	0	
384	0	0	0	0	
416	0	0	0	0	

0x00000000 (.data) Hexadecimal Addresses Hexadecimal

Name	Number	Value
$zero	0	0
$at	1	1
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	1
$t1	9	0
$t2	10	8738
$t3	11	-4369
$t4	12	0
$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	0
$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		12336
hi		0
lo		0

CPU：





三． 思考题

(LO,T2)

1.对于一个 sw 指令来说，如果用 PC 等于 32 位时能比 30 位的时候多出两位，可以表示更多的指令，并且对于 32 位来说刚好为一个字，方便进行操作。劣势：32 位太长，有很多无效的编码，浪费空间。

2.我认为合理。第一 IM 使用 ROM 只能读不能写入的，所以符合指令集这一特征，避免了意外的修改。第二对于 DM 使用 RAM 这是因为 RAM 即可以读又可以写，并且空间大，符合了数据存储器这一特征。第三对于 GRF，这是因为，只是有 32 个寄存器，所以使用寄存器不仅可以方便寻址，还可以节约空间,可以快速访问。

（LO,T3）：

1. 布尔表达式：

RegDst=~op5 ~op4 ~op3 ~op2 ~op1 ~op0

ALUSrc=~op5 ~op4 op3 op2 ~op1 op0 + 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 + ~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[0]=op5 ~op4 ~op3 ~op2 op1 op0 + op5 ~op4 op3 ~op2 op1 op0

ExtOp[1]=~op5 ~op4 op3 op2 op1 op0

2. 化简：

RegDst=~op0

ALUSrc=op0

MemtoReg=

RegWrite= ~op3 ~op2 + ~op5 op3

MemWrite= op3 ~op2

nPC_sel= ~op3 op2

3. 因为此时机器码全为 0，此时 RegWrite，MemWrite 都为 0，所以不会影响数据的改变，又因为 nPC_sel 为 0，所以 PC 的值也就是加 4 不会产生影响，所以没关系。

(LO,T4)：

1. 即如果数据开头的地址不是 0 的时候需要将地址进行修改，所以可以多加一个 DM 片选信号，所以就可以通过将偏移量加上这个地址与偏移量之间进行选择，如果需要改变的时候置 DM 片选信号为 1，其余时候为 0 即可。
2. 优势：形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了 100%；形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试激励；形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期

劣势：这种方法需要进行大量的数学推演，容易出错，没有机器验算更加准确，当设计的电路很大的时候，很容易出现一些 BUG，而且这种方法的工作量会大于相比于测试这一种方法。