计算机组成 CPU 设计文档-P3

目录

一、 整体结构与概览

1、CPU 基本参数与指标

处理器类型：单周期

处理器字长：32 位

处理器支持指令集：{addu, subu, ori, lw, sw, beq, lui, sll(nop)}

顶层封装模块端口：

表格 1 顶层封装模块端口

端口名	类型	描述
Reset	In	接受异步复位信号，对 PC、GRF 和 Mem 复位。
Instr[31:0]	Out	输出当前指令 32 位二进制码
RegWrite	Out	输出当前 GRF 写使能信号
RegAddr[4:0]	Out	输出当前 GRF 写入地址
RegData[31:0]	Out	输出当前 GRF 写入数据
MemWrite	Out	输出当前 Mem 写使能信号
MemAddr[4:0]	Out	输出当前 Mem 写入地址（按字单位编址）
MemData[31:0]	Out	输出当前 Mem 写入数据

2、CPU 模块结构

a) 数据通路

1、IFU（取指令单元）:包括 PC 和存放指令的 ROM，用于输出当前指令码。

2、GRF（通用寄存器组）：内含 32 个寄存器，支持对寄存器值的读写。

3、ALU（算术逻辑单元）：运算执行部件，对 32 位数执行多种运算。

4、DM（数据存储器）：存储数据部件，支持读写。

5、EXT（位扩展器）：将 16 位数扩展为 32 位数，支持有/无符号扩展。

b) 控制信号

1、控制器：识别指令并生成 CPU 各部分的控制信号，使用逻辑阵列实现。

2、ALU 控制器 :识别控制器信号和指令，驱动 ALU 输出对应的运算结果。

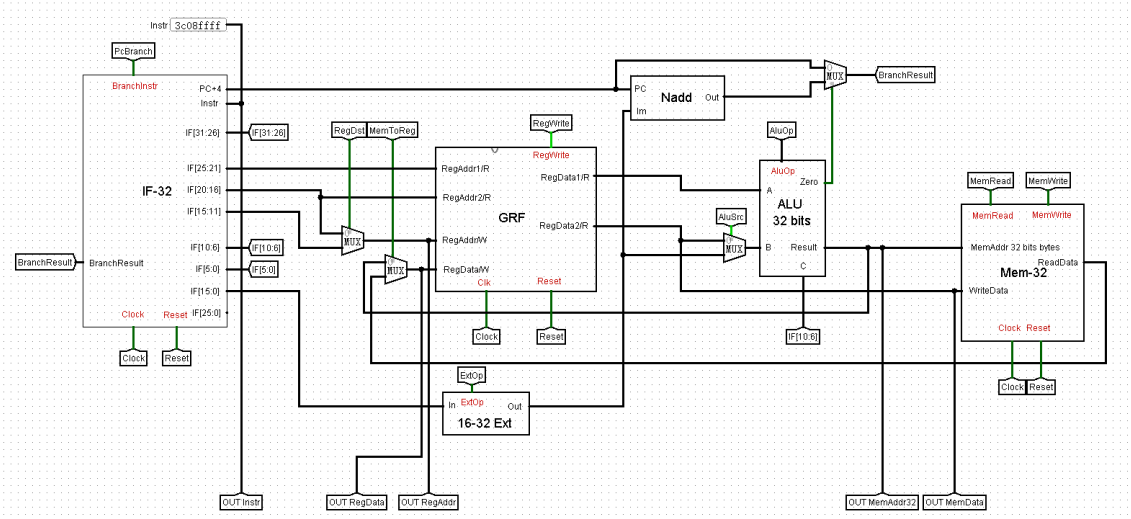

图 1 数据通路概览图

二、 模块规格（数据通路）

1、数据通路

表格 2 数据通路端口总表

	PC	IF	GRF				ALU			DM		Ex
Instr	PC	IF	RA1	RA2	WA	WD	A	B	C	Addr	WD	Ex-32
Addu	Pc+4	Pc	IF[25:21]	IF[20:16]	IF[15:11]	ALU	R1	R2				
Subu	Pc+4	Pc	IF[25:21]	IF[20:16]	IF[15:11]	ALU	R1	R2				
Sll	Pc+4	Pc		IF[20:16]	IF[15:11]	ALU		R2	IF[10:6]			
Lw	Pc+4	Pc	IF[25:21]		IF[20:16]	DM	R1	SE(IM)		ALU		IF[15:0]
Sw	Pc+4	Pc	IF[25:21]	IF[20:16]			R1	SE(IM)		ALU	R2	IF[15:0]
Ori	Pc+4	Pc	IF[25:21]		IF[20:16]	ALU	R1	ZE(IM)				IF[15:0]
Lui	Pc+4	Pc			IF[20:16]	ALU		ZE(IM)				IF[15:0]
Beq	Nadd|Pc4	Pc	IF[25:21]	IF[20:16]			R1	R2				

*SE : Sign-Extend; ZE: Zero-Extend

**Lui 指令操作与指令手册存在细微差异，为统一化预先进行了 32 位扩展。

***下划线端口：存在多路数据输入，需要使用选择器。

2、IFU（取指令单元）

a) 端口

表格 3 IFU 端口表

端口名称	类型	功能描述
Clock	In	控制信号，接受时钟信号
Reset	In	控制信号，接受异步复位信号
BranchInstr	In	控制信号，接受指令是否为跳转指令
BranchResult[31:0]	In	数据通路，跳转指令中接受 PC 更新值

PC+4[31:0]	Out	数据通路，输出 PC+4（32 位**内部编址需转换**）
Instr[31:0]	Out	数据通路，输出 32 位指令二进制码
IF[31:26]	Out	数据通路，Op
IF[25:21]	Out	数据通路，Rs
IF[20:16]	Out	数据通路，Rt
IF[15:11]	Out	数据通路，Rd
IF[10:6]	Out	数据通路，目前用于 sll 移位偏移量。
IF[5:0]	Out	数据通路，Function
IF[15:0]	Out	数据通路，立即数

b) 功能描述

- IFU 主要由 PC 和存放指令的 ROM 组成，用于取出指令和 PC 更新。
- ROM 规格为 32*32bits，Logisim 部件以 5 位二进制，**字编址**。
- PC 为 32 位二进制，起始地址为 0x00000000，**字节编址**，支持向字编址转换（除 4）和防指令地址溢出（控制 PC 使能）。

c) 注意事项

- ROM 和 RAM 部件的地址端口为字编址地址。为保证兼容性，该设计在顶层设计时用字节编址，而在次层具体部件设计时会进行字编址转换。

3、GRF（通用寄存器组）

a) 端口

表格 4 GRF 端口表

端口名称	类型	功能描述
Clock	In	控制信号，接受时钟信号
Reset	In	控制信号，接受异步复位信号
RegWrite	In	控制信号，接受寄存器写使能信号
RegAddr1/R[4:0]	In	数据通路，读，接受 Rs 寄存器地址
RegAddr2/R[4:0]	In	数据通路，读，接受 Rt 寄存器地址

RegAddr/W[4:0]	In	数据通路，写，接受被写入寄存器地址
RegData/W[31:0]	In	数据通路，写，接受被写入数据
RegData1/R[31:0]	Out	数据通路，读，输出 Rs 寄存器值
RegData2/R[31:0]	Out	数据通路，读，输出 Rt 寄存器值

b) 功能描述

- GRF 中共有 32 个寄存器，对应 MARS 中的 32 个通用寄存器。（注意：不包括 hi，lo，pc 寄存器。）
- 读：GRF 读功能时作为组合逻辑电路，根据输入地址信号，输出数据。
- 写：GRF 写功能作为时序逻辑电路，相应的 Addr, Data, RegWrite 应在时钟上升沿前做好准备。

c) 备注

- 此版本 GRF 读写功能的地址和数据端口独立，可实现同步读写操作。
- 0 号寄存器恒为 0 值，不可被改写。

4、ALU（逻辑运算单元）

a) 端口

表格 5 ALU 端口表

端口名称	类型	功能描述
AluOp[2:0]	In	控制信号，接受时钟信号
A[31:0]	In	数据通路，接受算术逻辑操作数 A
B[31:0]	In	数据通路，接受算术逻辑操作数 B
C[4:0]	In	数据通路，接受算术逻辑操作数 C（常于移位）
Zero	Out	数据通路，结果是否为 0（1 when result=0）
Result[31:0]	Out	数据通路，输出结果

b) 功能描述

- ALU 受 AluController 的控制信号控制，输出不同的算术逻辑结果：

AluOp	功能	适用指令
000	32 位左移运算 sll	sll, nop
001	32 位 + ，不带溢出检测	addu, lw, sw
010	32 位 - ，不带溢出监测	Subu, beq
011	32 位 |	ori
100	32 位 <<16	lui

5、DM（数据存储器）

a) 端口

表格 6 数据存储器端口表

端口名称	类型	功能描述
Clock	In	控制信号，接受时钟信号
Reset	In	控制信号，接受异步复位信号
MemWrite	In	控制信号，接受写使能信号
MemRead	In	控制信号，接受读使能信号
MemAddr[31:0]	In	数据通路，接受**读/写**操作地址（**内部编址需转换**）
WriteData[31:0]	In	数据通路，写，写入数据
ReadData[31:0]	Out	数据通路，读，输出数据

b) 描述

- 数据存储器使用 RAM 实现，容量为 32*32bits，RAM 地址为字编址。
- 读/写共用一个地址端口，同一时钟周期只能进行读/写的其中之一。
- 起始地址：0x00000000。

c) 注意事项

- DM 模块设计时并未直接将 RAM 置于顶层设计中，是由于其地址编码

方式和长度与顶层通路不符，需预先处理后接入。

6、Ext（位数扩展器）

表格 7 16-32 **位扩展器端口表**

端口名称	类型	功能描述
ExtOp	In	控制信号，控制扩展方式（0-zero，1-sign）
In[15:0]	In	数据通路，接收待扩展的 16 位数字。
Out[31:0]	Out	数据通路，输出扩展后的 32 位数字。

7、Nadd（分支指令寻址器）

端口名称	类型	功能描述
PC+4[31:0]	In	数据通路，不发生跳转时下条指令地址
Offset[31:0]	In	数据通路，偏移量（**正负，字编址**）
Out[31:0]	Out	数据通路，PC + 4 + Offset << 2

三、 模块规格（控制电路）

本设计电路中，和教程略有不同，区别主要体现于对 ALU 运算控制使用了独立的控制元件。控制电路模块由 Controller 和 ALUController 两部分组成，模块组成均为组合逻辑。

1、Controller 主控单元

a) 端口

表格 8 主控单元端口功能表

端口名称	类型	功能描述
Instr[31:26]	In	数据通路，指令的 Op 段
RegWrite	Out	控制信号，GRF 写使能信号 （1–允许，0-不允许）
MemRead	Out	控制信号，Mem 读使能信号（1-允许，0-不允许）
MemWrite	Out	控制信号，Mem 写使能信号（1-允许，0-不允许）
AluSrc	Out	控制信号，ALU-B 端口选择器信号（1-扩展器，0-Reg Read2）
PcBranch	Out	控制信号，IF 接受外部寻址计算信号（1-允许，0-不允许）
RegDst	Out	控制信号，GRF-写入端 Addr 选择信号（1-Rd 段，0-Rt 段）
MemToReg	Out	控制信号，GRF 接受 Mem 输出选择信号（1-Mem，0-Alu）
ExtOp	Out	控制信号，16-32 位扩展类型（1-符号扩展，0-无符号扩展）
AluCtrl[2:0]	Out	控制信号，ALUController 驱动信号

b) 信号真值表

表格 9 信号真值表

Instr	Addu/Subu/Sll(nop)	Lw	Sw	Ori	Lui	Beq
Op	000000	100011	101011	001101	001111	000100
RegWrite	1	1	0	1	1	0
MemRead	0	1	0	0	0	0
MemWrite	0	0	1	0	0	0

AluSrc	0	1	1	1	1	0
PcBranch	0	0	0	0	0	1
RegDst	1	0	x	0	0	x
MemToReg	0	1	x	0	0	x
ExtOp	x	1	1	0	x	1
AluCtrl[2:0]	000	001	001	010	011	100

*：请注意，ALUCtrl 在此为控制 ALUController 的控制信号，而不是控制 ALU 的 ALUOp 信号。

2、ALUController ALU 算术逻辑运算控制单元

ALUController 是 ALU 运算单元的直接上级控制单元，通过向 ALU 输出三位二进制数，驱动 ALU 输出对应的 32 位结果和 1 位结果。

a) 端口

表格 10 ALUController 端口功能表

端口名称	类型	功能描述
ALUCtrl[2:0]	In	控制信号，主控单元对本单元驱动信号
Function[5:0]	In	数据通路，指令 function 段
AluOp	Out	控制信号，ALU 运算类型选择信号

b) AluOp 信号真值表

表格 11 AluOp 信号真值表

F C	000	001	010	011	100
xxxxxx		001(add)	010(sub)	011(or)	100(shift 16)
100001	001(add)				
100011	010(sub)				
000000	000(sll)				

四、 CPU 功能测试

1、 功能测试原则

- 所测试的指令不能够超出 CPU 支持的范围。(谨防同指令标识的拓展指令。)
- 测试的首要目标：全面（例如：正负性、边界数据、边界存储等)。
- 测试步骤：控制信号 -> 单指令 -> 多指令，寄存器、主存 -> 综合

2、 测试策略

1、 控制信号正确性检查：独立检查每条指令的控制信号是否符合设计。

2、 取值防溢出检查：手动运行时钟，检查 ROM 取指是否溢出。

3、 单指令正确性检查：

a) Addu, subu: 正正（溢出/不溢)，负负（溢出/不溢)，正负，负正。

b) Lui，ori：负数的构造。

c) Sw，lw：非零数边界存储，正负偏移存储。

d) Beq：成立与不成立跳转，下跳与上跳。

e) Sll 检查：Nop 功能。

4、 综合性程序检查：带条件的存值+带条件的取值。

3、 测试实例

a) **控制信号正确性检查**

使用下列代码检查各指令运行时，控制信号值是否符合期望，控制信号期望输出于第三节以表格给出，在此不再赘述。

```
addu $t1, $t2, $t3
subu $t1, $t2, $t3
lw $t1, 12($t2)
sw $t1, 12($t2)
ori $t1, $t2, 0xffff
lui $t1, 0xf
sll $t1, $t2, 2
back:
nop
beq $t1, $t2,back
```

代码 1 **控制信号正确性检查代码**

b) **取指防溢出检查**

使用 reset 信号回到初始状态，Pc 值为 0x00000000，手动模拟时钟产生 32 次及以上的上升沿，观察 Pc 值变化情况。

期望状态：Pc 每次上升沿先+4，而后到达 0x1f 保持恒定，直至 reset 信号。

c) **单指令正确性检查**

单指令正确性检查首先是 lui 和 ori，因为这两条指令才能构造非零数。而后，利用构造完成的各种数值，对指令 addu、subu、lw 和 sw 进行测试。

测试代码与期望结果如下：

```
# 单指令运算正确性检查                    # 检查内容与期望结果
# Lui, Ori
lui $s0, 0x7ff1                    # 功能检查：$16 = 0x7ff10000
ori $s1, $zero, 0xf001                 # 功能检查：$17 = 0x0000f001
lui $s2, 0xffff
ori $s3, $s2, 0xfffc                   # 负数构造：$19 = -4
# addu, subu
addu $t0, $s0, $s1                 # 不溢出相加：$8 = 0x7ff1f001
addu $t1, $s0, $s0                 # 溢出相加：$9 = 0xffe20000
addu $t2, $s3, $s3                 # 负负相加 $10 = -8
addu $t3, $s1, $t2                 # 正负相加 $11 = 61433
subu $t4, $s1, $t3                 # 正正相减 $12 = 8
subu $t5, $t3, $s1                 # 正正相减 $13 = -8
subu $t6, $s3, $t5                 # 负负相减 $14 = 4
lui $s4, 0x8000
subu $t7, $s4, $t4                 # 负正相减(溢出) $15 = 0x7fffffff8
# sw, lw
ori $t8, 0
sw $t0, 0($t8)                     # sw 下边界测试：mem[0] = 0x7ff1f001
sw $t1, 16($t8)                    # sw 偏移量测试：mem[4] = 0xffe20000
ori $t9, 124
sw $t4, 0($t9)                     # sw 上边界测试：mem[31] = 8
sw $t5, -16($t9)                   # sw 负偏移测试：mem[27] = -8
ori $t8, 13
lw $s4, 3($t8)                     # 和为 4 倍偏测试：$20 = 0xffe20000
```

代码 2 **单指令真确性检查代码** 1

最后针对 beq 指令进行检测

```
# beq                                  # 检测目的与预期结果
ori $t0, 1
ori $t1, 2
ori $t2, 3
ori $t4, 1

addu $s0, $s1, $s2                     # 非跳转语句选择无效检测：pc = pc + 4
(not pc + 4 + 0x8021)
beq $t0, $t1, label2                   # 不跳转检测：pc = pc+4
nop
label1:
    lui $s0, 0xf000
    beq $t0, $t4, label4               # 正跳转检测：pc = pc+4+8
label2:
    lui $s1, 0x0f00
label3:
    lui $s2, 0x00f0
label4:
    lui $s3, 0x000f
    addu $t0, $t0, $t4
    beq $t0, $t1, label3               # 逆跳转检测：pc = pc+4-16

   # 最终期望输出：$16 = 0xf000, $17 = 0, $18 = 0x00f0, $19 =
0x000f, $8 = 3
```

代码 3 **单指令正确性检查代码** 2

d) 综合性程序检查：条件运算存值+条件取值。

```
# 条件运算存值+条件取值 综合程序设计
lui $t0, 0xffff
ori $t0, $t0, 0xffff                # t0 = -1
ori $t1, $t1, 15
ori $t2, $t2, 16
ori $t3, $t3, 116
ori $t4, $t4, 100
sll $t5, $t0, 2

beq $t1, $t2, sw_if_end
nop                 # 带条件地存值
sw_if_begin:
    sw $t0, 0($t3)
    sw $t1, 4($t3)
    sw $t2, 8($t3)
sw_if_end:
    sw $t3, -4($t3)
    sw $t4, -8($t3)
    sw $t5, -12($t3)

addu $t6, $t4, $t2          # 带条件的取值
beq $t6, $t3, lw_if_end
nop
lw_if_begin:
    lw $s0, 16($t4)
    lw $s1, 20($t4)
    lw $s2, 24($t4)
lw_if_end:
    lw $s3, 12($t4)
    lw $s4, 8($t4)
    lw $s5, 4($t4)
# 期望输出
# Memory: m[29] = -1, m[30] = 15, m[31] = 16, m[28] = 116, m[27] =
100, m[26] = -4, 其余内存段为 0
    # Register: $8 = -1, $9 = 15, $10 = 16, $11 = 116, $12 = 100,
$13 = -4, $14 = 116, $19 = 116 , $20 = 100, $21 = -4, 其余寄存器为 0
```

代码 4 综合性功能检测代码

五、 本章思考题

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

优点：计数器生产成本更低；实际运行时延时更低。

缺点：若指令字节地址超过 2^30-1 则无法表示；30 位不常见，相关其他部件需要进行微调变化。

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。

ROM 不合理 :实际情况中系统程序为 ROM 型存储, 但用户程序区应用 RAM, 因为用户操作目的不同，其中机器码应该会变化。因此在实际设计时应该使用 ROM 和 RAM 芯片阵列混合存储。

RAM 和 GRF 的设计我认为是合理的，这两个部分较真实地还原了 CPU 中的结构——32 个寄存器+缓存区。

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。

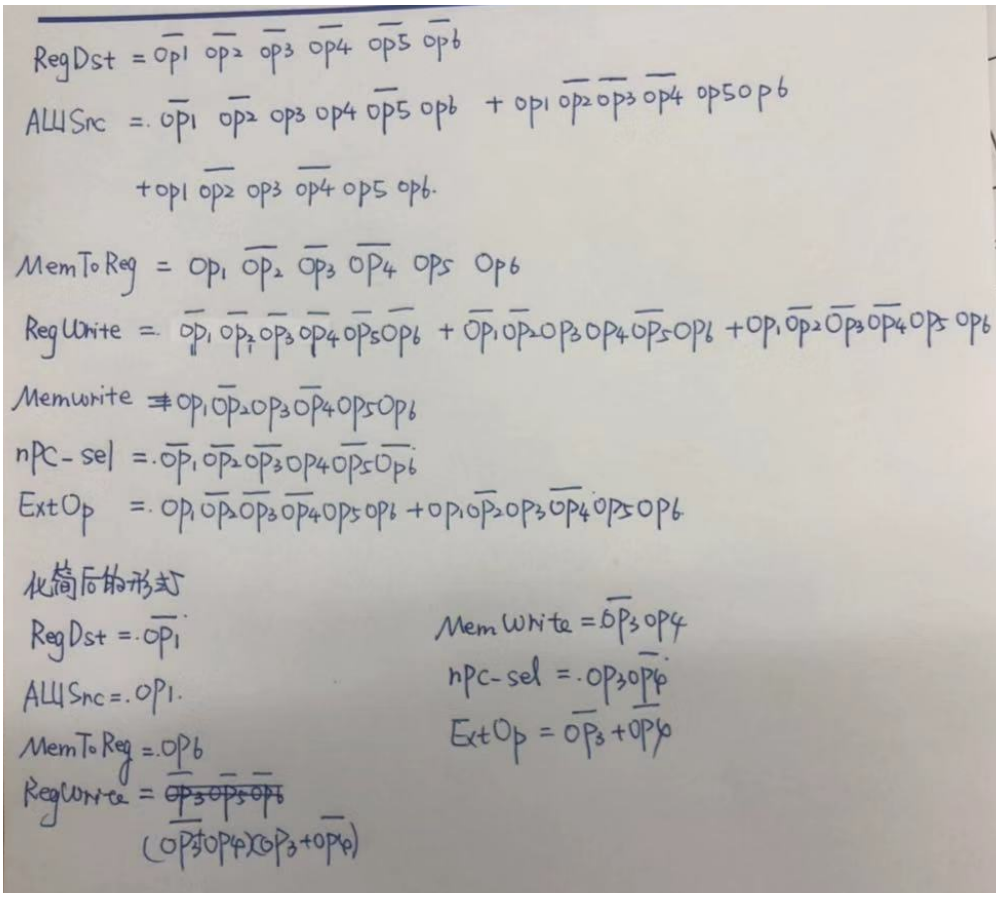

图 2 思考题图-逻辑表达式及其化简

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。

答：首先，在 CPU 中其实有组合逻辑和时序逻辑（写寄存器、写主存）两部分组成，而一条指令之所以能够产生影响是因为其执行了时序逻辑的写入（组合逻辑没有记忆）。

分析 nop 这条指令，Op 部分对应 R 型指令，由 Controller 编码可知 R 指令不会对 Mem 改写，因此只用分析 GRF 是否会被改写，但是由于其 Rd 对应的是 $zero，其值不会发生变化。综上，nop 不会产生改写，因而可以等价于没操作的语句=控制信号无用=不需要加到真值表。

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。

答：在确定了 DM 段的起始地址后（例如 0x00003000~0x00003ffff）我们便可以使用一个比较器判断当前访问地址是否在此段内，而后置片选信号为有效，那么这个被选择的 DM 则无需进行手工偏移。

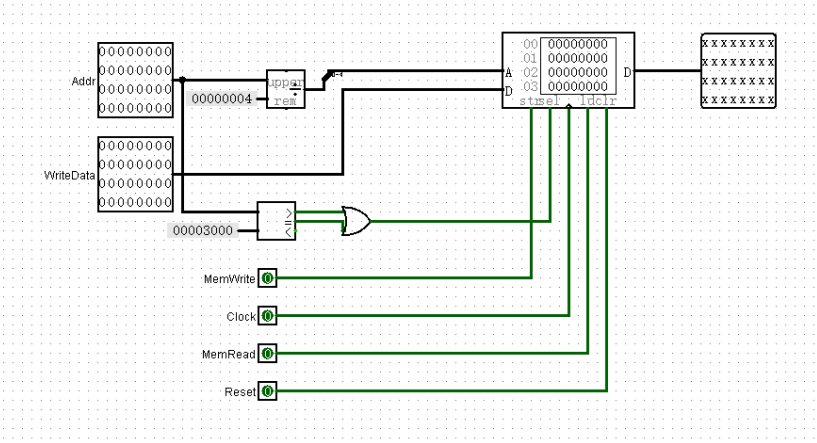

图 3 DM **片选电路模拟**

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。

在超大规模的 asic 中，进行仿真是非常耗时的，因此通常会采用等价对比来保证正确性（等级对比即指形式验证）。在形式验证时，对于参考设计和实际设计，对逻辑锥输入相同的激励信号，然后比较点的数据一致性。在比较时，形式验证会对所有可能的情况进行验证，而不是一部分情况。

- 优点：遍历了测验电路元件的所有情况，便于尽早地发现问题。
- 优点：形式验证速度快，耗时段，成本低，适用于仿真前的验证。
- 缺点：形式验证对电路实际运行时的时序关系等无法判断，容易遗留潜

在问题。

- 缺点 :形式验证不是仿真, 因为不能确定电路的时延、功耗等性能问题。

六、 有关 CPU 扩展的说明

本单周期 CPU 支持一定的扩展，允许使用 Logisim 软件对线路进行调整，以完成更多指令功能，用户在自行拓展时，应主要遵循以下几个步骤。

- 分析指令，手工绘制新增指令的数据通路。(考虑是否新增数据元件)
- 将新增指令加入数据通路端口表中。
- 先完善数据通路。先微观：以每个功能子电路为单位新增和修改功能；后宏观：备份主电路，在副本中根据端口表修改通路。
- 后完善控制电路 :根据分析结果在 Controller 和 ALUController 中新增或修改控制信号。
- 测试阶段：控制信号测试-单指令多情况测试-综合测试。