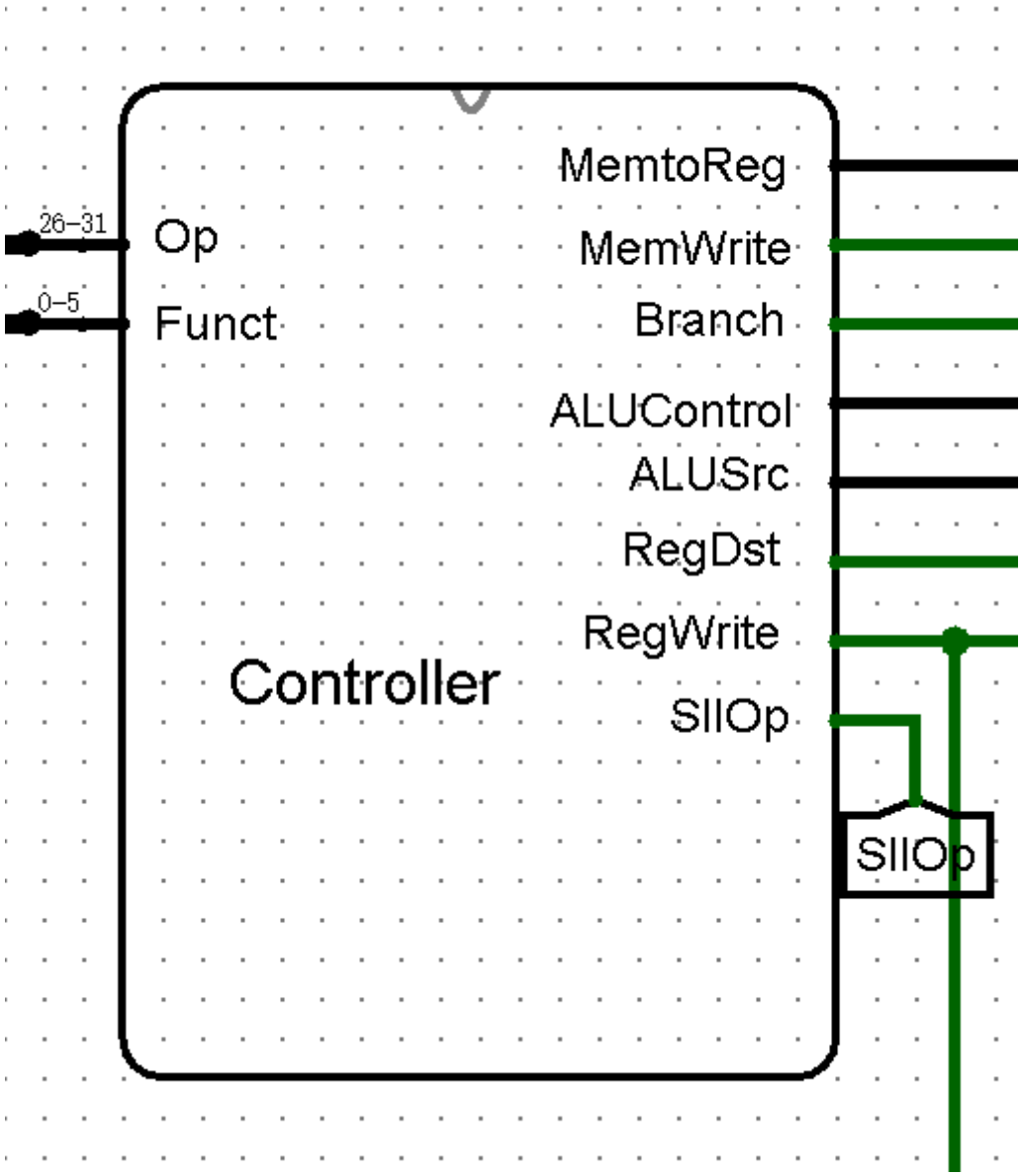
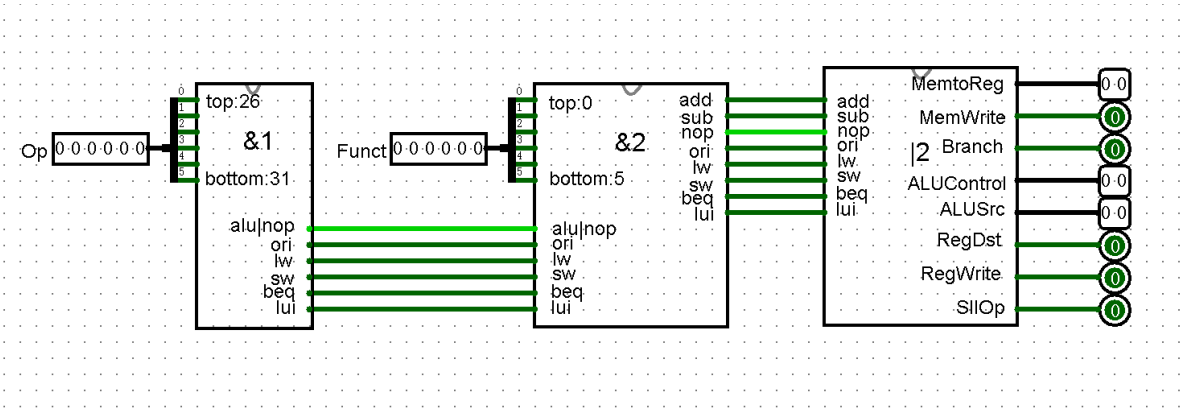
	PC	IM	Adder	Nadder	RF	ALU	Zeroext	Signext	DM	Sll	Comparator								
A	B	A	B	RA1	RA2	WA	WD	Op	A	B	DA	DD	Num	A					
add	Adder	PC	PC	4	\	[6:10]	[11:15]	[16:20]	ALU	+	RD1	RD2	\	\	\	\	\		
sub	Adder	PC	PC	4	\	[6:10]	[11:15]	[16:20]	ALU	-	RD1	RD2	\	\	\	\	\		
ori	Adder	PC	PC	4	\	[6:10]	\	[11:15]	ALU	|	RD1	Zeronext	[16:31]	\	\	\	\		
lw	Adder	PC	PC	4	\	[6:10]	\	[11:15]	DM	+	RD1	Signext	\	[16:31]	ALU	\	\	\	
sw	Adder	PC	PC	4	\	[6:10]	[11:15]	\	\	+	RD1	Signext	\	[16:31]	ALU	RD2	\	\	
beq	Adder|Nadder	PC	PC	4	Adder	Signext	[6:10]	[11:15]	\	\	-	RD1	RD2	\	Sll	\	2	[16:31]	ALU
lui	Adder	PC	PC	4	\	\	\	[11:15]	Sll	\	\	\	\	16	[16:31]	\			
nop	Adder	PC	PC	4	\	\	\	\	\	\	\	\							

H1 CPU

H2 Controller Signal

	add	sub	nop	ori	lw	sw	beq	lui	lb	sb	jal	jr
MemtoReg	0	0	\	0	1	\	\	2	3	\	4	
MemWrite	0	0	0	0	0	1	0	0	0	1	1	
Branch	0	0	0	0	0	0	1	0	0	0	2	
ALUControl	0	1	\	2	0	0	1	\	0	0	\	
ALUSrc	0	0	\	2	1	1	0	\	1	1	\	
RegDst	1	1	\	0	0	\	\	0	0	\	2	
RegWrite	1	1	0	1	1	0	0	1	1	0	1	
SllOp	\	\	\	\	\	\	0	1	\	\	\	
WDOp	\	\	\	\	\	0	\	\	\	1	\	





H3 MemtoReg

控制向寄存器RA3中写入数据的来源

MemtoReg	source
0	ALU

MemtoReg	source
1	DM(RDW)
2	SlllImm
3	DM(RDB)
4	PC+4

H3 MemWrite

控制主存写使能信号

MemWrite	write?
0	No
1	Yes

H3 Branch

控制新的PC生成

Branch	newPC
0	PC+4
1	PC+4+offset
2	jalPC

H3 ALUControl

控制ALU内运算符号

SelOp	ALUResult
0	SrcA + SrcB
1	SrcA - SrcB
2	SrcA | SrcB

ALUSrc

控制ALU中的SrcB的来源

ALUSrc	src
0	GRF(RD2)
1	SignImm
2	ZeroImm

RegDst

控制寄存器RA3在代码中的位置

RegDst	pos
0	[20:16]
1	[15:11]
2	31

RegWrite

控制GRF写入信号

RegWrite	write?
0	No
1	Yes

SllOp

控制Sll模块输出的数据

0为拼接两位，并进行符号拓展

1为零拓展，并左移16位

SllOp	SllImm
0	Signext(Imm+2[0])
1	Zeroext(Imm+16[0])

WDOp

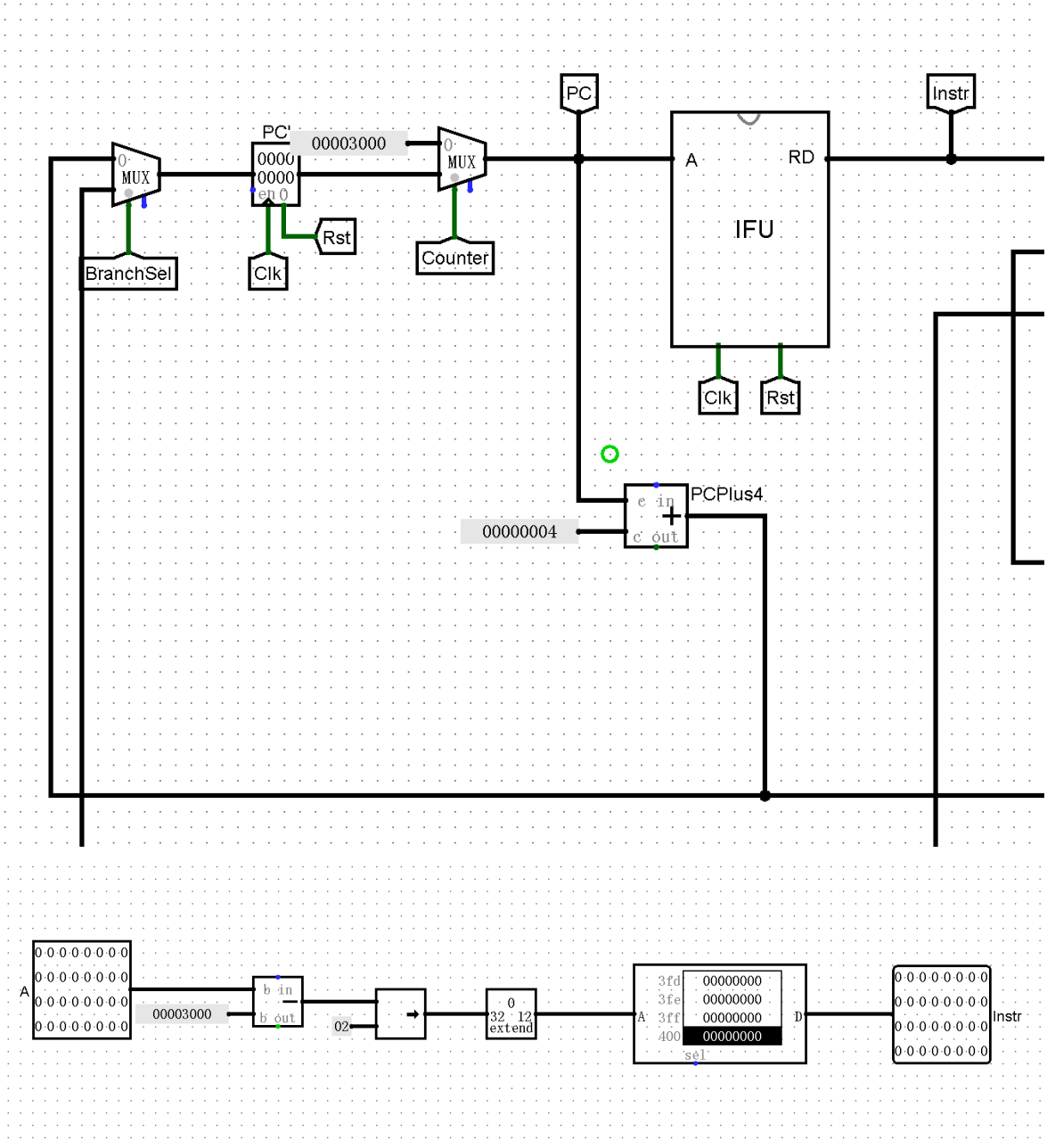
控制DM中写入的数据长度

WDOp	length
0	32(sw)
1	8(sb)

IFU（取指令单元）

- 内部包括 PC（程序计数器）、IM（指令存储器）及相关逻辑。
- PC 用寄存器实现，具有**异步复位**功能，复位值为起始地址。
- **起始地址：0x00003000。**
- 地址范围：0x00003000 ~ 0x00006FFF。
- IM 用 ROM 实现，容量为 4096 × 32bit。
- ROM 内部的起始地址是从 0 开始的，即 ROM 的 0 位置存储的是 PC 为 0x00003000 的指令，每条指令是一个 32bit 常数。
- ROM 实际地址宽度仅需 12 位

每次取指时，将地址为(PC-0x00003000)>>2的指令取出



H2 GRF（通用寄存器组，也称为寄存器文件、寄存器堆）

- 用具有写使能的寄存器实现，寄存器总数为 32 个，应具有**异步复位**功能。
- **0 号寄存器**的值始终保持为 0。其他寄存器**初始值（复位后）均为 0**，无需专门设置。

A1:第一个寄存器

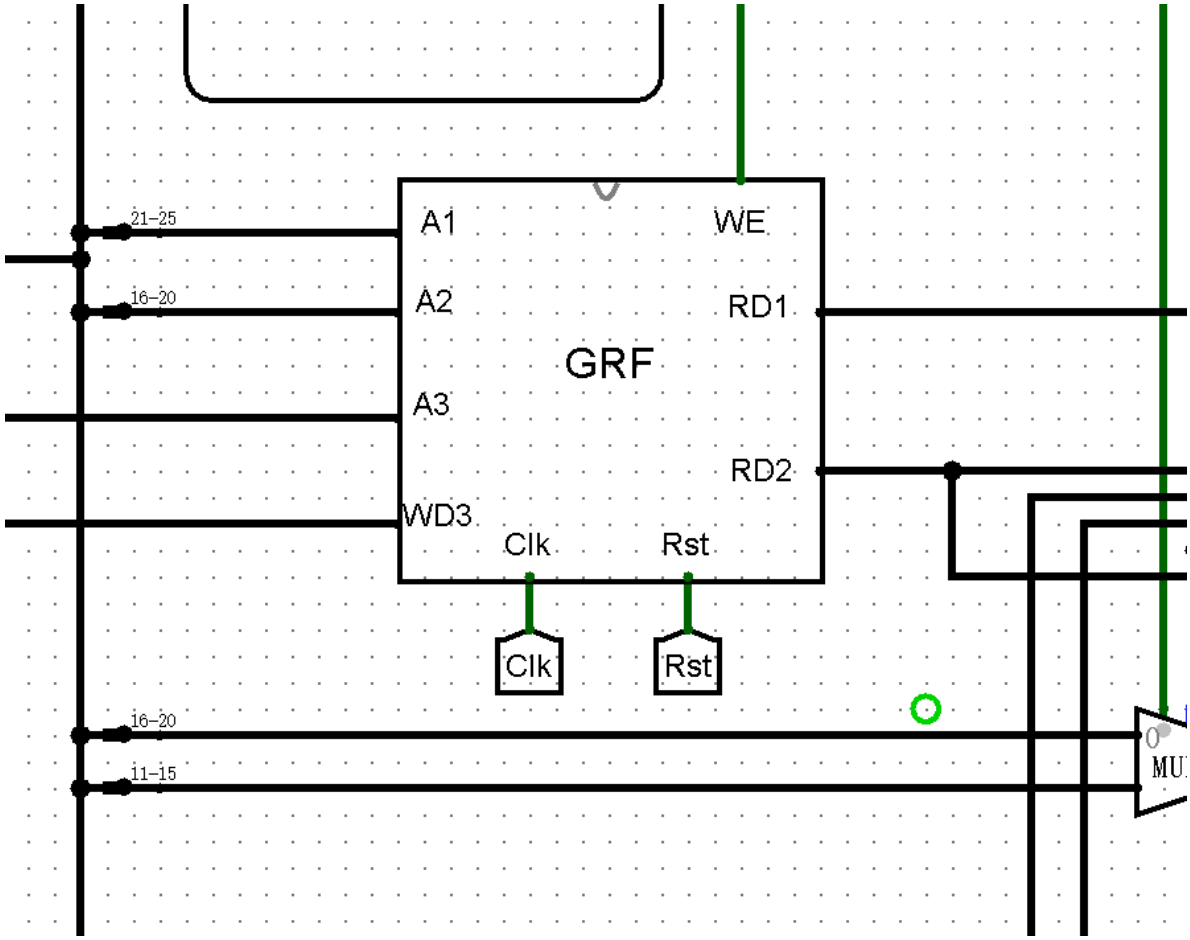
A2:第二个寄存器

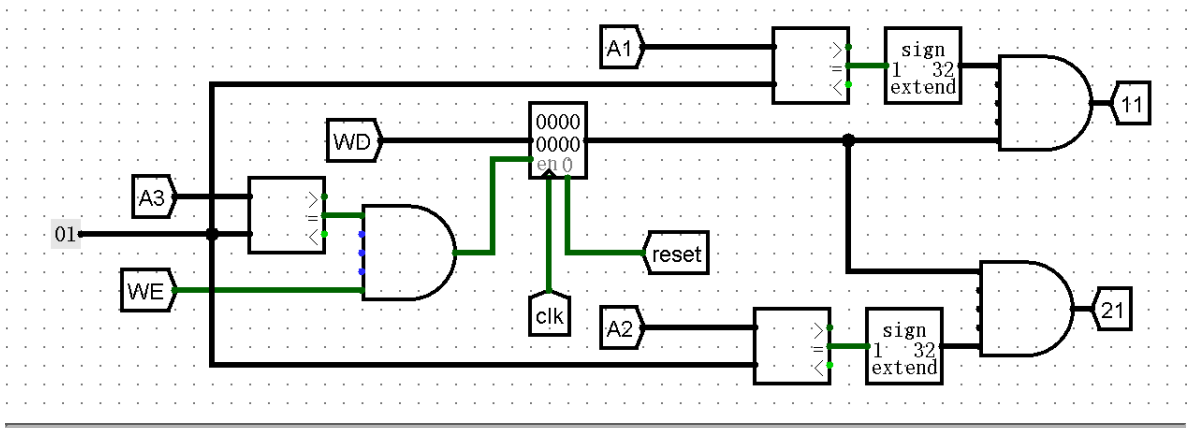
A3:第三个寄存器

RD1:GRF[A1]

RD2:GRF[A2]

WE:寄存器写使能信号





H2 ALU（算术逻辑单元）

- 提供 32 位加、减、或运算及大小比较功能。
- 加减法按无符号处理（不考虑溢出）。

SelOp:运算符号

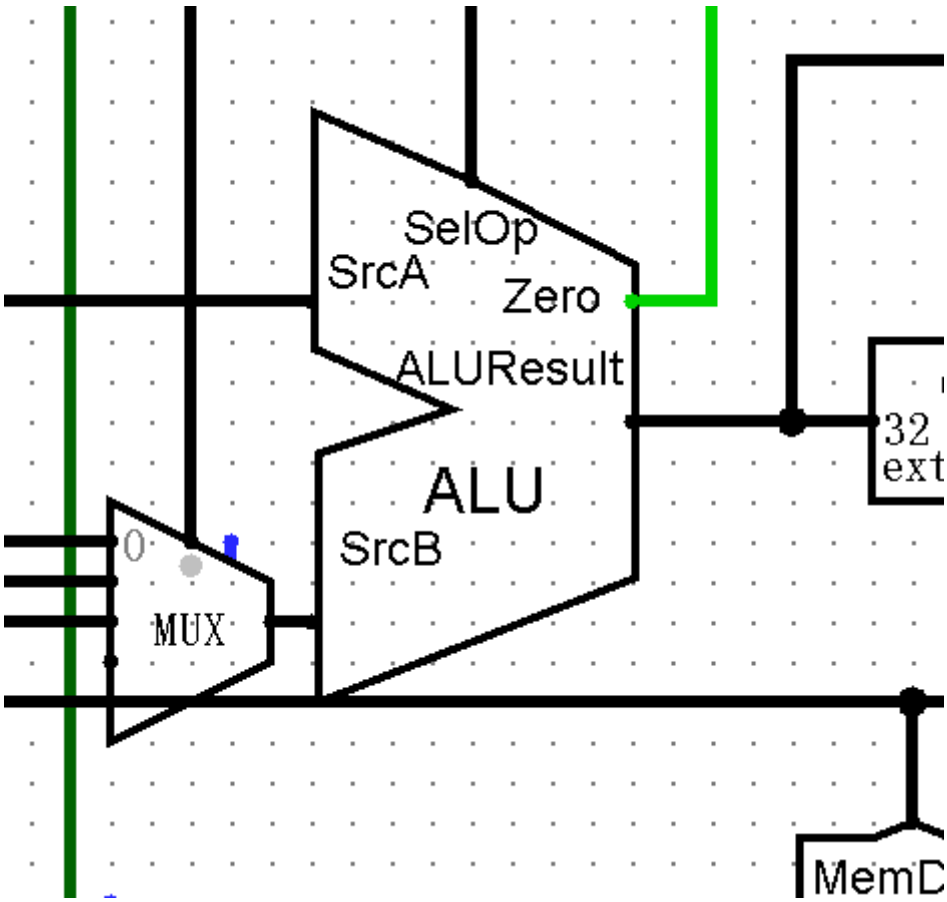
- 0 +
- 1 -
- 2 |

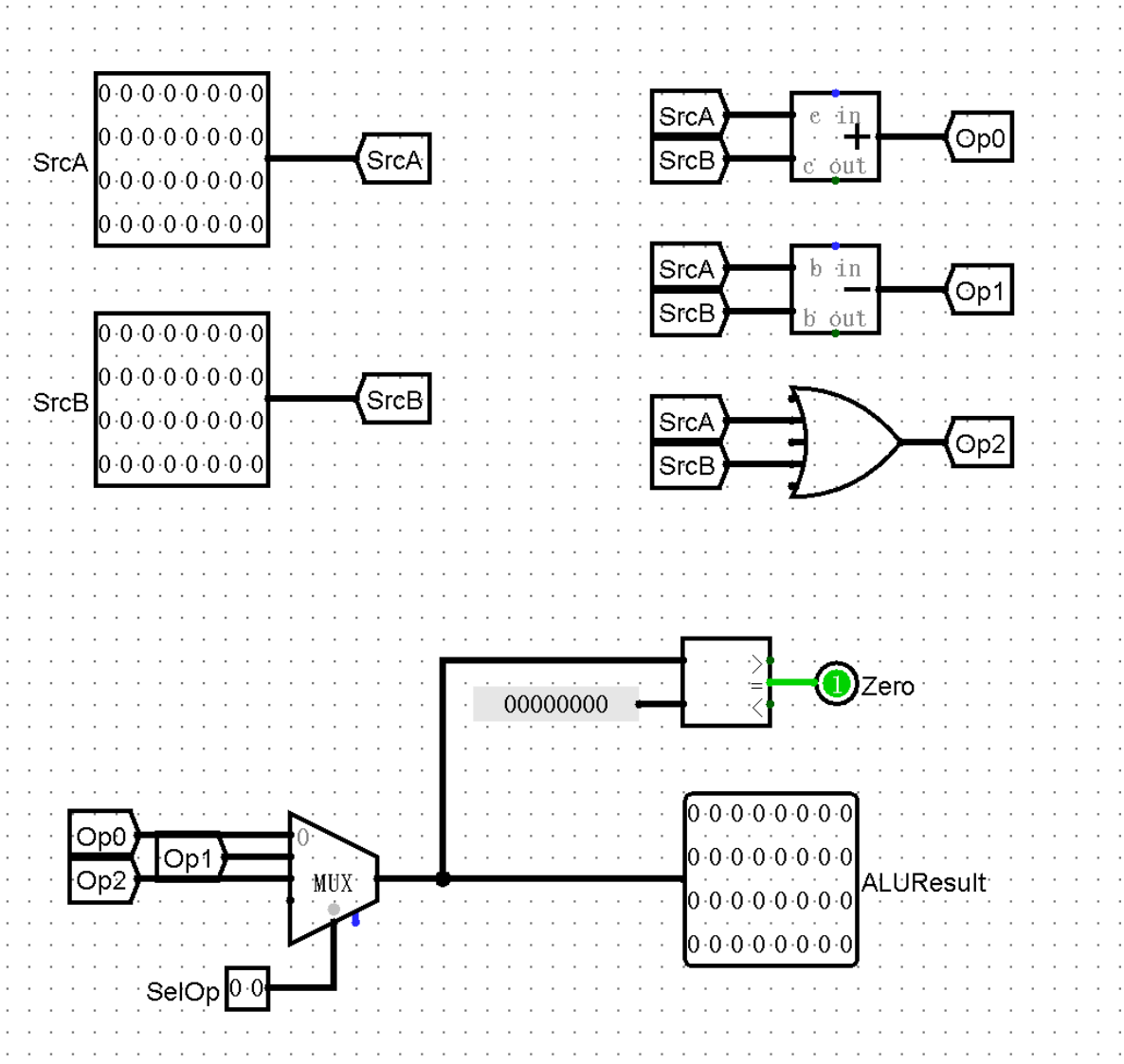
SrcA:第一个运算数

SrcB:第二个运算数

Zero:结果是否为0

ALUResult:运算结果





DM（数据存储器）

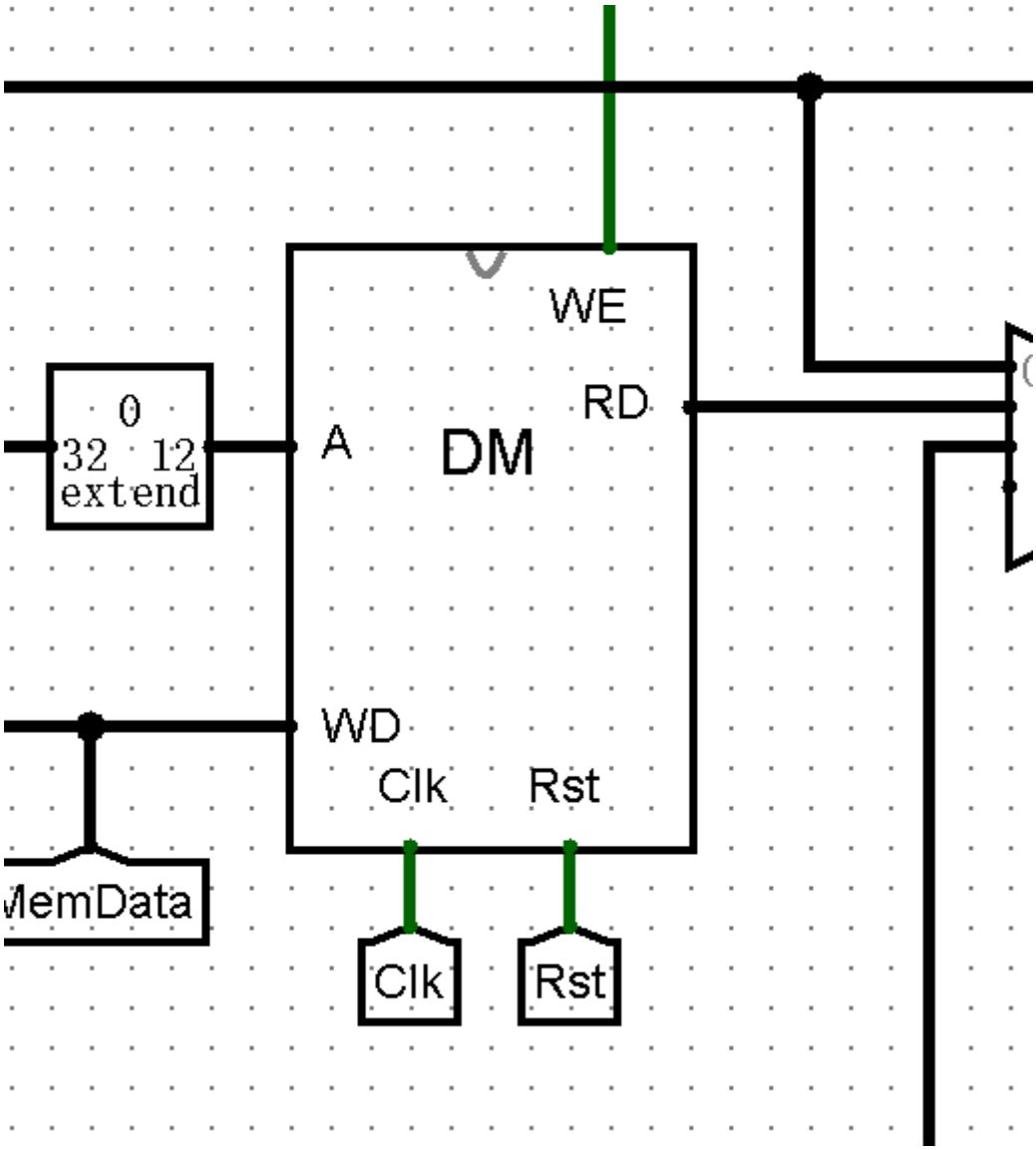
- 使用 RAM 实现，容量为 3072 × 32bit，具有**异步复位**功能，复位值为 0x00000000。
- **起始地址：0x00000000。**
- 地址范围：0x00000000 ~ 0x00002FFF。
- RAM 应使用双端口模式，即设置 RAM 的 **Data Interface** 属性为 **Separate load and store ports。**

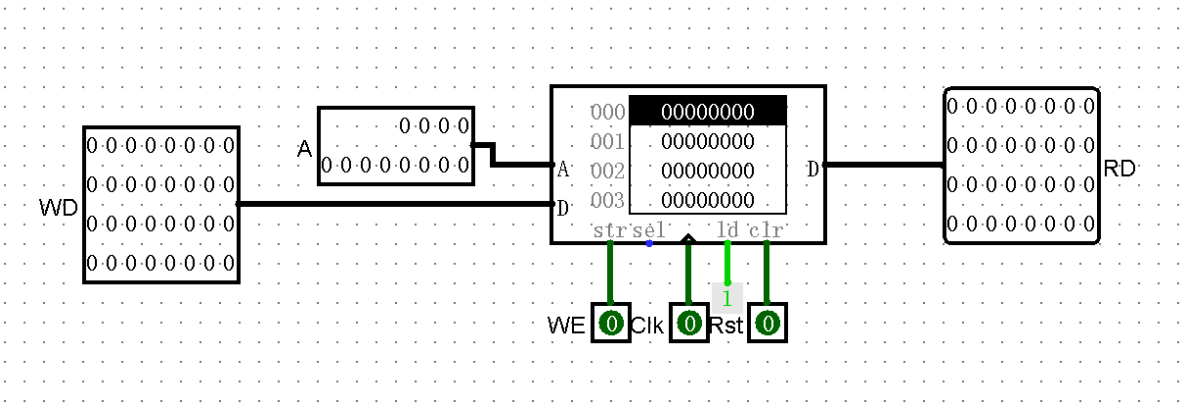
A:读或写地址

WD:写入数据

WE:写使能信号

RD:DM[A]





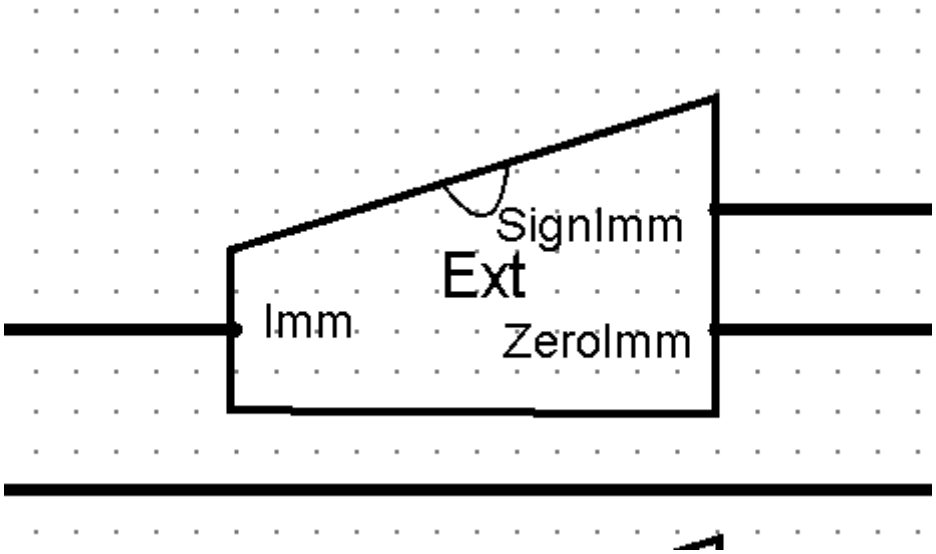
H2 EXT（扩展单元）

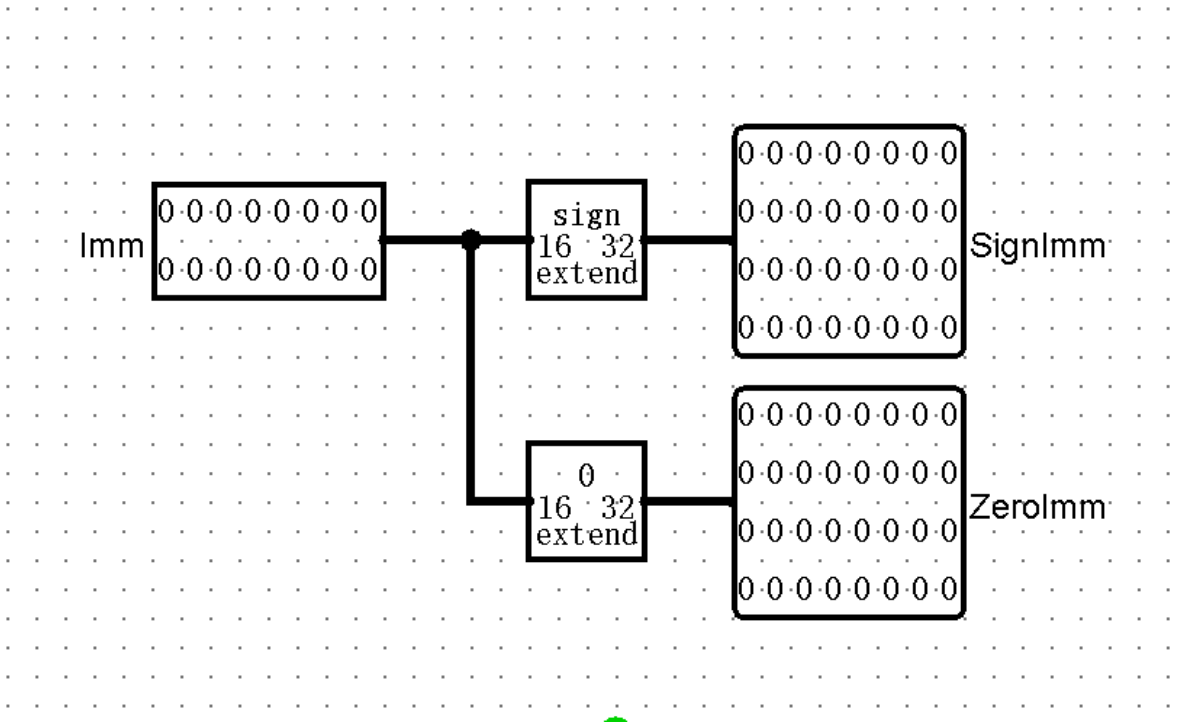
- 可以使用 Logisim 内置的 Bit Extender。

Imm:待进行拓展的立即数

SignImm:符号拓展的立即数

ZeroImm:零拓展的立即数





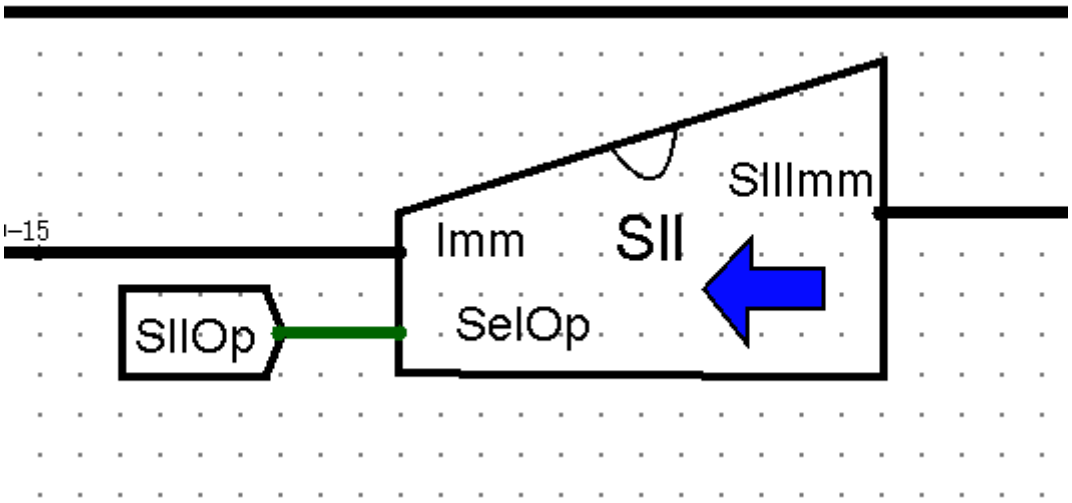
Sll（移位单元）

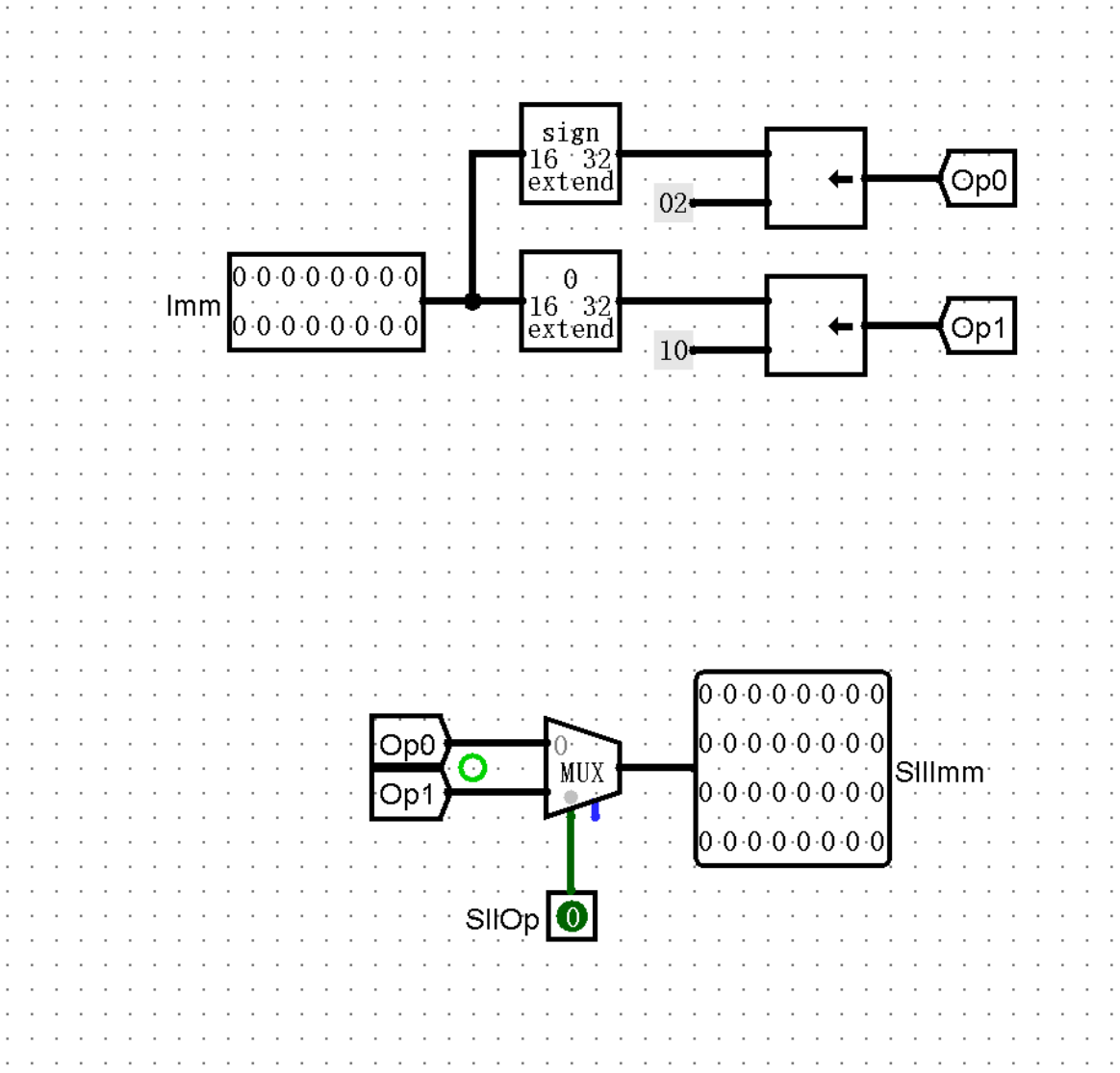
Imm:待移位立即数（16位）

SelOp:选择进行移位的符号

- 0：拼接两位，并进行符号拓展
- 1：零拓展，并左移16位

SllImm:





思考题：

1. 上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能。

   状态存储：DM,GRF

状态转移：IFU,NPC,EXT,ALU,Control

2. 现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用 Register，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中内容为程序执行前填入，在程序执行过程中不更改，所以为ROM；DM中内容需要进行读写操作，且内容较多，使用RAM；GRF只有32个需要读写的模块，使用使用Register

3. 在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路。

   见上的SII等

4. 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？

   因为nop什么事情都不做，只是保证信号稳定而已

5. 阅读 Pre 的 “MIPS 指令集及汇编语言” 一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处。

   所有指令都得到了测试，但不能覆盖所有情况，beq的跳转范围也较小