

微程序版 CPU 设计报告

姓名：张绍磊

学号：2016211392

班级：2016211310

实验环境：Proteus professional 8.6

小组成员：张绍磊 刘佳玮 陈博

任务描述：

1. 以小组为单位，自主设计微程序版 CPU（分模块、分层次）。建立清晰的计算机整机概念和内部机制，对计算机的基本组成、部件的设计、部件间的连接、微程序控制器的设计、微指令和微程序的编制与调试等过程有更深入的了解，加深对理论课程的理解。

2. 熟练掌握 proteus 仿真平台，学会创建元件、使用父子图模式以及常用的测试方法，实现整机调试。

顶层设计：

1. 本小组的微程序版 CPU 设计分为三个层次：

1. 实现 16 位简单 CPU 的基本功能：运算器+存储器（64K×16 位）
2. 实现微程序版模型机：CPU+时序+微程序控制器（6 条基本指令）
3. 实现附加功能：I/O、堆栈、单级中断等（8 条附加指令）

2. 在功能上，本小组进行了如下划分：

1. 存储逻辑：64K×16 位的主存由 ROM、RAM 构成，若干寄存器协调各个功能模块。
2. 运算逻辑：16 位 ALU，支持组间先行进位。
3. 控制逻辑：实现 14 条指令（16 位）和 30 条微指令（38 位）的指令系统。
4. 时序逻辑：由 4 个 T 脉冲构成 1 个 CPU 周期。
5. 中断逻辑：由中断触发电路随时触发中断，由 PSW、IAR 和堆栈区进行数据保护。

3. 小组特色：

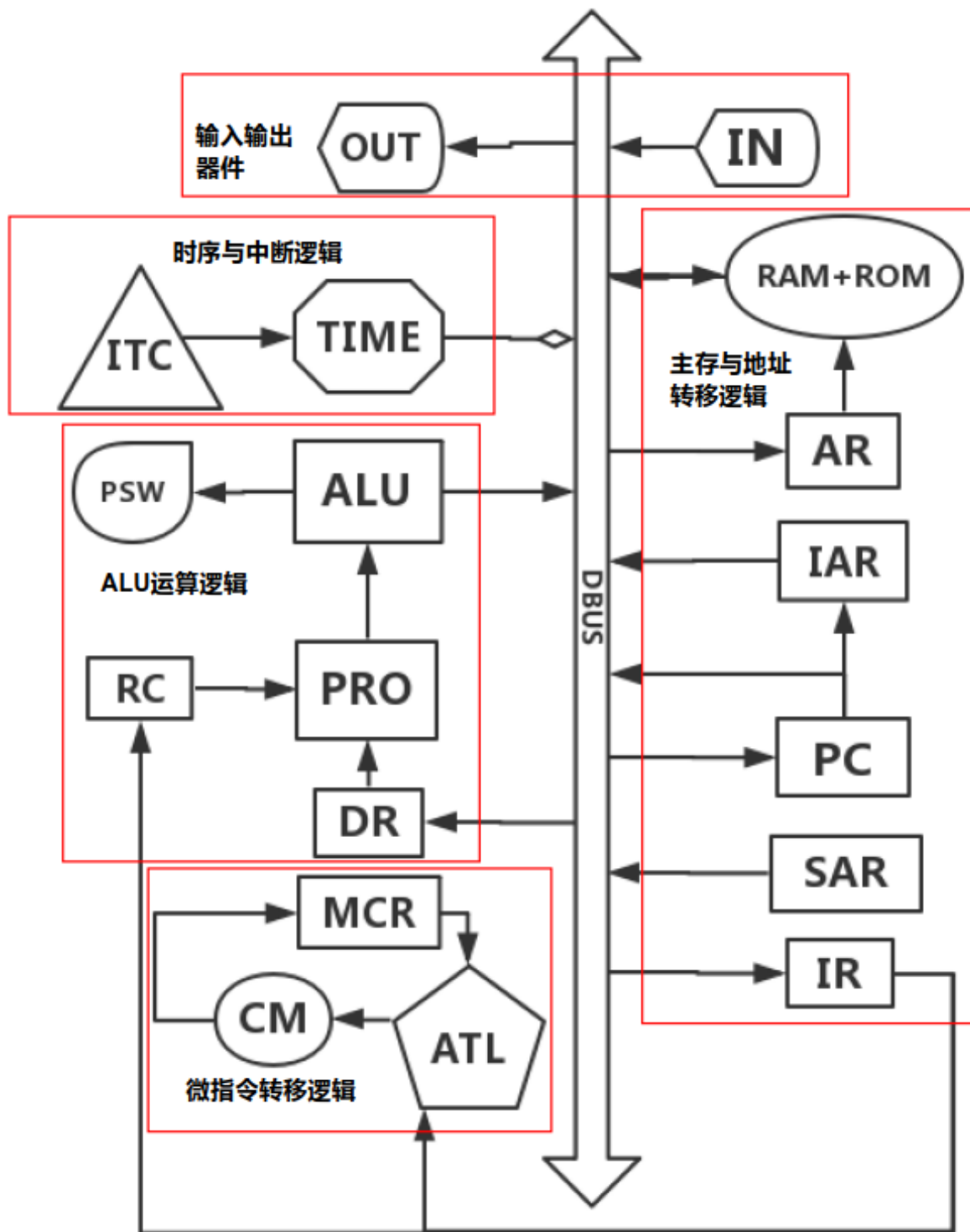
1. 父子图划分清晰，层次模块完整，功能划分合理明显，单总线便于观察。
2. 显示 PC、IR、AC、通用寄存器 R1~R3、当前微地址和后继微地址，支持单步调试和连续执行，便于观察、记录和分析实验数据。
3. 支持在中断期间对其余中断请求的屏蔽，对主程序状态进行完整保存，且由外部输入中断地址，可实现同一次运行中多个不同中断子程序的跳转，灵活度高。

4. 存储逻辑：

memory		
ROM	0000-2FFF	主程序
	3000-3FFF	中断子程序
	4000-7FFF	初始数据
RAM	8000-CFFF	主程序写入的数据
	D000-DFFF	中断写入的数据
	E000-FFFF	栈
	FFFE-FFFF	I/O

底层实现

1. 原理图：



说明：本小组 CPU 采用单总线+微程序控制，分为中断和时序逻辑、ALU 运算逻辑、微指令转移逻辑、主存和地址转移逻辑、输入输出器件 5 大部分，均采用父子图模式，I/O 交互性好，支持.hex 文件写入和运行时写入，配合 IN 指令和 OUT 指令随时观察。

2. 分工：

分工一览表		
姓名	自制元件	工程设计
陈博	DR,AR,PC,SAR,PSW,ATL	12条微指令设计，时序分配，地址转移逻辑设计，程序正确性调试，栈存在逻辑设计，数据流动情况分析，控制开关复用模式设计。
刘佳玮	IR,IAR,MCR,ITC,TIME,RC	18条微指令设计，中断电路设计，测试程序设计，程序正确性调试，时序产生逻辑设计，输入输出逻辑设计，寄存器控制逻辑设计。
张绍磊	RAM+ROM,CM,ALU,PRO	指令格式设计，微指令代码编写，ROM文件写入，测试程序设计，程序正确性调试，中间信号显示设计，微指令译码逻辑设计。

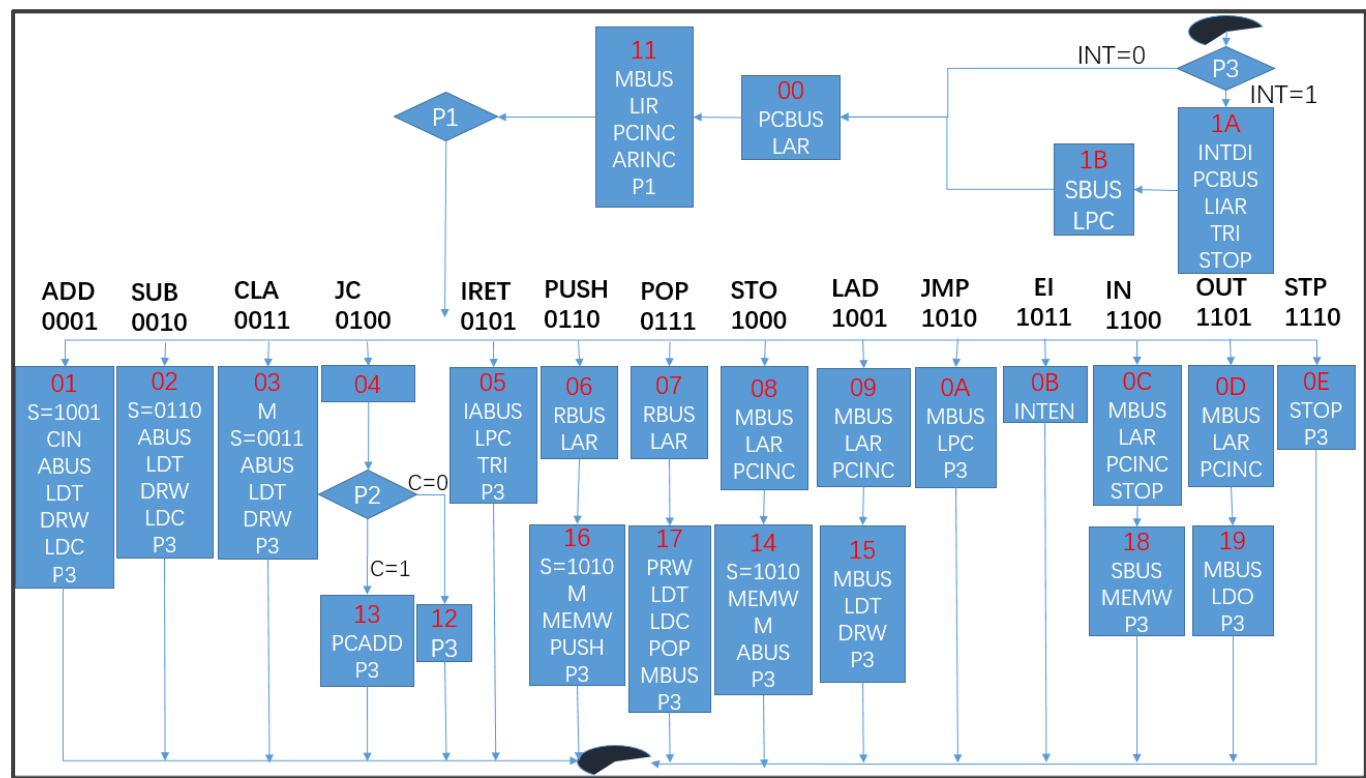
3. 元件：

元件一览表			
元件名			功能
ALU	算术逻辑运算单元	Arithmetic logic state	运算，主要利用74ls181和74ls182芯片实现。
PRO	处理程序	Processor	寄存器+选择器，内含1个累加器AC，三个寄存器R0,R1,R2，通过输入信号选择写入的信号和ALU运算的操作数，DRW信号控制写操作。
DR	数据缓冲寄存器	Data register	用来暂时存放ALU的运算结果。
RAM+ROM	存储器	Random access memory Read only memory	内存部分，用两片27C512（64K 八位ROM）芯片做成地址为16位的ROM部分，用两片64K八位RAM芯片实现地址为16位的RAM。
AR	地址寄存器	Address register	用来记录将要访问的内存地址，可以通过LAR信号完成将总线上数据置入的操作，可以通过ARINC信号完成地址自增操作。
PC	程序计数器	Program counter	用来记录当前执行的程序地址，可以通过PCINC完成地址自增操作，通过PCADD进行地址转移，从而使程序可以循环执行。
IR	指令寄存器	Instruction register	用来存储当前执行的指令，在取值周期通过置数信号LIR有效将取出的地址置入IR。
TIME	时序电路	Time	生成四个相邻且互不相重合的时钟脉冲，用来控制整个程序的时序执行。
ITC	中断触发电路	Interrupt trigger circuit	产生中断信号，在pause按钮按下后产生一个中断请求，并在中断程序执行周期内禁止其他中断请求。
IAR	中断地址寄存器	Interrupt address register	记录终端时程序执行到了地址，在LAR信号作用下将中断地址存入IAR，在中断结束后将中断地址返回。
PSW	状态字寄存器	Programming status word register	记录C信号的状态，可以在程序进入中断时通过TRA信号切换至中断模式，保留主程序的状态字，同时可以存储中断程序的状态字。
SAR	栈地址寄存器	Stack address register	用来存放栈地址，配合push，pop操作实现栈结构。
CM	控制存储器	Command register	将指令转换成40位微地址。
MCR	微命令寄存器	Microcommand register	存储微命令，对垂直型微命令进行译码。本设计采用水平型微指令，无需译码。
ATL	地址转移逻辑	Address transfer logic	通过P字段判别对微指令给出的下一位微地址。
IN	输入元件	Input element	配合IN指令和中断指令，实现程序执行时输入数据或地址。
OUT	输出元件	Output element	配合OUT指令，实现程序执行时查询内存指定区域的数据。
RC	寄存器控制器	Register controller	实现指令和中断微指令对通用寄存器的选择。

4. 指令（共 14 条，含 5 条双字长、9 条单字长指令）：

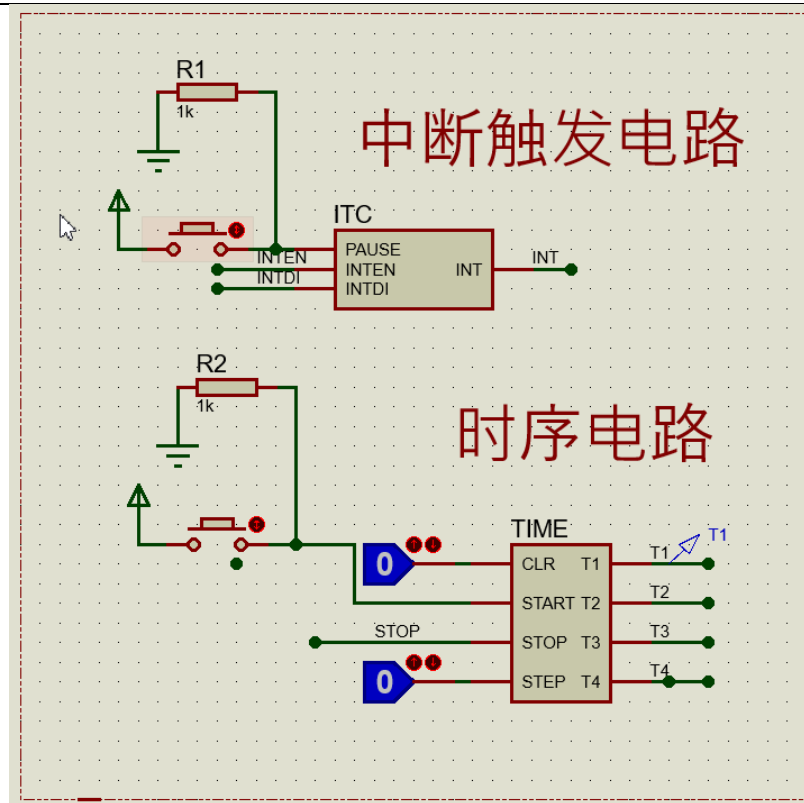
	12-15	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0001	Ri		0									
SUB	0010	Ri		0									
CLA	0011	0											
JC	0100	0				X							
IRET	0101	0											
PUSH	0110	Ri		0		-1							
POP	0111	0		Ri		0							
STO	1000	Ri		0									
	X												
LAD	1001	0		Ri		0							
	X												
JMP	1010	0											
	X												
EI	1011	0											
IN	1100	0											
	X												
OUT	1101	0											
	X												
STP	1110	0											

5. 微指令（共 30 条）：

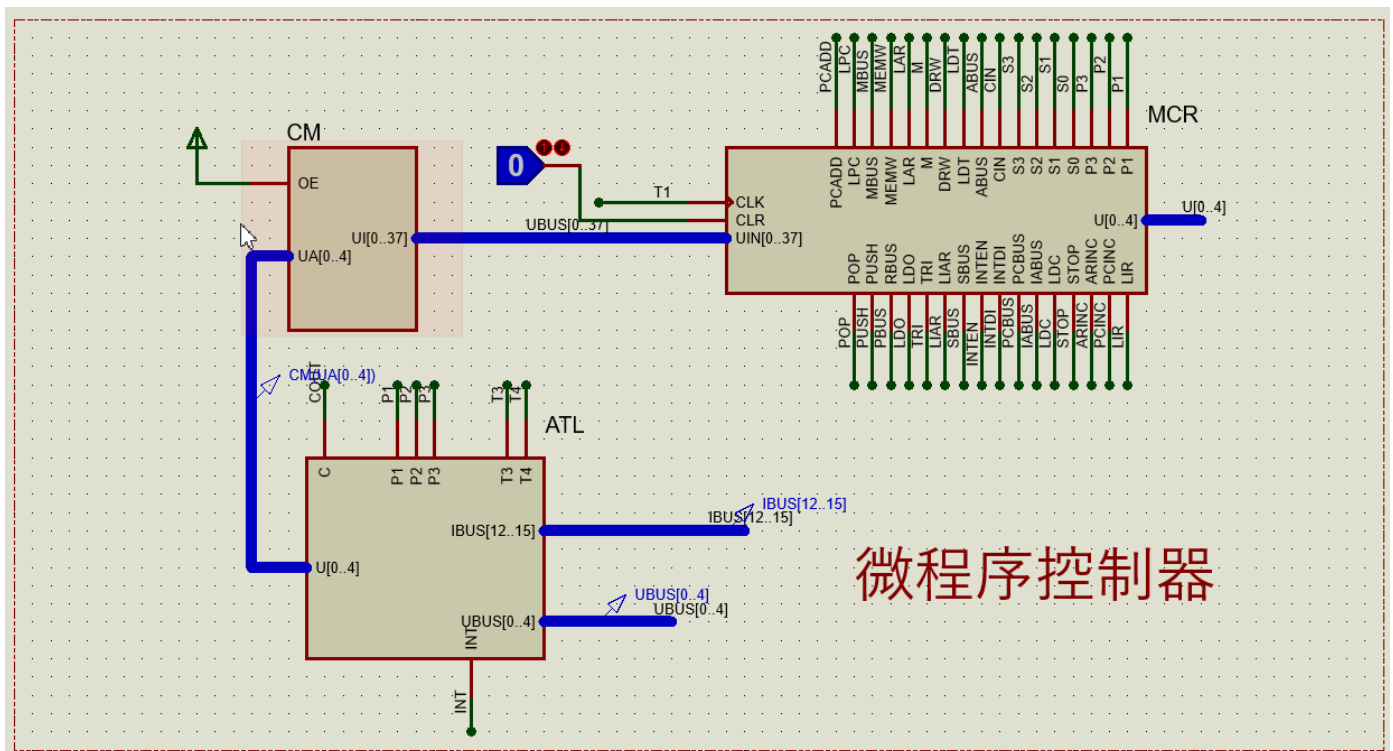


6. 电路图：

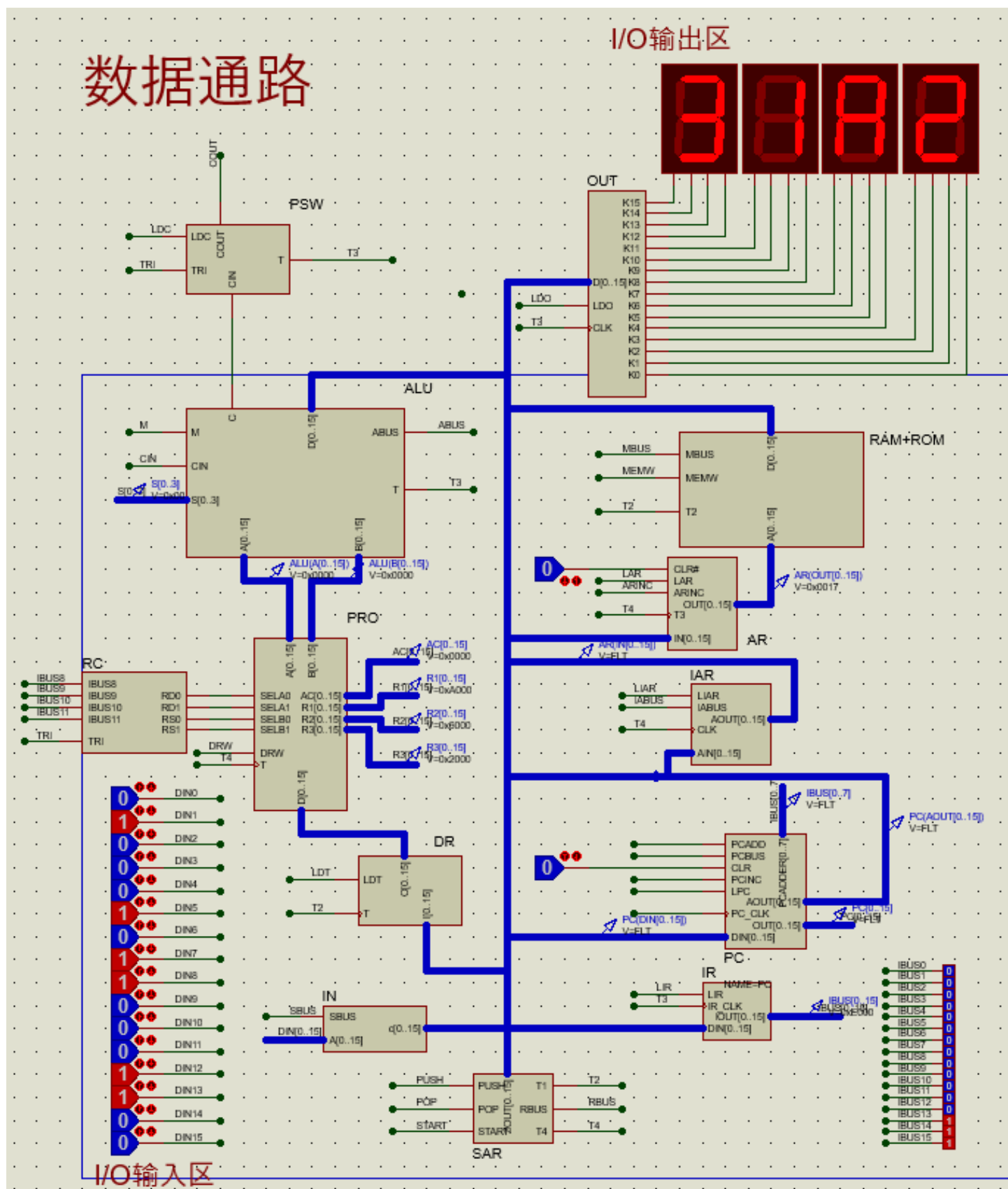
Part A 时序和中断触发电路



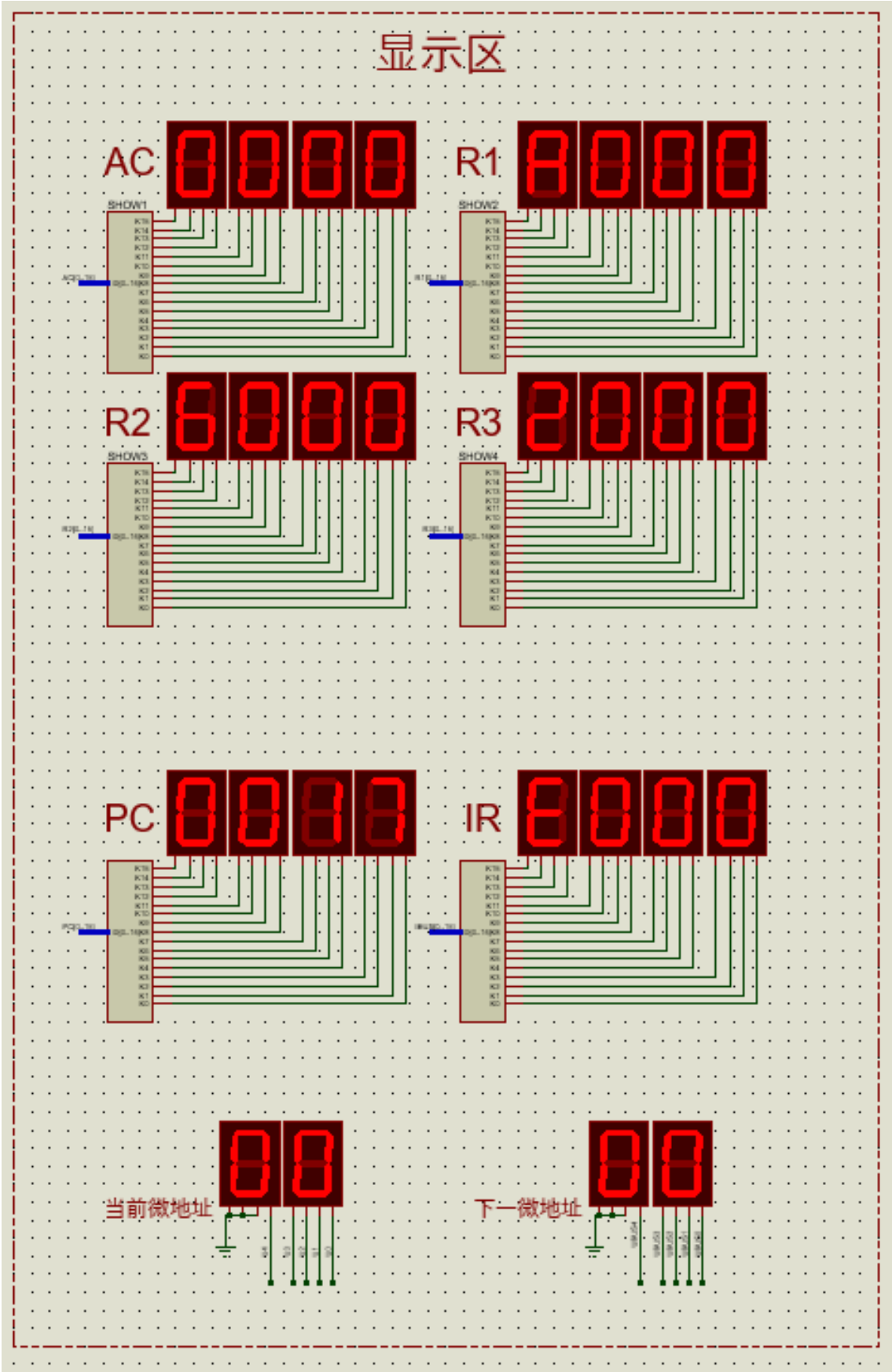
Part B 微程序控制电路



Part C 数据通路



Part D 显示区



程序测试：

开关说明	
START	1.开机功能，一次脉冲后开始产生时序信号 2.输入确认功能，程序等待输入时，输入完成后，通过一次START脉冲继续执行程序。
PC-CLR	PC清零功能，开机后通过按一次PC-CLR按钮，可以将指令地址清零，回到程序的开始地址。
STEP	操作模式切换开关，当STEP为0时，程序为连续运行模式，当STEP为1时，程序为单拍运行模式。
PAUSE	程序中断请求信号，单击PAUSE，程序收到中断请求。

操作步骤
1.主程序操作步骤
1) 开始PROTUES仿真
2) 点按PC-CLR，清零PC
3) 点按START，开始执行程序
4) 执行到停机指令后，程序运行结束。
2.I/O交换
1) 当程序运行到IN指令后，程序暂停，等待从输入区输入数据。
2) 从输入区输入需要的16位数据。（如：5555H）
3) 单击START确认输入，继续运行程序。
3.中断操作步骤
1) 单击PAUSE，产生中断请求。
2) 程序将继续运行，直到程序执行完当前指令，然后程序将停止等待输入。
3) 在输入区输入中断地址，在我们设计的程序中，两个中断子程序分别位于主存3000和3100位置。输入完地址后，单击START确认输入，程序便会跳转执行中断子程序。

测试程序							
内存地址	汇编指令	机器代码	AC	R1	R2	R3	说明
0000	LAD(4000)->R1	9100	0000	A000	0000	0000	将地址4000的数据存入R1 (4000存放的初值为A000)
0001		4000					
0002	LAD(4001)->R2	9200	0000	A000	4000	0000	将地址4001的数据存入R2 (4001存放的初值为4000)
0003		4001					
0004	ADD R1	1400	A000 0000	A000 A000	4000 6000	0000 0000	AC+R1->AC (分别对应2轮访问该指令)
0005	JC 07	4007	A000	A000	4000	0000	通过C判断是否跳转至000D
0006	SUB R2	2800	6000	A000	4000	0000	AC-R2->AC
0007	STO AC->(8000)	8000	6000	A000	4000	0000	将AC的数据 存入地址8000
0008		8000					
0009	LAD(8000)->R2	9200	6000	A000	6000	0000	将地址8000的数据存入R2
000A		8000					
000B	JMP(0004)	A000	6000	A000	6000	0000	跳转至0004
000C		0004					
内存地址	汇编指令	机器代码	AC	R1	R2	R3	说明
000D	LAD(4002)->R3	9300	0000	A000	6000	2000	将地址4002的数据存入R3
000E		4002					
000F	PUSH AC	6081	0000	A000	6000	2000	AC数据入栈
0010	CLA	3000	0000	A000	6000	2000	清零AC
0011	POP AC	7000	0000	A000	6000	2000	栈顶数据进入AC
0012	ADD R3	1C00	X000	A000	6000	2000	AC+R3->AC(多次AC+=2)
0013	JC 02	4002	X000	A000	6000	2000	通过C判断是否跳转之0016
0014	JMP(000F)	A000	X000	A000	6000	2000	跳转至000F
0015		000F					
0016	STP	E000					停机

说明：

本主程序覆盖 LAD、ADD、SUB、JC、STO、JMP、PUSH、CLA、POP、STP 等 10 条指令，功能齐全。（ 剩余 4 条指令在中断子程序中进行验证 ）。

Part B 中断子程序

内存地址	汇编指令	机器代码	说明
中断子程序A			
3000	SUB R3	2C00	AC-R3->AC
3001	SUB R3	2C00	AC-R3->AC
3002	EI	B000	中断允许
3003	IRET	5000	中断返回
中断子程序B			
3100	IN	C000	从输入区输入数据
3101		F000	
3102	ADD R3	1C00	AC+R3->AC
3103	OUT	D000	输出数据
3104		F000	
3105	EI	B000	中断允许
3106	IRET	5000	中断返回

说明：

本测试程序包含 2 个中断子程序，子程序 A 验证基础中断功能，子程序 B 验证 IN 和 OUT 指令。

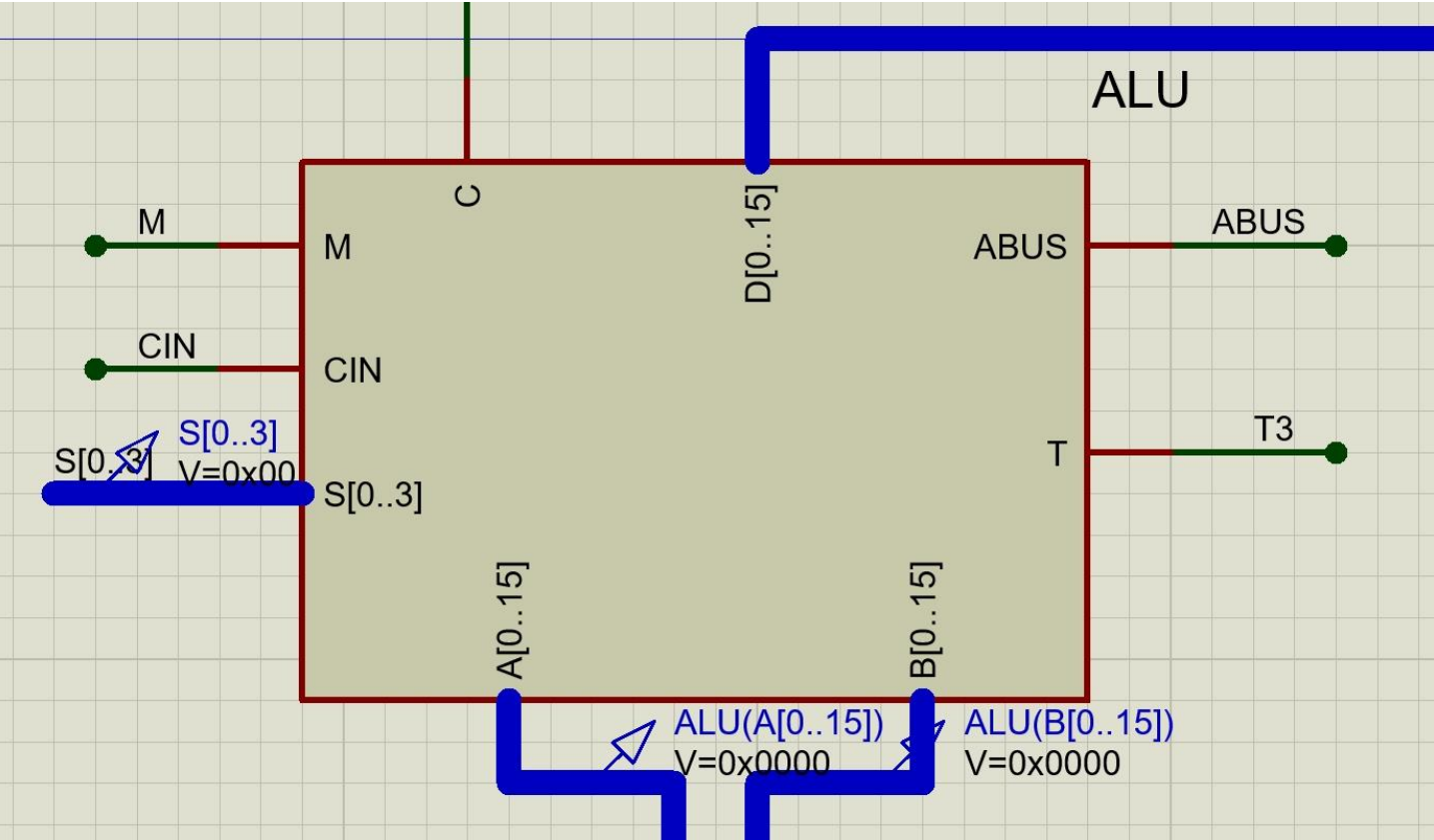
自制元件介绍:

1. ALU

实现功能:

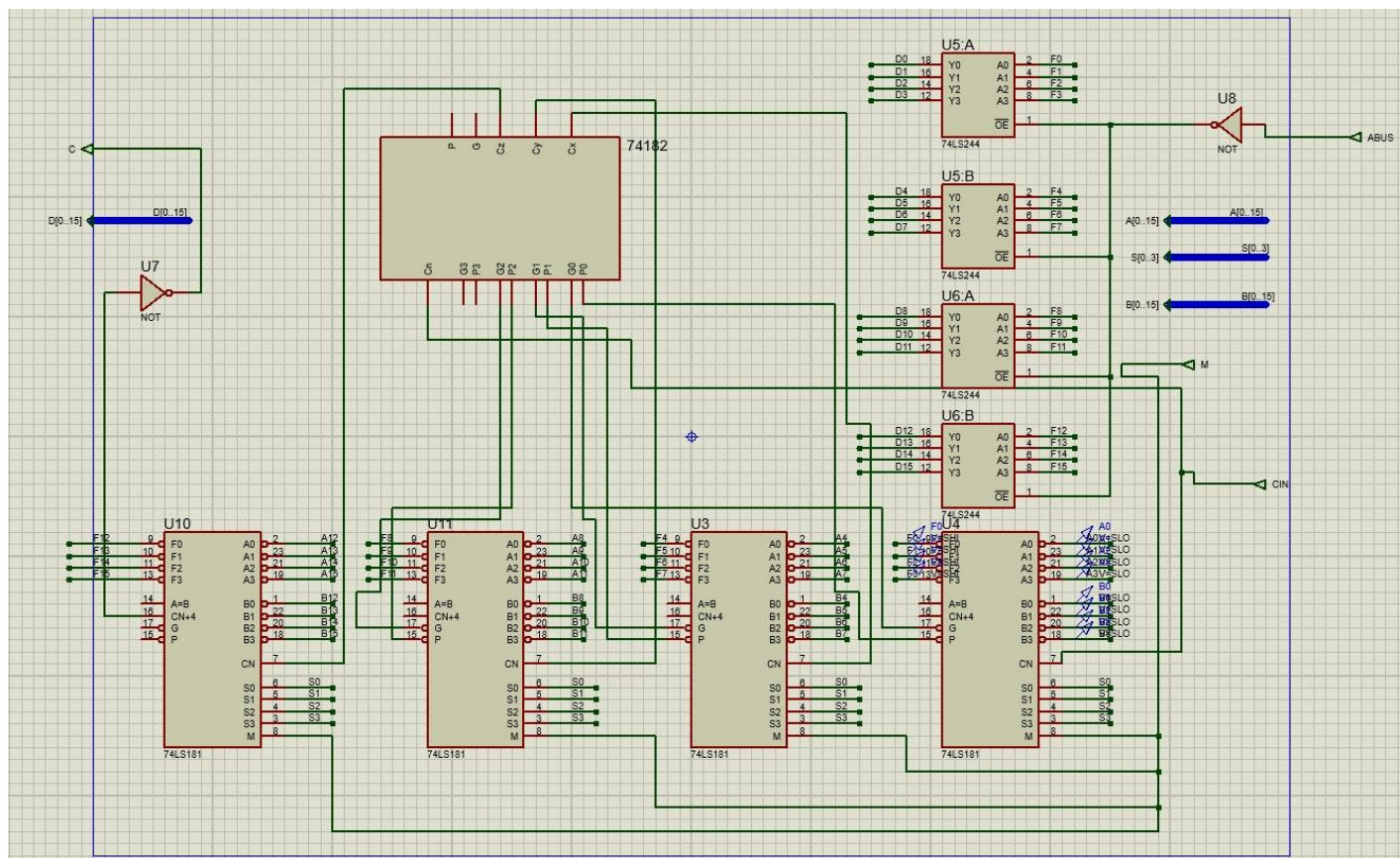
通过 M、CIN、S3、S2、S1、S0，对 A、B 输入的两个 16 位数据进行算术逻辑运算，进行并行运算，并将结果输出到总线。

外部管脚:

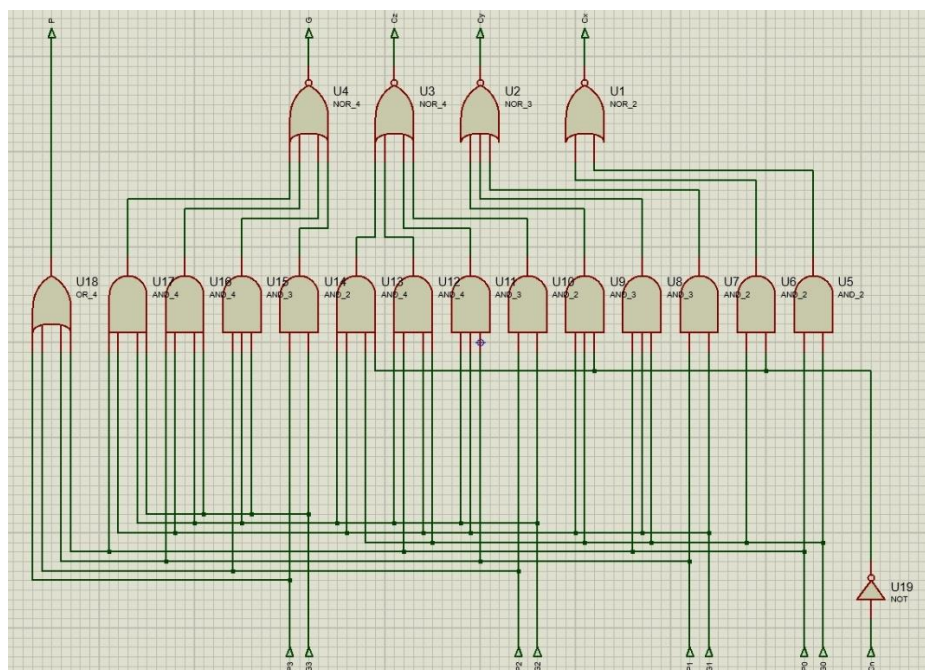


管脚功能介绍		
A[0..15]	B[0..15]	将进行运算的两个 16 位数据
M		控制算术或逻辑运算
CIN		进位端
S[0..3]		控制运算方式
T		脉冲信号
ABUS		输出使能端
D[0..15]		输出的 16 位计算结果
C		输出进位信号（用于 JC 判断）

内部子电路:



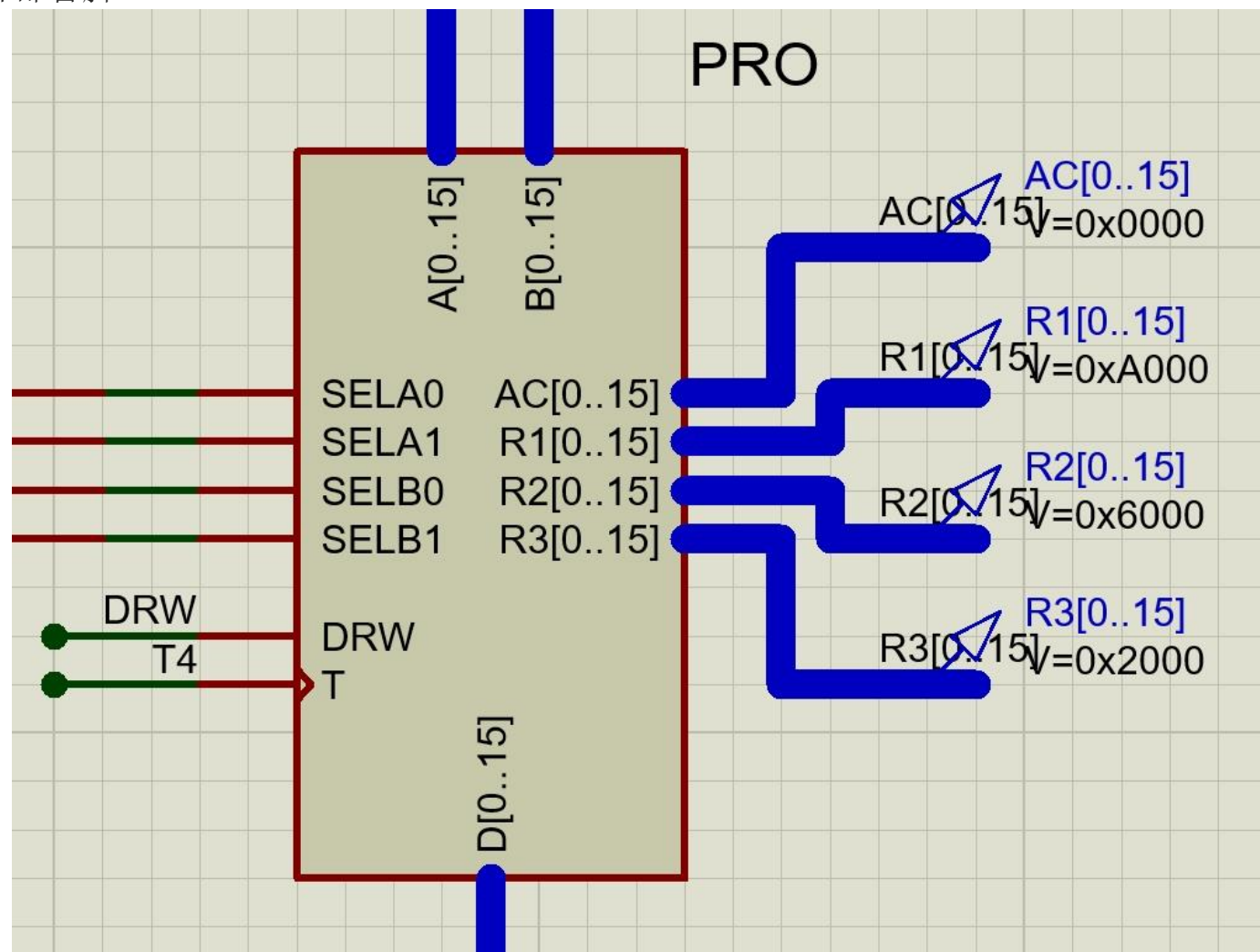
说明：ALU 内部由四片 74LS181、一片 74182、四片 74LS244 组成。四片 74LS181 通过高低位组合，实现两个 16 位数据的运算。四片 74LS244 通过高低位组合，实现 16 位数据的三态门控制。一片 74182 实现并行进位（74182 为自制元件，下附内部子电路图）。



2. PRO

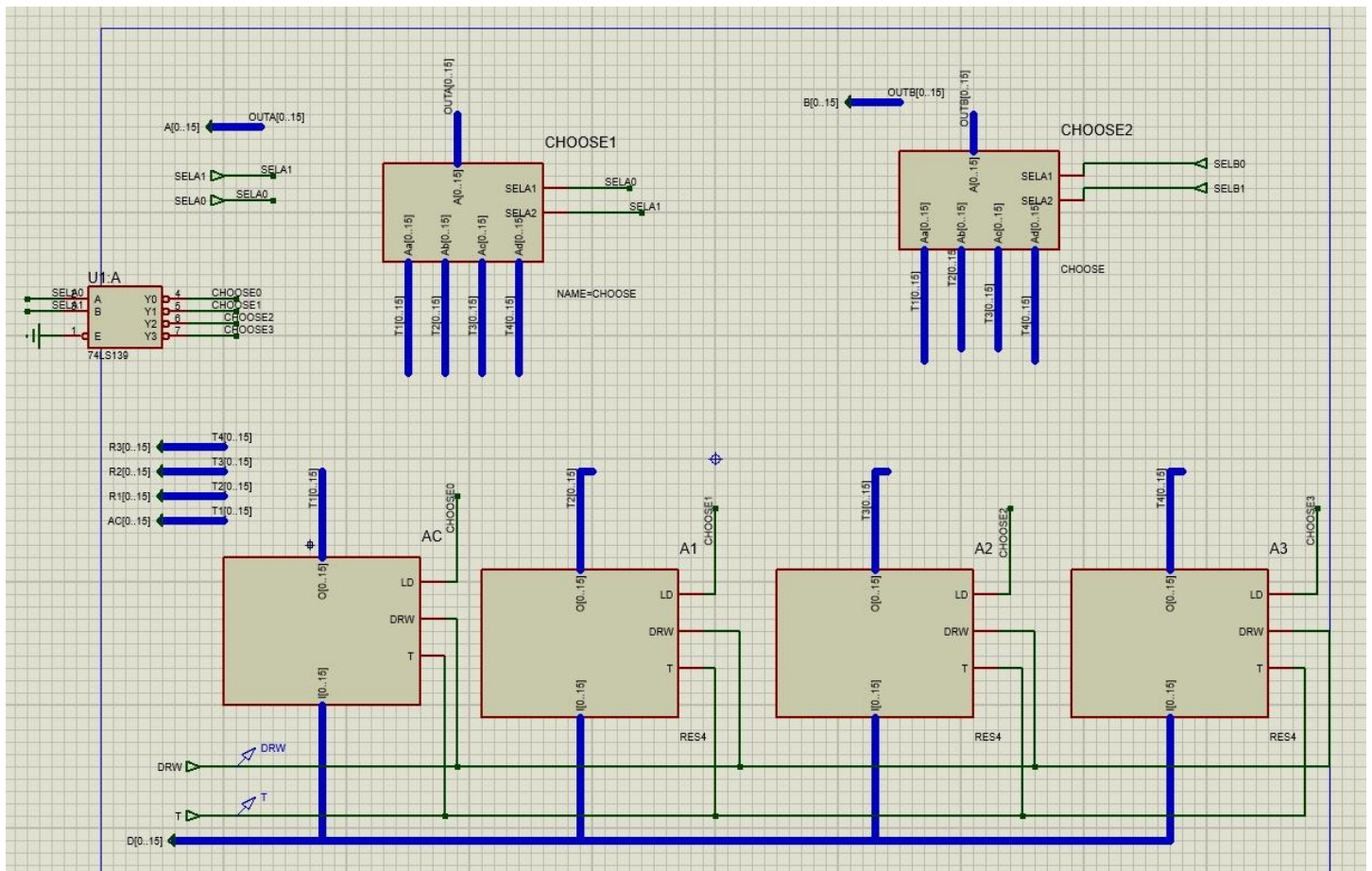
实现功能：
16 位寄存器+4 选 1 选择器，实现从总线往寄存器写入，和从寄存器选择输出的功能。

外部管脚：



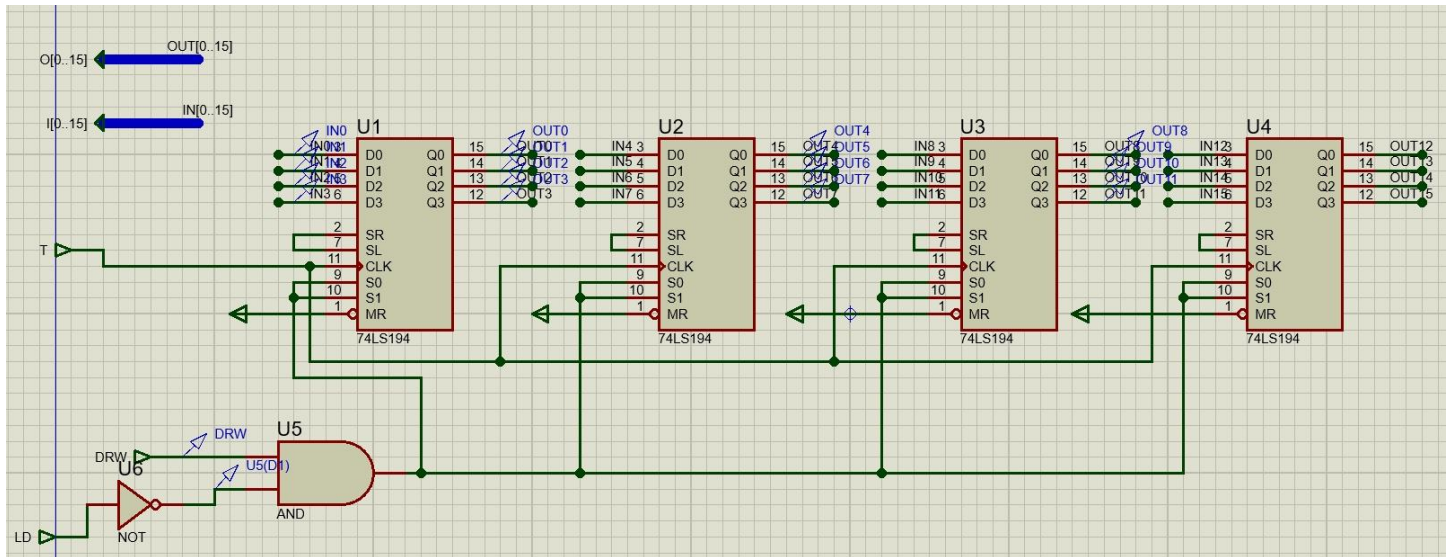
管脚功能介绍	
D[0..15]	从总线输入的 16 位数据
SELA0 SELA1	写寄存器选择信号
SELA0 SELA1 SELB0 SELB1	读取选择信号
DRW	写入使能端
T	脉冲信号
AC[0..15] R1[0..15] R2[0..15] R3[0..15]	累加器和三个寄存器数据输出显示
A[0..15] B[0..15]	输出的两个 16 位数据

内部子电路:



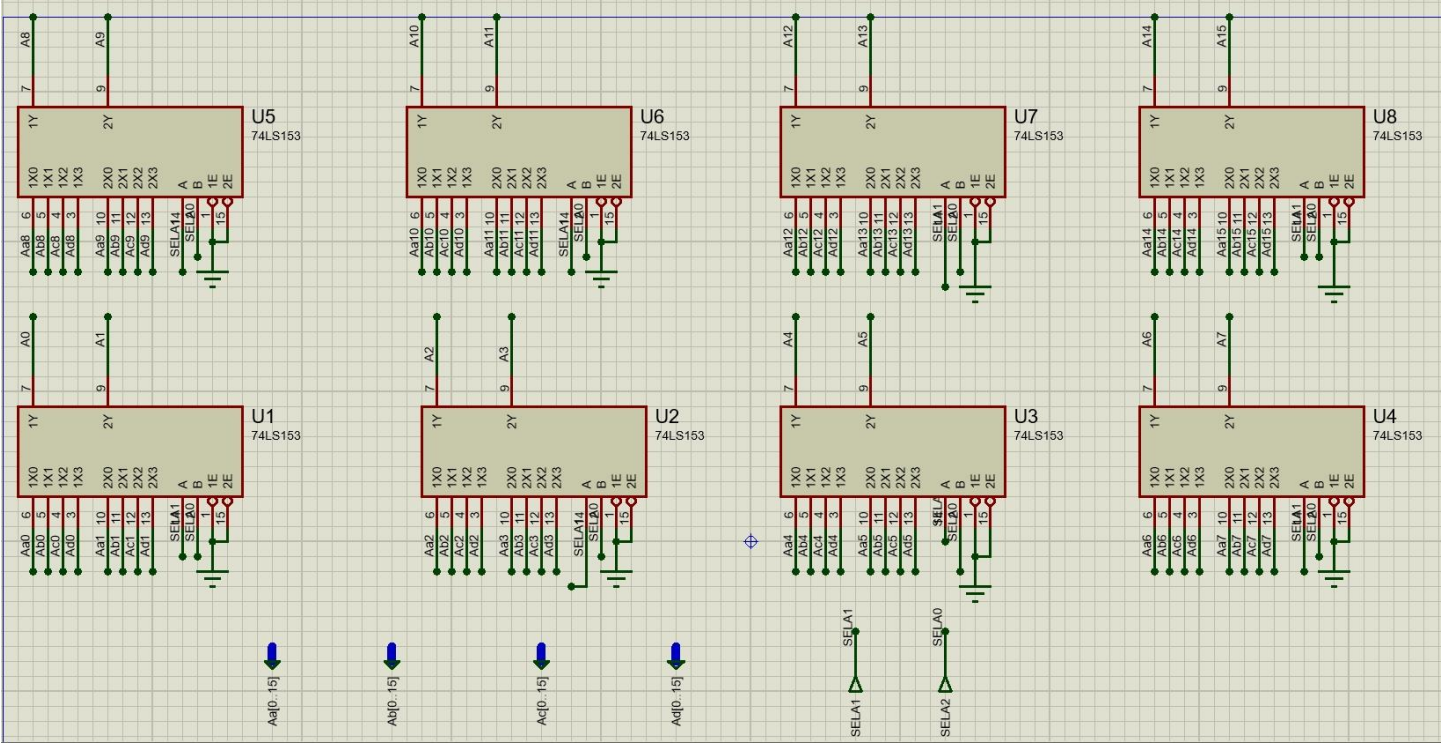
说明：PRO 由一个累加器、三个寄存器、一个 2-4 译码器、两个 4 选 1 选择器组成。其中 2-4 译码器使用了 74LS139。

寄存器（累加器）子电路：



寄存器（累加器）内部由四片 74LS194 组成，寄存 16 位数据。

4 选 1 选择器子电路：



4 选 1 选择器由 8 片 74LS153 组成，选择 16 位数据。

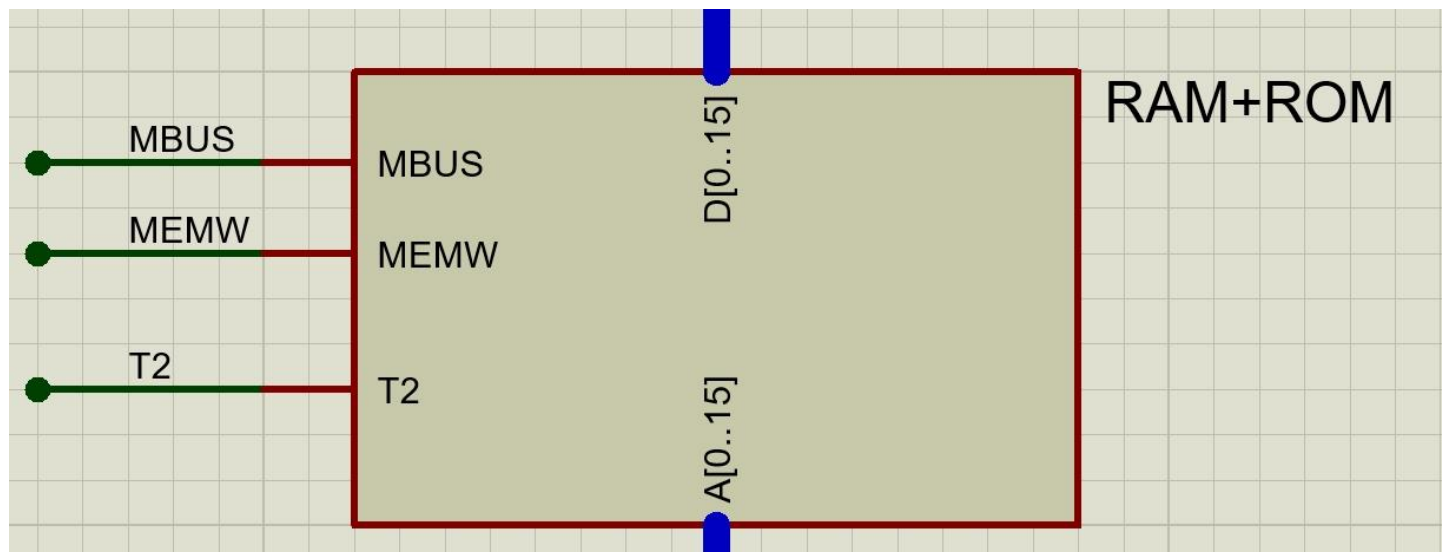
3. ROM+RAM

实现功能：

仿真前，ROM 中拷入执行的程序。程序运行中，RAM 作为可读写存储器。
RAM 中划分成 8000-DFFF 数据读写区，E000-FFFF 堆栈区，FFFE-FFFF I/O 区。

memory		
ROM	0000-2FFF	主程序
	3000-3FFF	中断子程序
	4000-7FFF	初始数据
RAM	8000-CFFF	主程序写入的数据
	D000-DFFF	中断写入的数据
	E000-FFFF	栈
	FFFE-FFFF	I/O

外部管脚：



管脚功能介绍	
A[0..15]	从 AR 输入的 16 位地址
D[0..15]	从总线输入 16 位数据 或输出 16 位数据到总线
MBUS	读取使能端
MEMW	写入使能端
T	脉冲信号

说明：

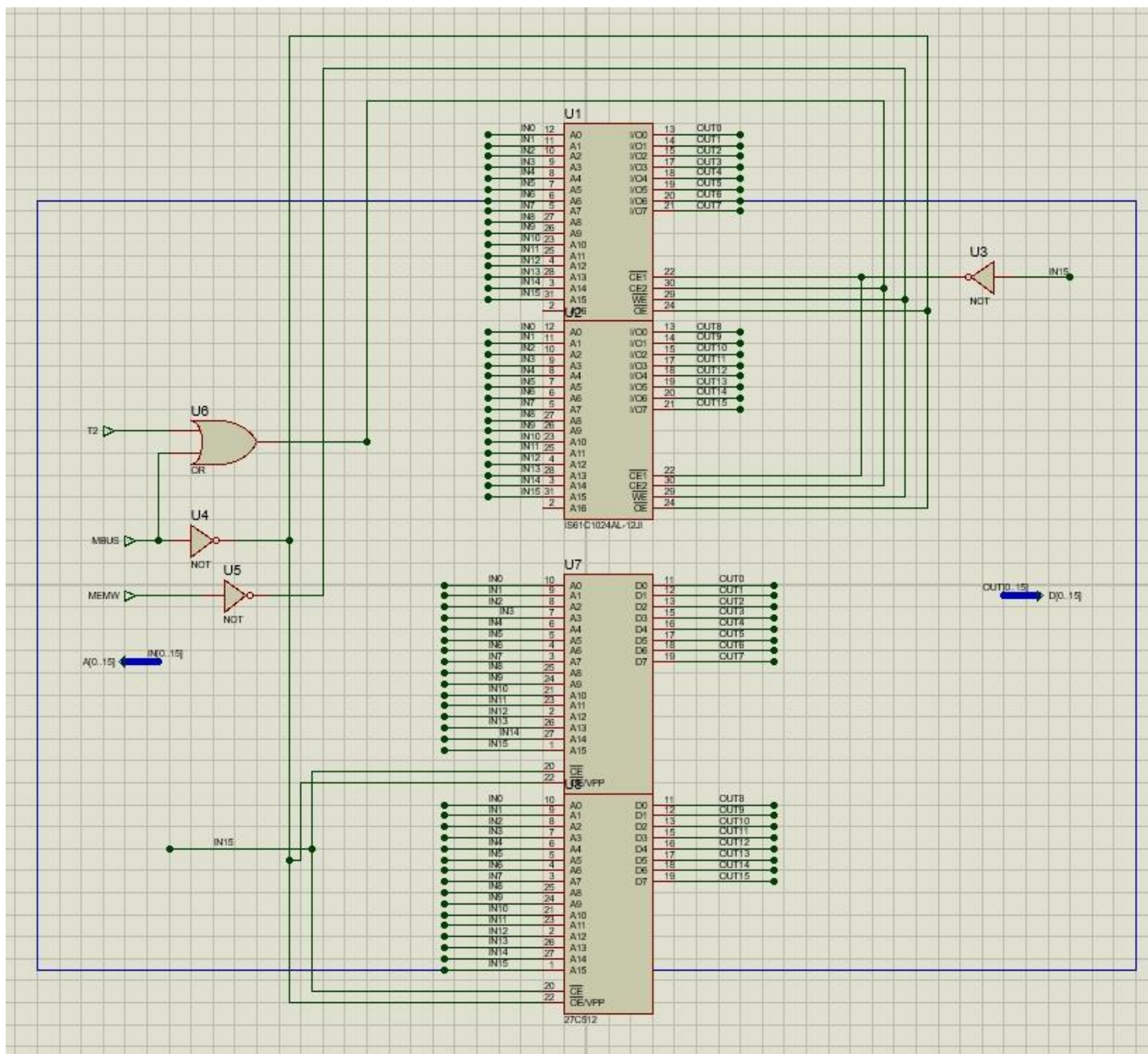
ROM 只能读出数据，不能写入。关机数据仍然保留；RAM 可读可写，关机数据丢失。通过最高位对 ROM 和 RAM 进行片选。

ROM 容量为 32K*16，RAM 容量为 32K*16。存储器整体容量为 64K*16。

写入存储器时：MEMW 有效，将总线上的 16 位数据从 D[0..15] 写入 A[0..15] 的 16 位地址中。

读取存储器时：MBUS 有效，将 A[0..15] 的 16 位地址中的 16 位数据从 D[0..15] 读取到数据总线上。

内部子电路：



说明：

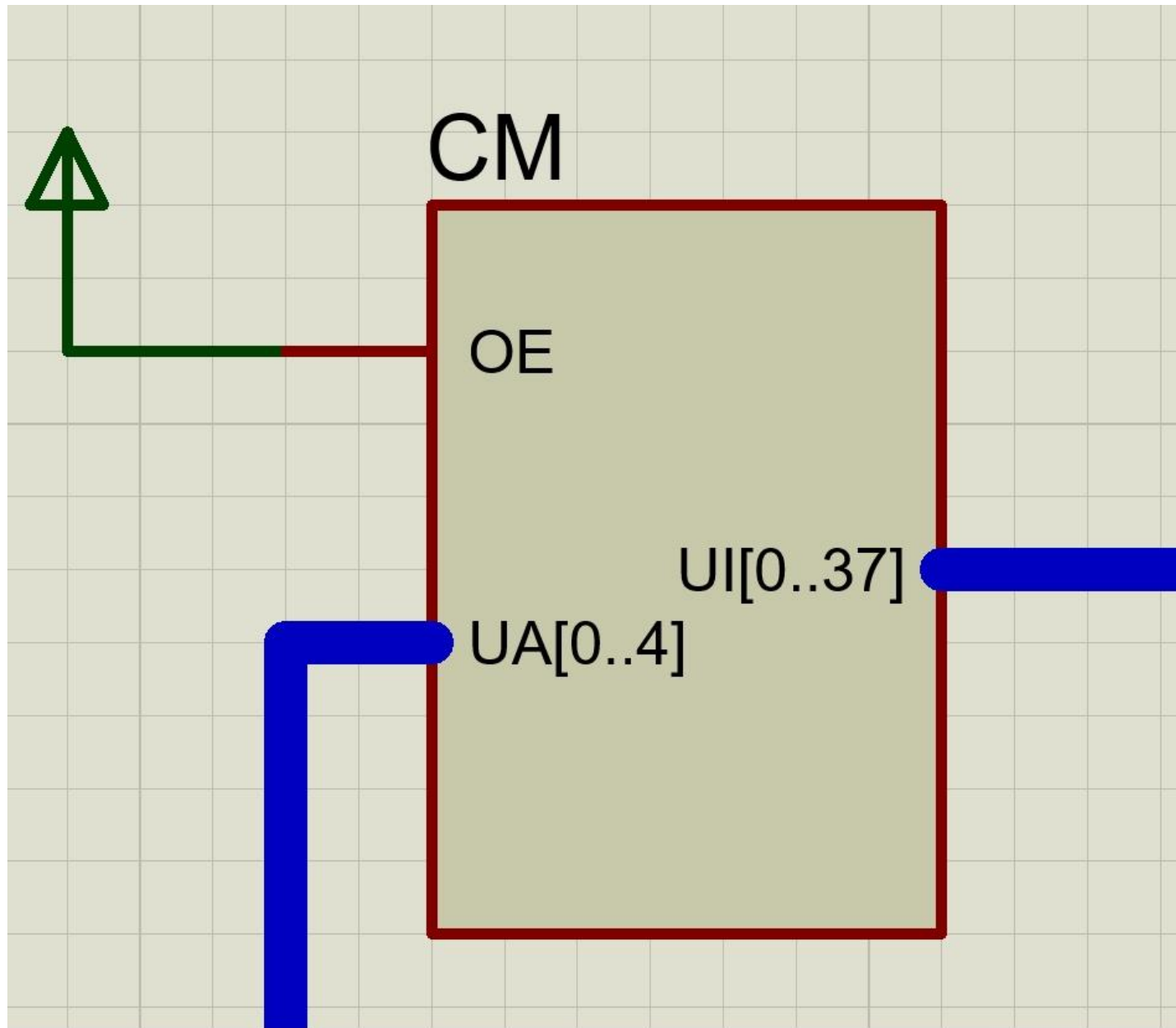
ROM 由两片 27C512 组成，RAM 由两片 IS61C1024AL-12JI 组成。
其中 27C512 芯片可读取 HEX 文件。

4. CM

实现功能：

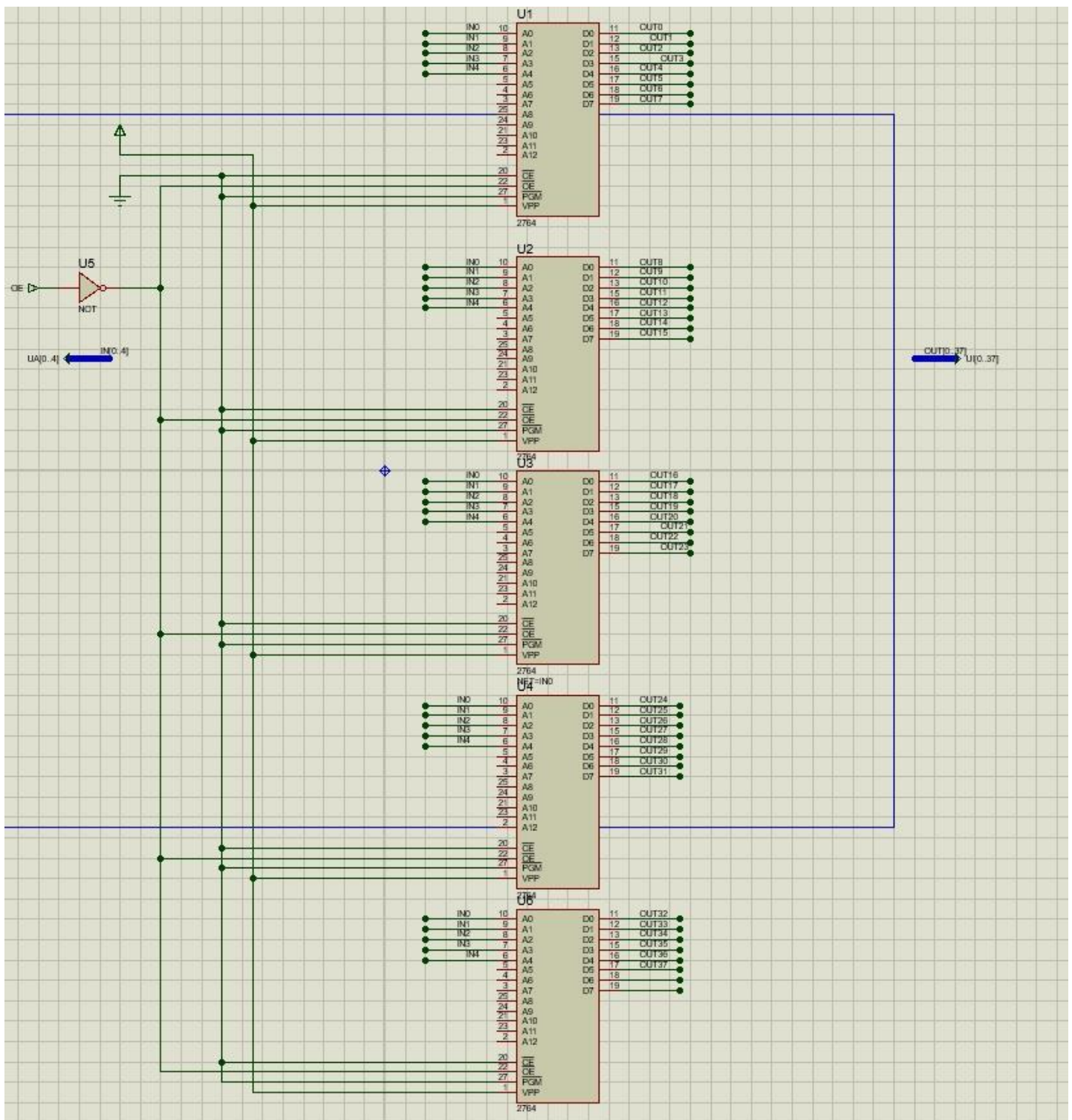
通过当前微地址，译出 38 位微指令，通过 38 位微指令，控制数据通路信号。

外部管脚：



管脚功能介绍	
UA[0..4]	输入当前 5 位微地址
UI[0..37]	输出 38 位微指令地址
OE	读取使能端

内部子电路:



说明:

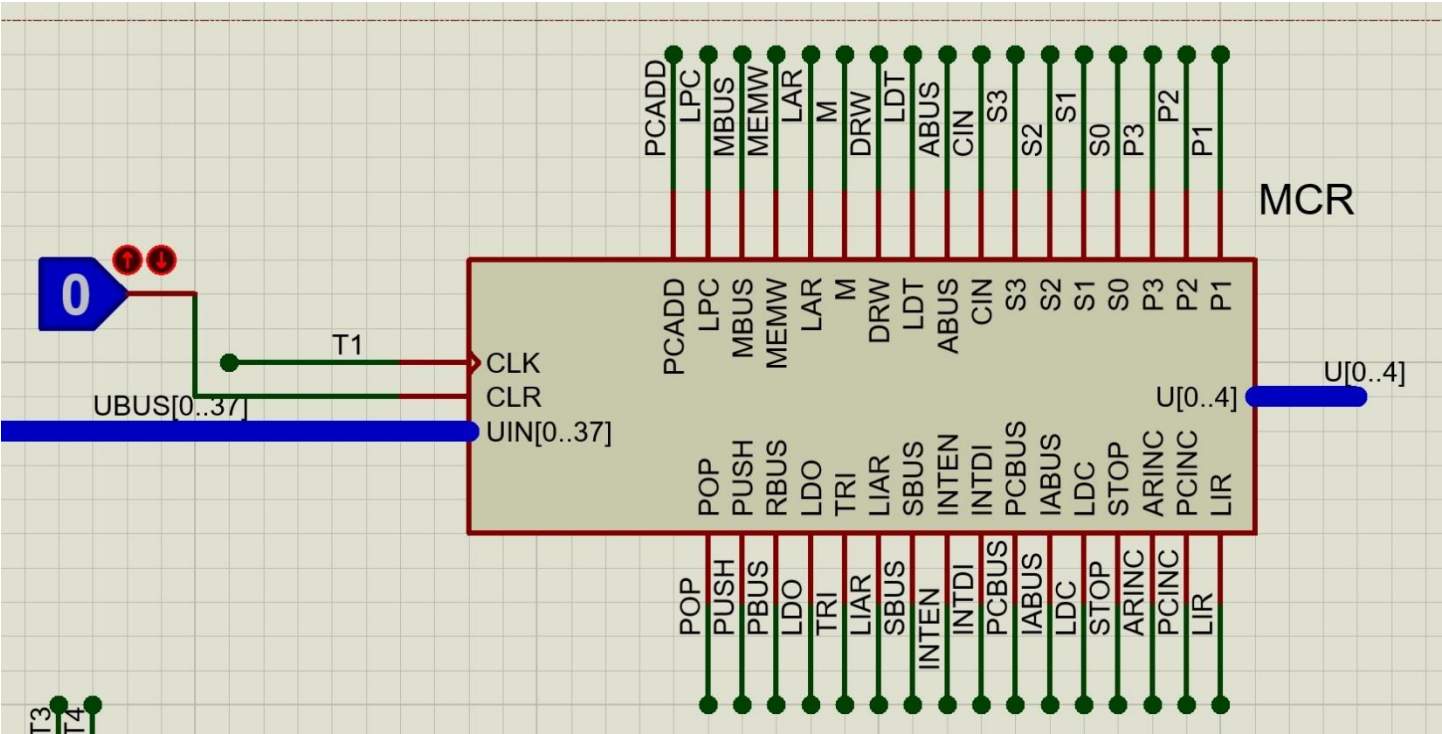
由五片 2764 组成，输出由高到低组成 38 位。
2764 芯片内数据通过 .HEX 文件写入。

5. MCR

实现功能：

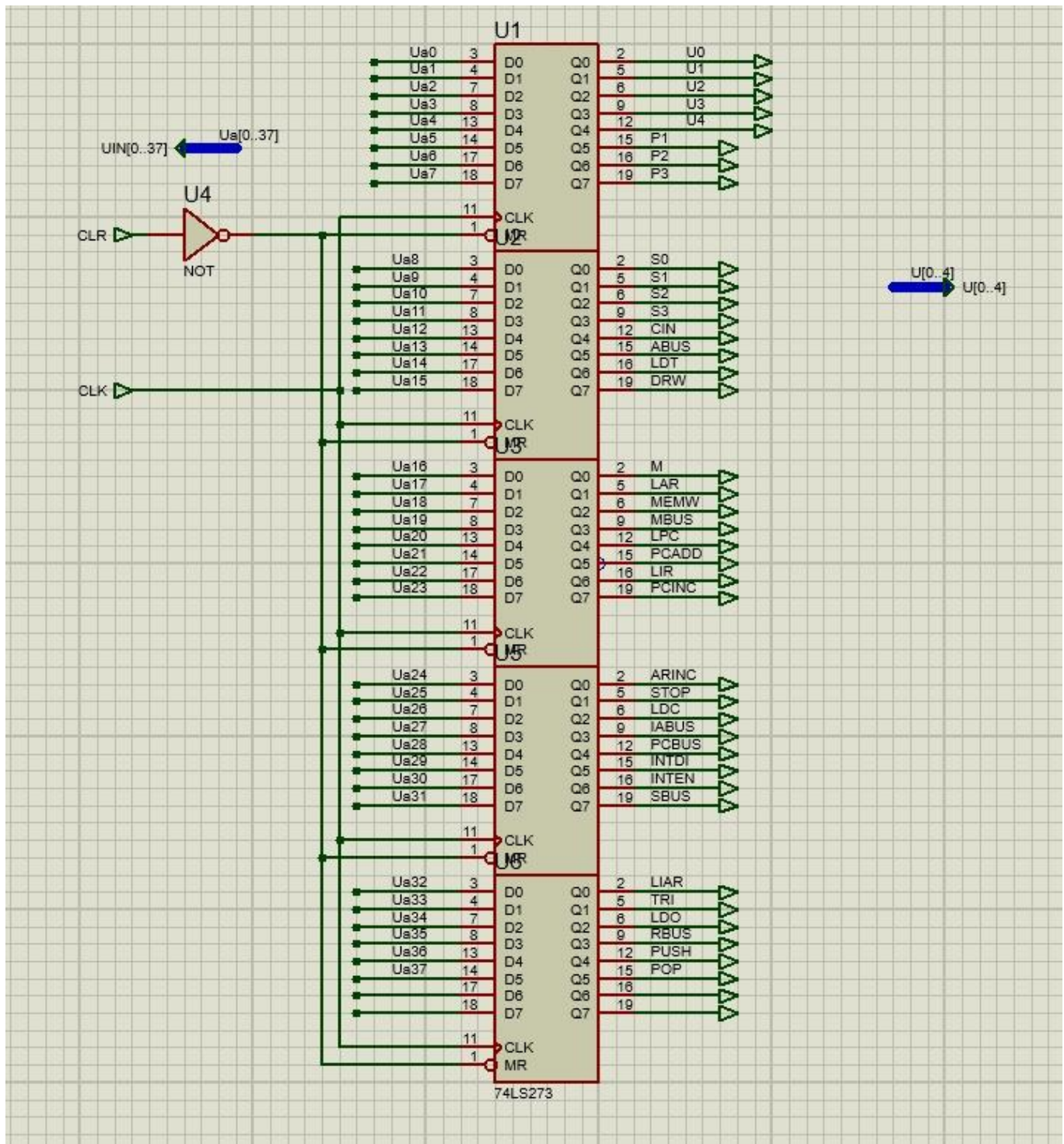
储存微指令，对 CM 中垂直微指令进行译码，采用水平微指令。对数据通路进行控制。

外部管脚：



管脚功能介绍	
CLR	清零信号
CLK	脉冲信号
UIN[0..37]	输入 38 位微指令地址
U[0..4]	输出下一个微地址
其余	控制信号

内部子电路：



说明：MCR 采用五片 74LS273。

工程设计思路:

1. 设计指令系统

本次 CPU 实验共要求 10 条指令:

序号	指令格式	功能描述	备注
1	STO (XXXX),Ri	Ri->存储器/IO (XXXX)	R 有四个 8 位, 包含 AC
2	LAD Ri,(XXXX)	存储器/IO (XXXX) ->Ri	地址: 16 位, 64KB
3	ADD Ri	Ri+AC->AC	4 个通用寄存器和累加器
4	SUB Ri	AC-Ri->AC	
5	JMP XXXX	转移到 XXXXH, PC=XXXX	64KB 范围
6	CLA	AC=0	
7	JC XX	CF=1, PC=PC+XX	XX: -128~127
8	IRET	中断返回	恢复现场
9	PUSH Ri	SP=SP-1,Ri->(SP)	写 1 个字节
10	POP Ri	(SP)->Ri,SP=SP+1	

设计思路:

1. 由于有 14 条指令, 所以操作码至少需要 4 位。
2. 其中 STO、LAD、JMP、IN、OUT 五条指令需要至少 20 位, 又只有一条数据总线, 所以无法用 16 位数据总线传输。

有两种解决方法:

- (1) 采用双字长指令;
- (2) 采用间接寻址的方法;

方法 (2) 由于初始寄存器中没有数据, 开始间接寻址较麻烦。所以本次指令系统采取部分指令双字长的方式

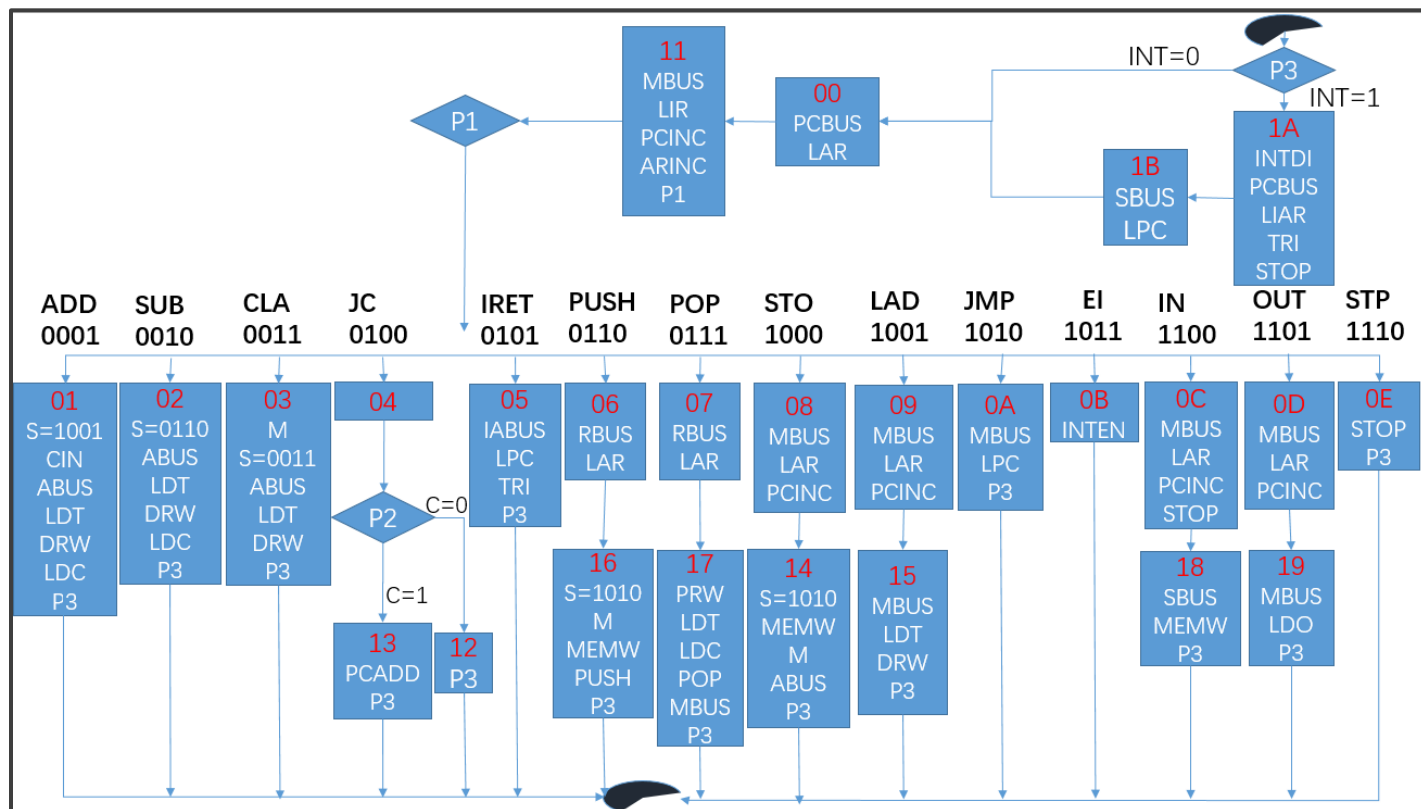
3. STO、LAD、JMP、IN、OUT 5 条指令需要双字长, 另外 9 条指令采用单字长。
4. 为了方便逻辑判断双字长, 五条双字长指令和 EI, STP 操作码位 1XXX, 另外 7 条指令为 0XXX。

指令系统如下：

	12-15	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0001	Ri		0									
SUB	0010	Ri		0									
CLA	0011	0											
JC	0100	0				X							
IRET	0101	0											
PUSH	0110	Ri		0		-1							
POP	0111	0		Ri		0							
STO	1000	Ri		0									
	X												
LAD	1001	0		Ri		0							
	X												
JMP	1010	0											
	X												
EI	1011	0											
IN	1100	0											
	X												
OUT	1101	0											
	X												
STP	1110	0											

（注：JC 中 X 为 8 位偏转值表示-127—127；STO、LAD、JMP、IN、OUT 中 X 为 16 位地址）

2. 微指令格式设计



根据微程序流程图设计微指令格式:

流程图中共有 4 逻辑跳转:

- (1) 由当前微地址跳转到固定微地址;
- (2) 由当前微地址跳转到 P 字段判断;
- (3) 由 P 字段逻辑运算跳转到相应微地址;
- (4) 由当前微地址跳转到中断;

根据这四种逻辑，进行微程序跳转。

38 位微指令，其中最后 5 位 U4-U0 是下一个微地址。下一个微地址固定的跳转，u4-u0 为固定值，其余 u4-u0 为 00000，经过 P 字段判断跳转到下一地址。

U7、U6、U5 表示 P 字段。

其余 31 位，为控制数据通路的信号。

微指令格式:

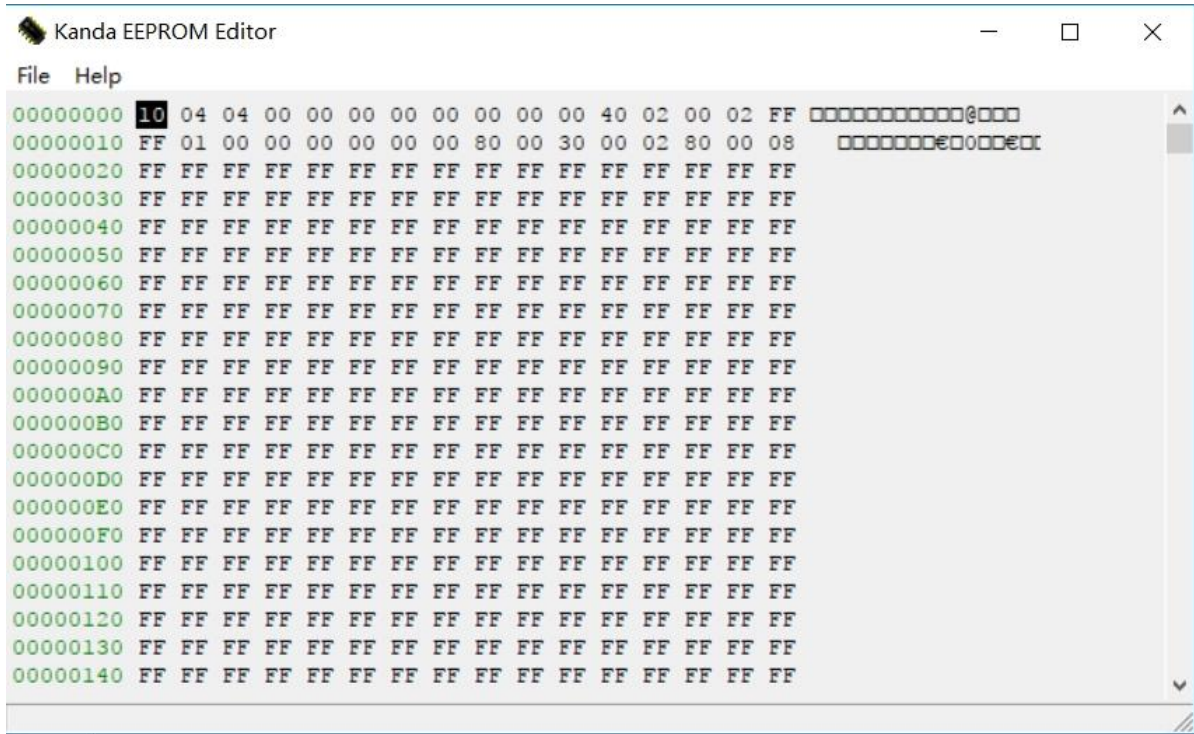
		00	11	1A	1B	1C	1D	01	02	03	04	13	12	05	1E	1F	06	16	07	17	08	14	09	15	0A	0B	0C	18	0D	19	0E
0	u0	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0
1	u1	0	0	1	0	0	0	0	0	0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	u2	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
3	u3	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
4	u4	1	0	1	1	1	0	0	0	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	0
5	P1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	P2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	P3	0	0	0	0	0	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1
8	S0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	S1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
10	S2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	S3	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
12	CIN	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	ABUS	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
14	LDT	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
15	DRW	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
16	M	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
17	LAR	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	0
18	MEMW	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0
19	MBUS	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1	1	1	0	1	0	1	1	0
20	LPC	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
21	PCADD	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	LIR	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	PCINC	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0
24	ARINC	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	STOP	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
26	LDC	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	IABUS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	PCBUS	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	INDTI	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	INTEN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
31	SBUS	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
32	LIAR	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	TRI	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	LDO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
35	RBUS	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	

（注：图中横轴为 30 个微地址，纵轴位 38 位微指令）

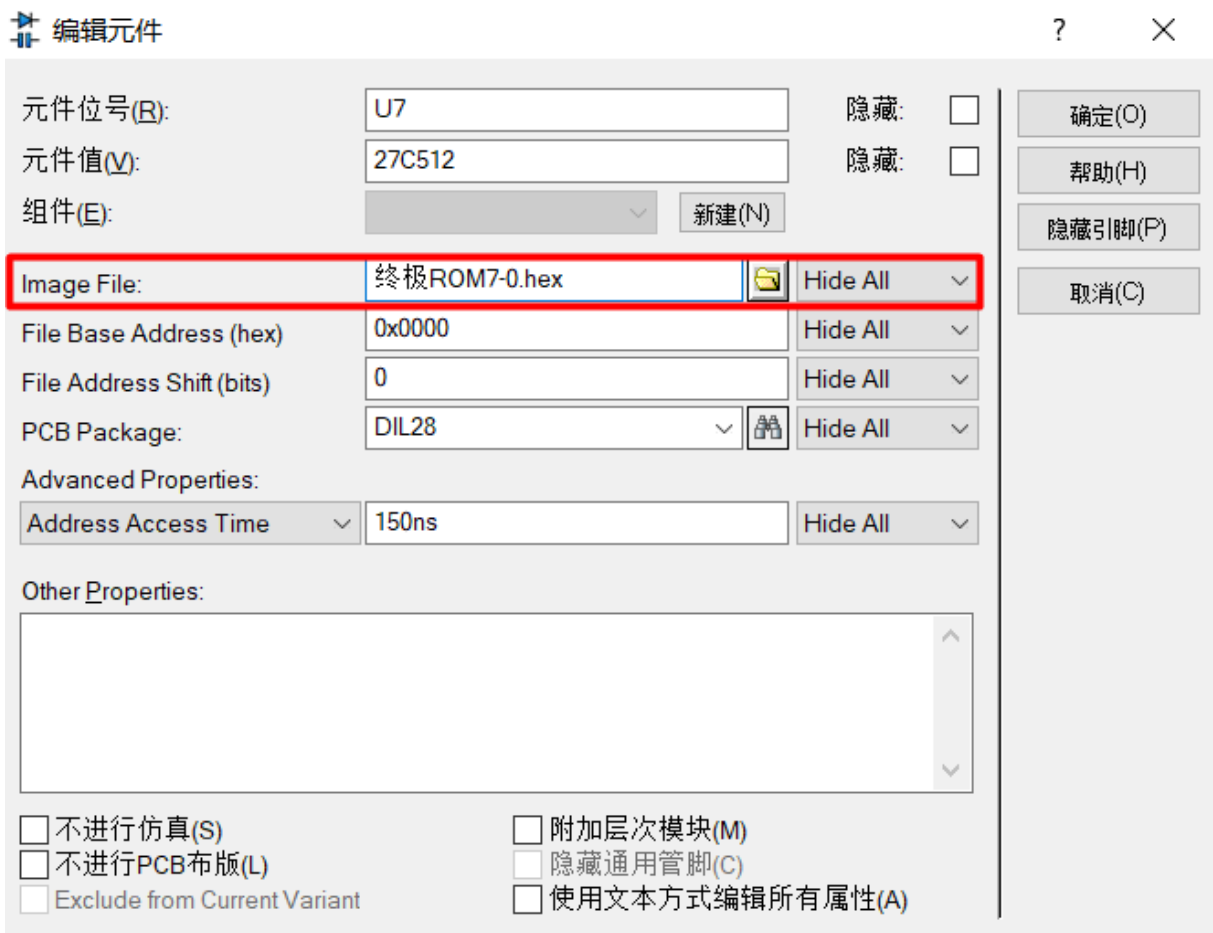
将设计好的 38 位微指令通过 .HEX 文件存入 CM。

3. ROM、CM 文件写入

. HEX 文件写入 ROM 以及 CM，使用了“EEPROMEDIT.EXE”。










编辑元件，通过下图红框中将文件导入。



每个.HEX 文件为 16 位地址，8 位数据。
所以，ROM 需要两个.HEX 文件，CM 需要五个.HEX 文件。

7 个文件命名如下（文件见附件）：

 CODE7-0.hex	2018/1/8 22:14	HEX 文件	1 KB
 CODE15-8.hex	2018/1/7 22:54	HEX 文件	1 KB
 CODE23-16.hex	2018/1/7 22:51	HEX 文件	1 KB
 CODE31-24.hex	2018/1/7 22:49	HEX 文件	1 KB
 CODE39-32.hex	2018/1/7 22:47	HEX 文件	1 KB
 终极ROM7-0.hex	2018/1/10 23:09	HEX 文件	1 KB
 终极ROM15-8.hex	2018/1/8 20:43	HEX 文件	1 KB

4. 测试程序设计

本次实验中，我共设计两个测试程序，一个初级版，一个终极版。
初级版只包含 LAD、ADD、SUB、STO 四条指令，用于验证各硬件正确性。

初级测试程序：

简单测试程序							
内存地址	汇编指令	机器代码	AC	R1	R2	R3	说明
0000	LAD(0010)->R1	9100	0000	0008	0000	0000	将地址0010的数据存入R1
0001		0010					
0002	LAD(0011)->R2	9200	0000	0008	0005	0000	将地址0011的数据存入R2
0003		0011					
0004	ADD R1	1400	0008	0008	0005	0000	AC+R1->AC
0005	SUB R2	2800	0003	0008	0005	0000	AC-R2->AC
0006	STO AC->(8000)	8000	0003	0008	0005	0000	将AC的数据存入地址8000
0007		8000					
0010		0008					数据
0011		0005					数据

终极测试程序覆盖 LAD、ADD、SUB、JC、STO、JMP、PUSH、CLA、POP、STP 等 10 条指令，功能齐全。增加了中断和跳转程序，测试整体硬件和指令设计的正确性。

程序设计包括诸多细节，包括 JC 的跳转判断，中断，压栈弹栈，I/O，各个分支均测试到，以保证完全性和正确性。

终极测试程序：

测试程序							
内存地址	汇编指令	机器代码	AC	R1	R2	R3	说明
0000	LAD(4000)->R1	9100	0000	A000	0000	0000	将地址4000的数据存入R1 (4000存放的初值为A000)
0001		4000					
0002	LAD(4001)->R2	9200	0000	A000	4000	0000	将地址4001的数据存入R2 (4001存放的初值为4000)
0003		4001					
0004	ADD R1	1400	A000 0000	A000 A000	4000 6000	0000 0000	AC+R1->AC (分别对应2轮访问该指令)
0005	JC 07	4007	A000	A000	4000	0000	通过C判断是否跳转至000D
0006	SUB R2	2800	6000	A000	4000	0000	AC-R2->AC
0007	STO AC->(8000)	8000	6000	A000	4000	0000	将AC的数据 存入地址8000
0008		8000					
0009	LAD(8000)->R2	9200	6000	A000	6000	0000	将地址8000的数据存入R2
000A		8000					
000B	JMP(0004)	A000	6000	A000	6000	0000	跳转至0004
000C		0004					
000D	LAD(4002)->R3	9300	0000	A000	6000	2000	将地址4002的数据存入R3
000E		4002					
000F	PUSH AC	6081	0000	A000	6000	2000	AC数据入栈
0010	CLA	3000	0000	A000	6000	2000	清零AC
0011	POP AC	7000	0000	A000	6000	2000	栈顶数据进入AC
0012	ADD R3	1C00	X000	A000	6000	2000	AC+R3->AC(多次AC+=2)
0013	JC 02	4002	X000	A000	6000	2000	通过C判断是否跳转之0016
0014	JMP(000F)	A000	X000	A000	6000	2000	跳转至000F
0015		000F					
0016	STP	E000					停机

内存地址	汇编指令	机器代码	说明
中断子程序A			
3000	SUB R3	2C00	AC-R3->AC
3001	SUB R3	2C00	AC-R3->AC
3002	EI	B000	中断允许
3003	IRET	5000	中断返回
中断子程序B			
3100	IN	C000	从输入区输入数据
3101		F000	
3102	ADD R3	1C00	AC+R3->AC
3103	OUT	D000	输出数据
3104		F000	
3105	EI	B000	中断允许
3106	IRET	5000	中断返回

本测试程序包含 2 个中断子程序，子程序 A 验证基础中断功能，子程序 B 验证 IN 和 OUT 指令。

测试结果：

对终极程序进行单拍和连续性测试。
时序电路、微程序控制器、数据通路、中断程序、I/O 均运行正确。
显示区各数据均显示正确。

测试程序运行完全正确。

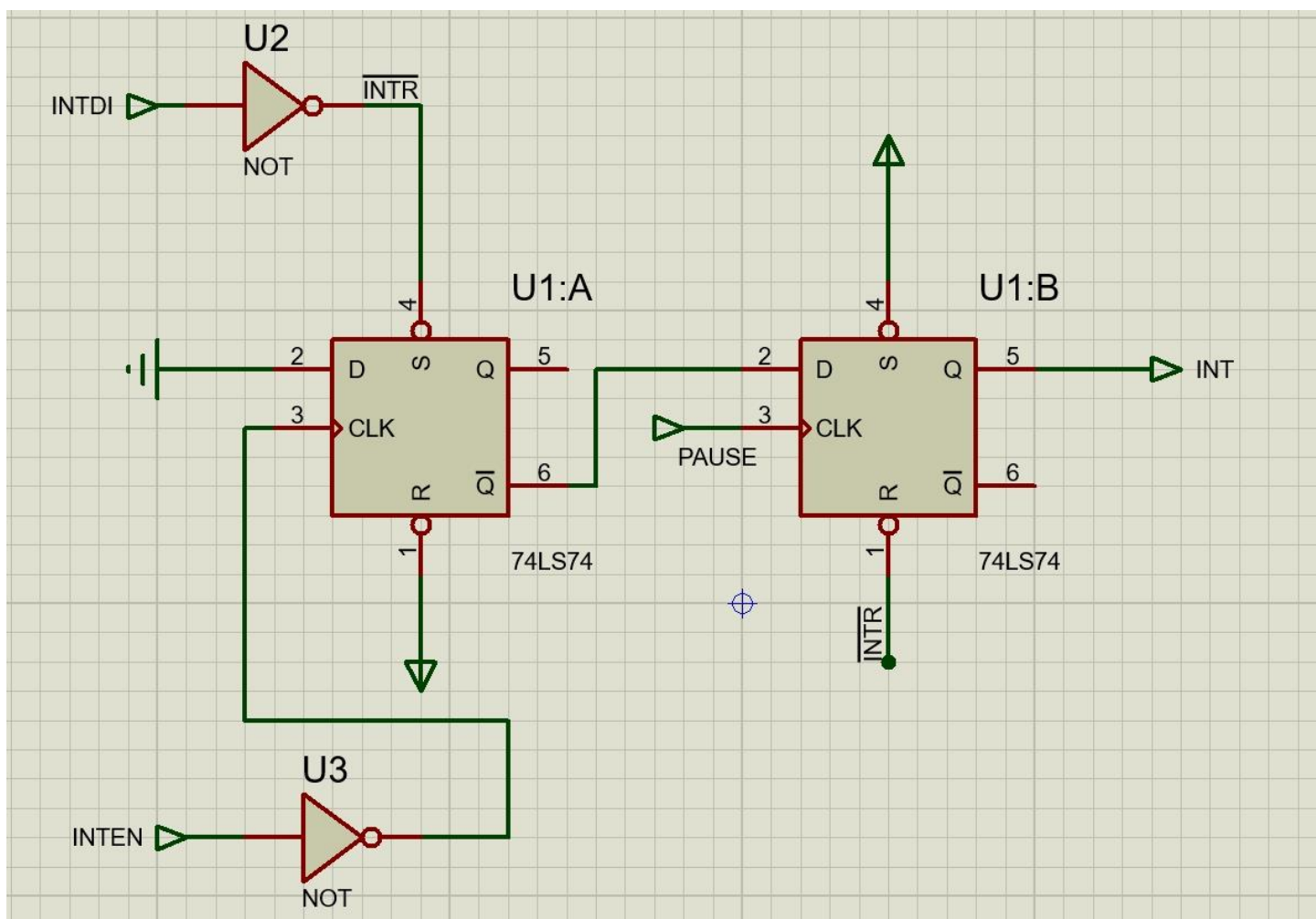
5. 程序正确性调试

遇到的问题：

1. JC 指令判断时，进位输出 C 不稳定。
2. PC 时序出现问题，导致跳转 PCADD 出错。
3. 中断 STOP 无法持续。
4. STOP 指令无法停机。

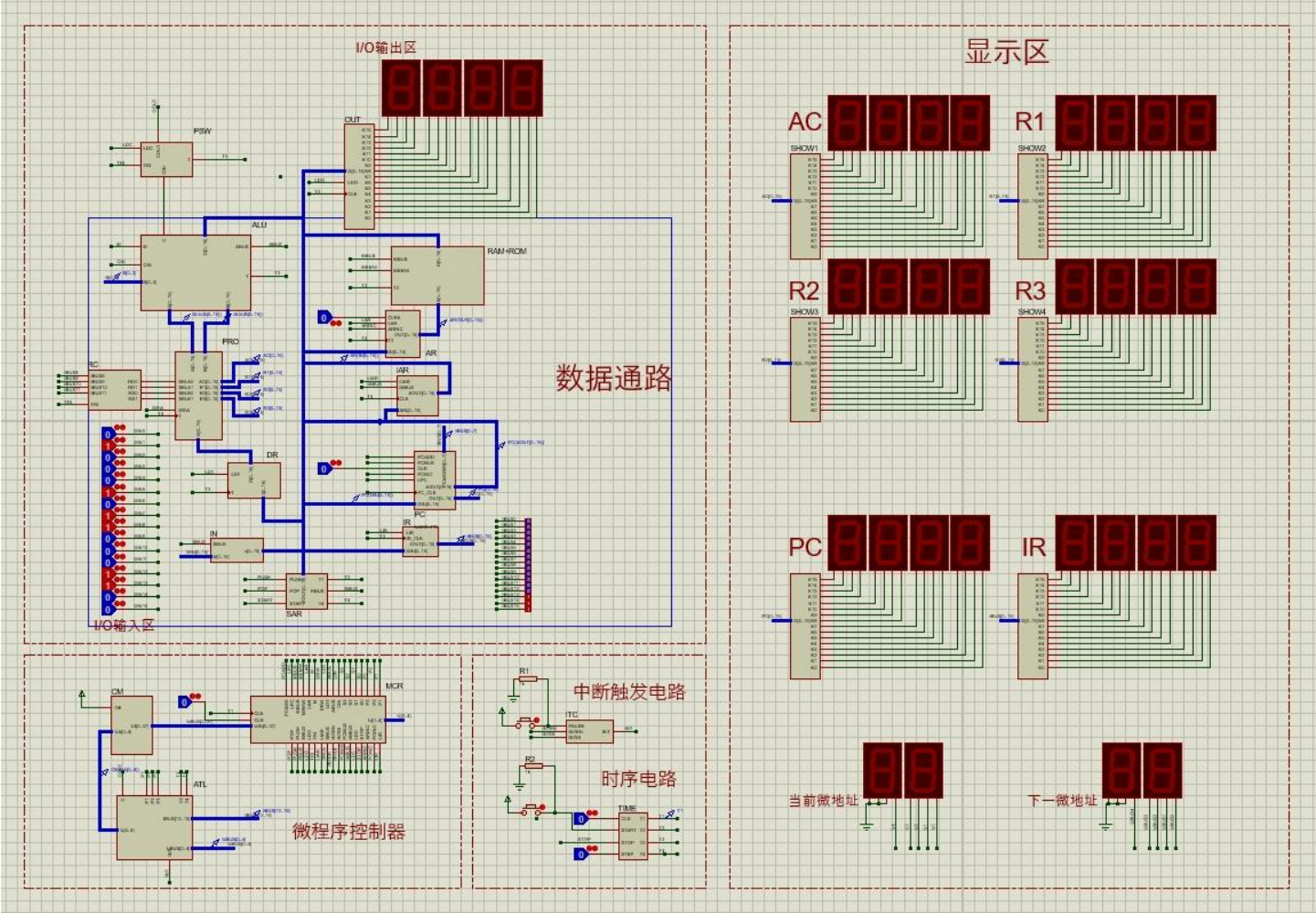
改进措施：

1. 调整 ALU 中进位输出。
 2. 调整脉冲信号顺序。
 3. 增加逻辑电路，实现功能：识别第一次 STOP 信号并保持 STOP 有效，且在 START 有效时，STOP 置为 0。
- （逻辑电路如下）



6. 中间信号显示及布局设计

整体结构布局：



整个硬件结构分为四个部分：微程序控制器、时序电路、数据通路、显示区。

显示区通过晶体管显示。

仿照 TEC-8 电路板设计，显示区时刻显示 AC、R1、R2、R3、PC、IR、当前微地址、下一微地址。用于观察程序运行进度、微指令转移进程和程序运行正确性。

7. 微指令译码逻辑设计

主要元件为 MCR。通过从 CM 中取垂直微指令，将 38 位微指令通过寄存器译码，译成水平微指令，同时输出 38 位微指令，对数据总线信号进行控制。

心得体会：

1. 本次 CPU 设计让我体会到了由整体到局部，层层细分的思想。在搭建 CPU 时，整个制作过程总体分为 4 个步骤：设计指令及微指令、构建硬件整体模块、为各模块细化建立子电路、仿真验证。

2. 自主设计微程序版 CPU 的过程中。建立清晰的计算机整机概念和内部机制，对计算机的基本组成、部件的设计、部件间的连接、微程序控制器的设计、微指令和微程序的编制与调试等过程有更深入的了解，加深对理论课程的理解。

3. 掌握了 ALU、RAM、ROM、PRO、AR、IAR、PC、IR、DR 等各个元件的功能以及内部构造。

4. 能够自主设计指令系统，微程序控制器。

5. 掌握数据在数据通路中流动和控制信号有效的顺序。

6. 让我深深体会了调试过程中自主发现问题并修正的方法。首先将各模块分开，检查各模块功能的正确性，再将各模块连载起来检查总线传输的正确性，最后验证运算的正确性。

总体而言通过本次实验，思路，我认识到在 CPU 设计过程当中，电路的整体性分析是至关重要的，掌握了从整体到局部的层层细分模块化的思想。知识上，运用了课本上对于指令集和微指令的设计，更加清晰的理解。同时，操作上，我熟悉了 Proteus 这个软件的使用，掌握了父子图的制作方法，了解了各个元件功能和构造。

更加重要的是，原本在书本上的元件，实际使用之后愈加加深了我对计算机计算方式以及硬件组成的理解。计算机不再是书本上图片和模型，我自己对计算机组成有了更深刻的体会，很好地掌握了计算机组成原理！