简易 4 位 CPU

黎鸿儒 李岳 杨润卓 王世彩 何振方

2020 年 6 月 9 日

目录

1 电路综述

本小组将本学期学习的《数字电子技术》与《计算机组成原理》两门课内容结合，使用 74 系列芯片及门电路搭建了一个能实现简单指令集的 4 位 CPU。可实现的指令集包括在寄存器，RAM 存储单元，立即操作数之间的 MOV 指令，ADD 指令，IN 指令，OUT 指令，还包括与程序跳转相关的 JMP 及 JNC 指令以及子程序的调用 CALL 指令与返回主函数指令 RET。

详细电路图及指令集真值表将以附件形式上传。电路文件名：cpu_4bit.pdf，cpu_4bit.ms14，所用到的仿真软件为 Multisim 14.0。指令集真值表：Instruction.pdf。

1.1 总体结构图

电路模块结构图如下，图中黑色网络为控制信号总线，蓝色网络为 4 位数据总线，紫色网络为指令指针计数器，绿色部分为外部控制信号（时钟与清零）。

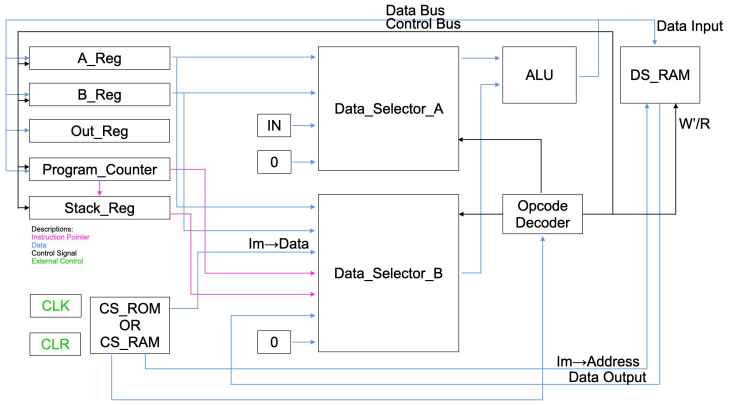

图 1: 总体结构

2 模块分析

2.1 时钟与清零模块

时钟信号模块有两种输出模式，一种是频率可选的自动方波源（用于写好程序自动执行），另一种是对应汇编中 debug 操作的手动源（用于对程序进行调试），两种模式可以通过单刀双掷开关控制。

其中频率可选的自动方波源用 NE555 芯片实现：

$$f = \frac{1}{T} = \frac{1}{(R_1 + 2R_2)C \ln 2} \tag{1}$$

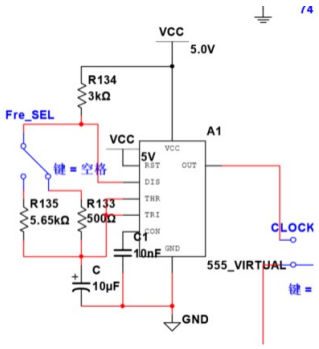

图 2: NE555 自动时钟源

手动时钟与清零信号结构相同，都采用了 RS 触发器来消抖，实物制作可以用按键来代替，更加方便。

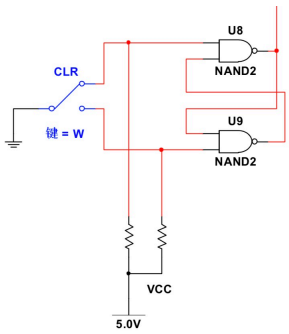

图 3: 手动时钟及清零信号

2.2 A 寄存器与 B 寄存器

A 寄存器与 B 寄存器通过 74LS161 芯片实现。用到 74LS161 芯片的锁存及置数功能，由于不需要计数功能故其中 ENP 与 ENT 两个使能端接无效电平。

74LS161 芯片的四位输入接入数据总线（图中蓝色线），绿色线为外部控制信号（时钟与清零）。

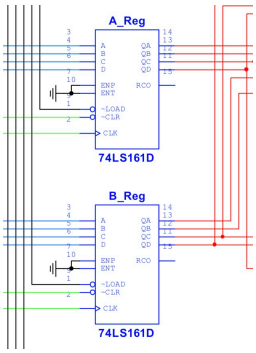

图 4: A 寄存器与 B 寄存器

A 或 B 寄存器的主要功能是目的操作数及源操作数的寻址。作为源操作数时，A 或 B 寄存器的 4 位输出分别和后面的两个双四选一数据选择器及四个单八选一数据选择器相连，当数据选择器的地址端得到相应的控制信号时会将 A 或 B 寄存器的数据送到 ALU 进行运算；作为目的操作数时，指令执行后 ALU 的运算结果会将送到数据总线，A 或 B 寄存器通过控制信号（74LS161 芯片中的 LOAD‘端）来接收数据。

2.3 输出寄存器

OUT 寄存器的功能是用于将数据输出到 CPU 外部（可理解为目的操作数的一种寻址方式，但计算机原理中不这么称呼这种功能），仿真时通过 LED 灯作为 CPU 外部，便于观察结果。该寄存器也通过 74LS161 芯片实现，用到 74LS161 芯片的锁存及置数功能，由于不需要计数功能故其中 ENP 与 ENT 两个使能端接无效电平。74LS161 芯片的四位输入接入数据总线（图中蓝色线），绿色线为外部控制信号（时钟与清零）。

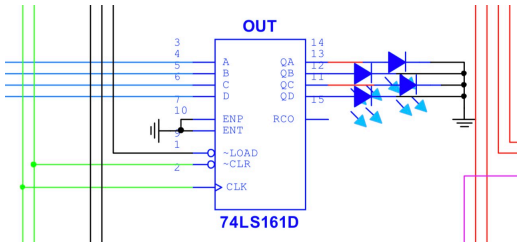

图 5: 输出寄存器

2.4 PC 寄存器与堆栈寄存器

Program Counter 指令指针寄存器用来控制取指操作，由于设置为最多编写 16 条指令，故需要指令指针寄存器能寻址 16 个代码单元，故依然采用 74LS161 芯片，此时用到的是 74LS161 的计数功能（顺序执行指令），置数功能（用于 CALL RET JMP JNC 等指令），故需要将 74LS161 的 ENT 与 ENP 使能端接高电平。在未遇到跳转指令时，置数控制无效，74LS161 正常计数；当跳转指令执行时，数据总线把跳转地址通过数据总线送给 PC 寄存器，控制信号控制 PC 的置数有效，实现跳转。

堆栈寄存器：实际应用中的堆栈段应该是一个“段”，也即能存储许多数据，并有着和其他存储模块不一样的数据结构。但在本项目中唯二用到堆栈段的功能是 CALL 子程序跳转与 RET 返回主程序指令，未设置 POP 指令与 PUSH 指令，且未设计递归调用子程序的功能，故唯一需要存储的就是 PC 寄存器的值，在子程序调用之前保护现场，供子程序执行完毕返回主程序时提供跳转地址，所以本项目中并未采用 RAM 这样有较多存储单元的模块，而是使用了有锁存功能的 74LS161。保护现场的功能通过让堆栈寄存器存储 CALL 指令执行前 PC 寄存器 +1 的值来实现，即图中在指令指针线（紫色）中引出一条支路与 1（设置低位进位为 1）做加法，这样在 RET 指令执行时，堆栈寄存器将数据（子程序调用前的指令指针 +1）送到数据总线，PC 的控制信号控制置数端有效，PC 的输出变为堆栈寄存器输出的值，实现子程序调用完毕的返回。

2.5 加法单元与进位标志位

本 CPU 的核心功能就是加法运算，74LS283 四位输入满足本项目的要求，故采用 74LS283 芯片作为运算单元，两个加数分别来源于两组数据选

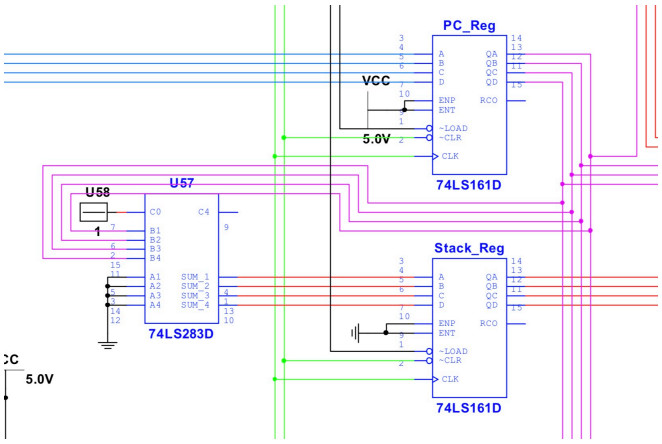

图 6: PC 寄存器与堆栈寄存器

择器，运算结果送往数据总线（图中蓝色线），进位输出接 D 触发器，用来存储进位标志，其中 D 触发器的时钟端与清零端均接入电路的外部控制信号线（电路总体时钟源与清零信号，图中为绿色线）。本项目中唯一用到进位标志位的指令是 JNC（Jump when not carry），即“进位标志为 0 时，指令跳转”所以没有将 D 触发器的 Q 接入电路，而是接入了 Q’，该信号会作为控制单元的信号输入。

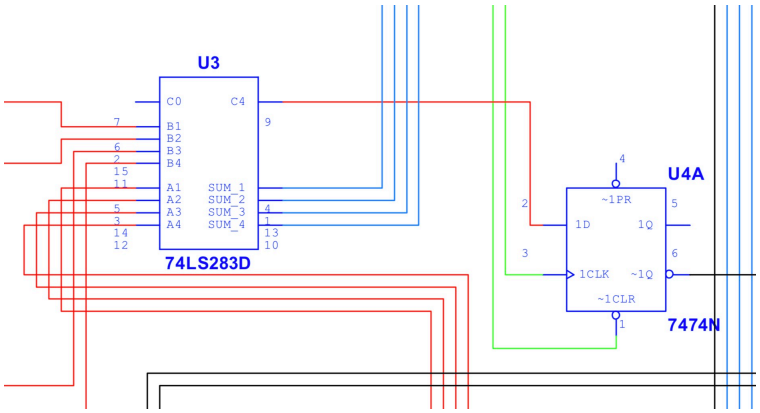

图 7: 加法单元与进位标志位

2.6 数据选择器组

CPU 中有两组数据选择器，分别是两个双四选一数据选择器 74LS153，以及四个单八选一数据选择器 74LS151，这两组数据选择器唯一的功能是选择 4 位数据作为 ALU 加法运算单元的加数，其中 74LS153 数据选择器组选择的四位数据为：

- A 寄存器
- B 寄存器
- 外部输入数据：该部分用四位拨码开关与下拉电阻实现数据输入
- 0

74LS151 数据选择器组选择的四位数据为：

- A 寄存器
- B 寄存器
- 输入代码的低四位，即立即操作数 IM
- PC 寄存器
- 堆栈寄存器
- DS 段数据
- 0

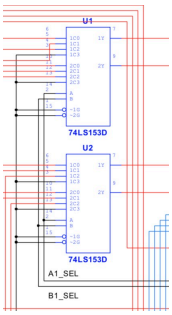

图 8: 74153 数据选择器组

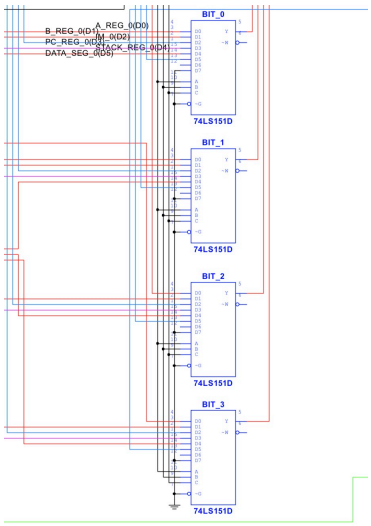

图 9: 74151 数据选择器组

以 74LS153 数据选择器组说明数据选择原理：两组双四选一数据选择器有 4 组共 16 个数据输入端，将同一数据（四位）分别接入四组输入端的相同地址，这样只需要把两个 74LS153 的 AB 地址端连接在一起，当控制信号到来时，两个 74LS153 的四个输出端就输出了同一个数据的四位，实现四位数据的选择。八选一数据选择器组同理，不同之处在于一共可以选择 8 个 4 位数据（图中只用到了 7 个）。

2.7 ROM 代码存储模块

ROM 存储模块是 CPU 的 Code Segment 代码段，由一个 4-16 线译码器和十六组开关组成，开关组并连在输出线上。译码器的输入与 PC 寄存器相连，对指令指针进行译码，输出为高电平译中。PC 寄存器在计数时，4-16 线译码器的输出随之变化，选择不同的开关组。

以 PC 送到译码器的值为 0010 时说明 ROM 代码模块原理：当 PC 输出为 0010 时，译码器的输出 O_2 为高电平，其余输出为低电平。O_2 所连接的开关组中的二极管有可能导通，其余输出连接的开关组的二极管不导通。由于有下拉电阻的存在，这样当 O_2 所连接的开关组中一根线上的开关闭合时，对应的输出线上为高电平，当开关断开时，对应的输出线上为低电平。

这样只需要在外部选择开关的通断即可实现代码编写，这种开关阵列

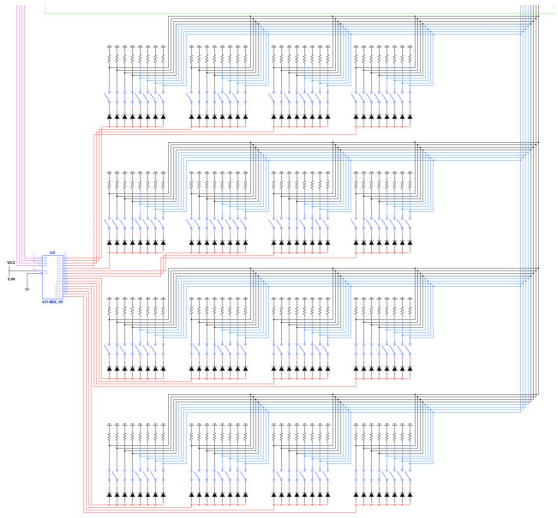

图 10: ROM 代码存储模块

存储模块相比于集成的 2114 等 RAM 芯片更加直观，可以直接观察到输入的机器码。

2.8 RAM 代码存储模块

除了开关阵列作为代码输入以外，CPU 也支持集成芯片式的 RAM 代码存储，结构如下：

RAM 的 8 个 I/O 端与 8 组控制逻辑相反的三态门相连，三态门的控制端同时与 RAM 芯片的 WE‘（写允许）与 OE’（输出允许）端相连，来控制读写逻辑具体读写原理如下：

- 代码写入：当控制读写的信号源（图中 U56）为 0 时，8 组三态门的上面的三态门均导通，通过右边的 8 组开关代码输入数据，随后信号源变为 1，地址变更，输入下一组代码，在用 RAM 写入代码期间，电路中的其他电路均无时钟接入，保证在信号源变为 1 时电路状态不会随之改变。

- 代码读出：当控制读写的信号源时钟保持为 1 时，8 组三态门下面的三态门均导通，RAM 将内部存储单元的数据送往代码输出线，地址变更，输出下一组代码，在用 RAM 读出代码期间，电路中其他电路均

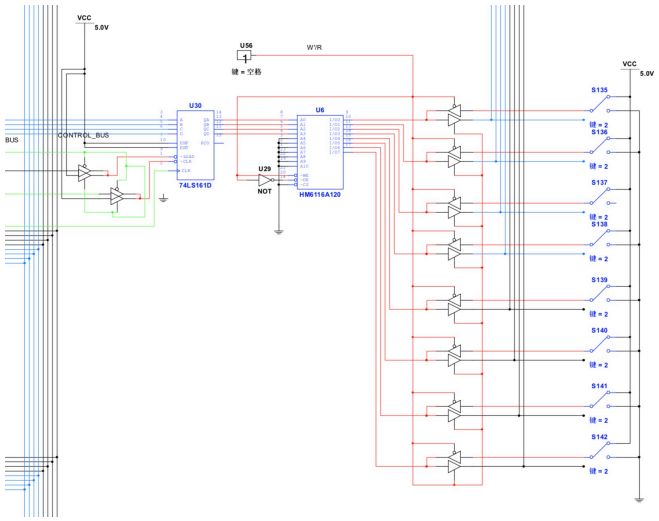

图 11: RAM 代码存储模块

接入时钟。

2.9 数据存储模块

数据存储模块 Data Segment 用于存储 CPU 中的数据，相比 AB 具有锁存功能的寄存器而言，存储模块可以存储更多数据，CPU 中的存储模块采用 RAM 芯片来实现，结构如图 12所示：

读写原理如下：内部读写控制连接方式同 RAM 代码存储模块，由四组控制逻辑相反的三态门与 RAM 芯片的 WE'（写允许）与 OE'（输出允许）端相连，来控制电路读写。与 RAM 代码存储模块不同的是，由于需要存储的数据是 4 位的，故多余的 4 位接地，且该 RAM 芯片的地址端来源于代码段的低四位（即代码中立即操作数 IM），此时代表 CPU 需要向 IM 对应地址的数据进行读出或写入操作。另外，读写信号的控制由控制译码单元来控制读写状态。

2.10 控制译码模块

控制译码单元是控制电路中所有模块状态的核心部分，图 13是控制译码模块的部分电路，该模块是组合逻辑电路，由与门非门或门组成。

该部分的输入包括代码输出线的高四位（即代码中的控制信号部分）及

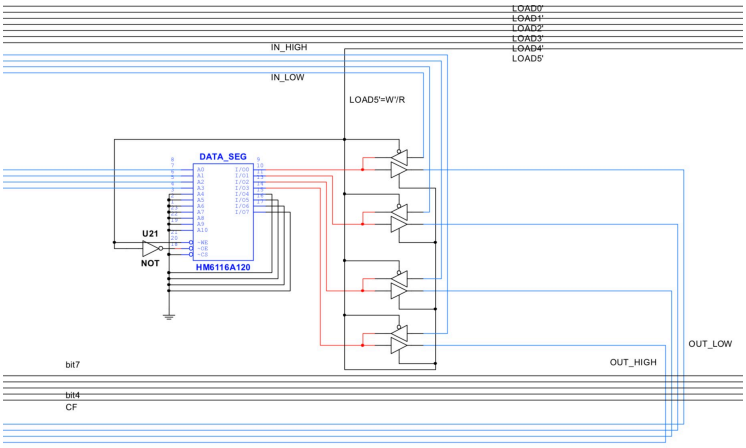

图 12: 数据存储模块

进位标志位组成，输出包含两个数据选择器组的地址线，A 寄存器，B 寄存器，OUT 寄存器，PC 寄存器，堆栈寄存器及数据存储模块的读写信号在内共 11 个控制信号。门电路的搭建是根据编写的指令集建立的真值表连接的，对每一个输出进行卡诺图化简得到对应的门电路，真值表详见附件 Instruction.dpf，此处略去卡诺图化简过程。

3 总线分析

3.1 数据线

数据线包括两大部分：数据总线及代码段低四位的立即数网络。在电路图中均用蓝色标出。

3.1.1 数据总线

数据总线的输入是 ALU 的加法输出，输出是 A 寄存器，B 寄存器，OUT 寄存器，PC 寄存器及数据存储模块，用于把 ALU 的运算送往各个部分，但相连但各个部分是否接收数据，哪部分接受数据则由控制信号（置数信号与 RAM 的读写信号）决定。该线的用途是传输数据。

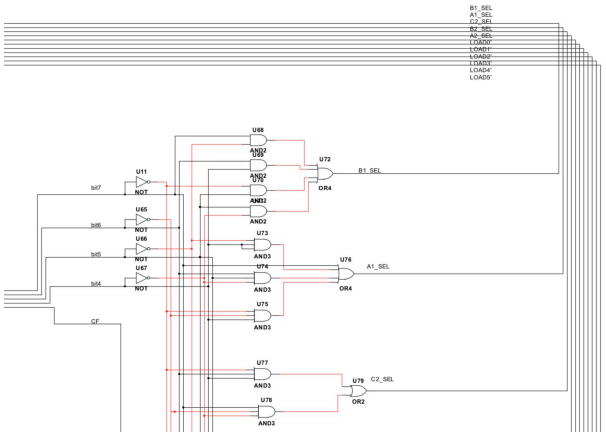

图 13: 控制译码模块

3.1.2 立即数网络

立即数网络的输入是代码的低四位，并将立即数送往八选一数据选择器组的 D_2 地址，同时还是数据存储模块的地址输入，用来查找数据段地址为 IM 的存储单元。该线的用途是传输立即数到数据选择器及数据存储模块的寻址。

3.2 控制信号线

控制信号线在电路图中用黑色标出，核心部分是控制译码模块。总线的输入包括代码的高四位及进位标志位，总线的输出包括 74LS153 数据选择器组的地址端 B_{1_SEL}, A_{1_SEL}，74LS151 数据选择器组的地址端 $C_{2\_SEL}, B_{2\_SEL}, A_{2\_SEL}$，A 寄存器的置数信号 $LOAD'_0$，B 寄存器的置数信号 $LOAD'_1$，OUT 寄存器的置数信号 $LOAD'_2$，PC 寄存器的置数信号 $LOAD'_3$，堆栈寄存器的置数信号 $LOAD'_4$，数据存储模块的读写信号 $LOAD'_5(W'/R)$。该线的用途是在指令执行期间控制不同模块的状态。

3.3 指令指针线

指令指针线在图中用紫色线标出，输入 PC 寄存器，输出为堆栈寄存器前置的 74LS283 加法器（用于 CALL 时的 PC+1）与 74LS151 八选一数据选择器组的 D_4。该线的用途是进行顺序执行指令时为取指部分提供代码地

址，以及在执行 JMP，JNC，CALL，RET 等指令执行时进行跳转。

3.4 外部源

外部源线路包括 CPU 的总时钟信号与总清零信号，电路图中用绿色线标出，该部分与 A 寄存器，B 寄存器，OUT 寄存器，PC 寄存器，堆栈寄存器及进位标志位（D 触发器）的时钟与清零信号相连，时钟或清零信号到来时这些寄存器发生动作或进行清零。

3.5 代码输出选择

电路中代码段输出线的上的数据来源通过八组控制逻辑相反的三态门来实现，每组三态门的两个输入来源于 ROM 和 RAM 代码存储模块。蓝色线代表代码的低四位，为立即数 IM；黑色线代表代码的高四位用于生成控制逻辑。

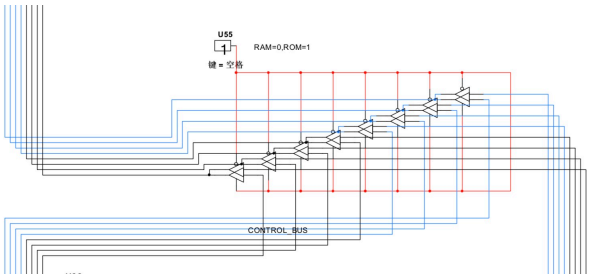

图 14: 输出选择

4 指令集分析

4.1 指令集表

指令集表如图 15所示（附件中也有指令集文件），其中指令高四位按顺序赋值，低四位为 0000 或立即操作数。JNC 指令由于进位标志位的状态不同而分成两种。

Instruction	bit7-bit4	bit3-bit0	C	B1_SEL	A1_SEL	C2_SEL	B2_SEL	A2_SEL	#LOAD0	#LOAD1	#LOAD2	#LOAD3	#LOAD4	#LOAD5
ADD A,IM	0000	IM	X	L	L	L	H	L	L	H	H	H	H	H
MOV A,B	0001	0000	X	L	H	L	H	L	L	H	H	H	H	H
IN A	0010	0000	X	H	L	L	H	L	L	H	H	H	H	H
MOV A,IM	0011	IM	X	H	H	L	H	L	L	H	H	H	H	H
ADD A,B	0100	0000	X	L	L	L	L	H	L	H	H	H	H	H
MOV A,[IM]	0101	IM	X	H	H	H	L	H	L	H	H	H	H	H
MOV [IM],A	0110	IM	X	H	H	L	L	L	H	H	H	H	H	L
IN [IM]	0111	IM	X	H	L	H	H	H	H	H	H	H	H	L
OUT [IM]	1000	IM	X	H	H	H	L	H	H	H	L	H	H	H
CALL IM	1001	IM	X	H	H	L	H	L	H	H	H	L	L	H
RET	1010	0000	X	H	H	H	L	L	H	H	H	L	H	H
OUT B	1011	0000	X	L	H	L	H	L	H	H	L	H	H	H
OUT IM	1100	IM	X	H	H	L	H	L	H	H	L	H	H	H
JNC IM	1101	IM	L	H	H	L	H	L	H	H	H	L	H	H
JNC IM	1101	IM	H	X	X	X	X	X	H	H	H	H	H	H
JMP IM	1111	IM	X	H	H	L	H	L	H	H	H	L	H	H

图 15: 指令集

4.2 ADD A,IM

该指令是将代码低四位的立即操作数与 A 寄存器中的数据相加，并将结果送往 A 寄存器。

- 源操作数寻址方式：A 寄存器，代码低四位（注：在计算机原理课程中教授的源操作数只包括立即操作数，这里为了便于理解，将指令需要的所有数据都成为源操作数；在计算机原理课程中寻址方式也不是指数据存放位置，这里为了便于直观理解，将数据存放位置成为寻址方式，后续其他指令同理）
- 目的操作数寻址方式：A 寄存器

由上述分析可知：

- ALU 中的 B 加数通过 74LS153 选择 A 寄存器，则 $B_{1\_SEL} = L, A_{1\_SEL} = L$
- ALU 中的 A 加数通过 74LS151 选择立即操作数，则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往 A 寄存器，则 $LOAD'_0 = 0$，其余无效。

4.3 MOV A,B

该指令是将 B 寄存器的值送往 A 寄存器。

- 源操作数寻址方式：B 寄存器
- 目的操作数寻址方式：A 寄存器

由上述分析可知:

- ALU 中的 B 加数通过 74LS153 选择 B 寄存器,则 $B_{1\_SEL} = L, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择立即操作数（此时立即操作数为 0），则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往 A 寄存器，则 $LOAD'_0 = 0$，其余无效。

4.4 IN A

该指令是将输入端的数据送往 A 寄存器。

- 源操作数寻址方式：输入端
- 目的操作数寻址方式：A 寄存器
- ALU 中的 B 加数通过 74LS153 选择输入端,则 $B_{1\_SEL} = H, A_{1\_SEL} = L$
- ALU 中的 A 加数通过 74LS151 选择立即操作数（此时立即操作数为 0），则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往 A 寄存器，则 $LOAD'_0 = 0$，其余无效。

4.5 MOV A,IM

该指令是将代码的低四位立即操作数送往 A 寄存器。

- 源操作数寻址方式：代码低四位——立即操作数
- 目的操作数寻址方式：A 寄存器
- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择立即操作数（此时立即操作数为 0），则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往 A 寄存器，则 $LOAD'_0 = 0$，其余无效。

4.6 ADD A,B

该指令是将 A 寄存器与 B 寄存器的值相加，并将结果送往 A 寄存器。

- 源操作数寻址方式：A 寄存器，B 寄存器
- 目的操作数寻址方式：A 寄存器
- ALU 中的 B 加数通过 74LS153 选择 A 寄存器,则 $B_{1\_SEL} = L, A_{1\_SEL} = L$
- ALU 中的 A 加数通过 74LS151 选择 B 寄存器,则 $C_{2\_SEL} = L, B_{2\_SEL} = L, A_{2\_SEL} = H$
- 结果送往 A 寄存器，则 $LOAD'_0 = 0$，其余无效。

4.7 MOV A,[IM]

该指令是将数据存储模块中地址为 IM 的数据送往 A 寄存器。此时 IM 不再通过数据选择器作为加数之一，而是作为数据存储模块的地址输入。

- 源操作数寻址方式：数据存储模块
- 目的操作数寻址方式：A 寄存器
- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择数据存储模块，则 $C_{2\_SEL} = H, B_{2\_SEL} = L, A_{2\_SEL} = H$
- 结果送往 A 寄存器，则 $LOAD'_0 = 0$，其余无效。

4.8 MOV [IM],A

该指令是将 A 寄存器的数据送往数据存储模块中地址为 IM 的存储单元。此时 IM 不再通过数据选择器作为加数之一，而是作为数据存储模块的地址输入。

- 源操作数寻址方式：A 寄存器
- 目的操作数寻址方式：数据存储模块

- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择 A 寄存器，则 $C_{2\_SEL} = L, B_{2\_SEL} = L, A_{2\_SEL} = L$
- 结果送往数据存储模块，则 $LOAD'_5 = 0$，表示存储模块进入写模式，其余无效。

4.9 IN [IM]

该指令是将输入端的数据送往数据存储模块中地址为 IM 的存储单元。此时 IM 不再通过数据选择器作为加数之一，而是作为数据存储模块的地址输入。

- 源操作数寻址方式：输入端
- 目的操作数寻址方式：数据存储模块
- ALU 中的 B 加数通过 74LS153 选择输入端，则 $B_{1\_SEL} = H, A_{1\_SEL} = L$
- ALU 中的 A 加数通过 74LS151 选择”0”，则 $C_{2\_SEL} = H, B_{2\_SEL} = H, A_{2\_SEL} = H$
- 结果送往数据存储模块，则 $LOAD'_5 = 0$，表示存储模块进入写模式，其余无效。

4.10 OUT [IM]

该指令是将数据存储模块中地址为 IM 的数据送到 OUT 寄存器进行输出。此时 IM 不再通过数据选择器作为加数之一，而是作为数据存储模块的地址输入。

- 源操作数寻址方式：数据存储模块
- 目的操作数寻址方式：OUT 寄存器
- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$

- ALU 中的 A 加数通过 74LS151 选择数据存储模块，则 $C_{2\_SEL} = H, B_{2\_SEL} = L, A_{2\_SEL} = H$
- 结果送往输出寄存器，则 $LOAD'_2 = 0$，其余无效。

4.11 CALL IM

该指令是调用子程序指令，子程序的首地址为 IM，将 IM 送往 PC 寄存器即实现跳转。同时需要保护现场，所以同时需要将跳转前的地址 +1 保存送给堆栈寄存器。

- 源操作数寻址方式：代码段低四位——立即操作数；PC 寄存器
- 目的操作数寻址方式：PC 寄存器；堆栈寄存器
- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择立即操作数，则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- ALU 的结果送往 PC 寄存器实现跳转，则 $LOAD'_3 = 0$；PC 跳转前的数据加一后送往堆栈寄存器，则 $LOAD'_4 = 0$，其余无效。

4.12 RET

该指令是子程序调用完毕后返回到主程序跳转前的下一条指令，只需要把堆栈寄存器返回给 PC 寄存器即可。

- 源操作数寻址方式：堆栈寄存器
- 目的操作数寻址方式：PC 寄存器
- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择堆栈寄存器，则 $C_{2\_SEL} = H, B_{2\_SEL} = L, A_{2\_SEL} = L$
- 结果送往 PC 寄存器，则 $LOAD'_3 = 0$，其余无效。

4.13 OUT B

该指令是将 B 寄存器的值送到输出寄存器进行输出。

- 源操作数寻址方式：B 寄存器
- 目的操作数寻址方式：输出寄存器
- ALU 中的 B 加数通过 74LS153 选择 B 寄存器，则 $B_{1\_SEL} = L, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择立即操作数（此时为“0”），则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往输出寄存器，则 $LOAD'_2 = 0$，其余无效。

4.14 OUT IM

该指令是将立即操作数的值送到输出寄存器进行输出。

- 源操作数寻址方式：立即操作数
- 目的操作数寻址方式：输出寄存器
- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择立即操作数，则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往输出寄存器，则 $LOAD'_2 = 0$，其余无效。

4.15 JNC,C=0

JNC 指令是进位标志不为 0 时进行 PC 寄存器跳转到地址为 IM 的指令，该分支为 C=0。

- 源操作数寻址方式：立即操作数
- 目的操作数寻址方式：PC 寄存器

- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择立即操作数，则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往 PC 寄存器，则 $LOAD'_3 = 0$，其余无效。

4.16 JNC,C=1

JNC 指令是进位标志不为 0 时进行 PC 寄存器跳转到地址为 IM 的指令，该分支为 C=1，则电路不进行任何操作。

4.17 JMP

JMP 指令是无条件跳转指令，PC 寄存器跳转到地址为 IM 的指令。

- 源操作数寻址方式：立即操作数
- 目的操作数寻址方式：PC 寄存器
- ALU 中的 B 加数通过 74LS153 选择“0”，则 $B_{1\_SEL} = H, A_{1\_SEL} = H$
- ALU 中的 A 加数通过 74LS151 选择立即操作数，则 $C_{2\_SEL} = L, B_{2\_SEL} = H, A_{2\_SEL} = L$
- 结果送往 PC 寄存器，则 $LOAD'_3 = 0$，其余无效。

5 程序举例及分析

本章节将展示部分指令，包括 ADD，MOV，IN，OUT，CALL，RET，JMP 指令，还包括存储器寻址方式的演示。

5.1 OUT 寄存器 LED 循环闪烁程序

该程序的代码为：

- 0000: OUT 1000

- 0001: OUT 0100
- 0010: OUT 0010
- 0011: OUT 0001
- 0100: JMP 0000

在 ROM 开关阵列编写指令如图 16所示，时钟信号选择手动信号如图 17所示，代码端输出选择 ROM 阵列，输出线三态门控制端置 1，如图 18

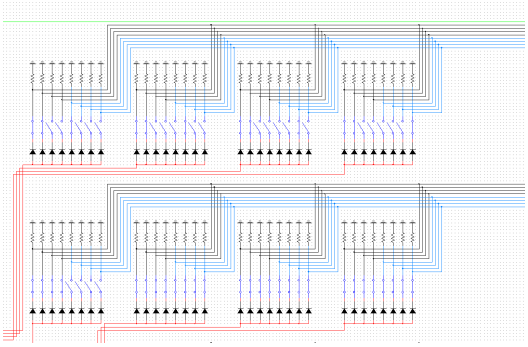

图 16: 编写指令

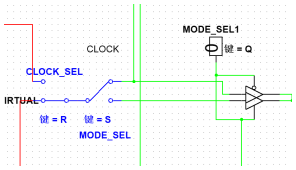

图 17: 手动时钟信号

做好以上准备工作后开始仿真，PC 寄存器开始从 0000 开始计数，如图 19。

此时由于上升沿未到，数据已经送到 OUT 寄存器的输入端，但还未被输出。如图 20。

手动产生上升沿后 PC+1。如图 21。

此时 OUT 寄存器输出当前 PC 的上一条指令。OUT 寄存器此时的输入为 0100，输出为 1000。如图 22。

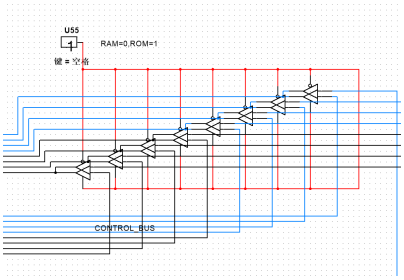

图 18: 三态门控制置 1

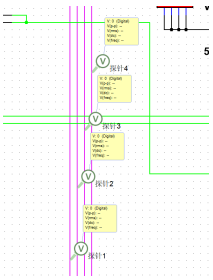

图 19: PC=0000

继续产生上升沿。OUT 寄存器此时的输入为 0010，输出为 0100。如图 23。

继续产生上升沿。OUT 寄存器此时的输入为 0001，输出为 0010。如图 24。

继续产生上升沿。此时 PC 取出的指令为 JMP 0000，OUT 寄存器执行该指令的上一条指令 OUT 0001，接收的输入为 0000。如图 25。

此时 PC 的输入为 0000。如图 26。

继续产生上升沿，PC 跳转到 0000，跳回到代码段首地址，这样就实现了用 OUT 寄存器使小灯循环闪烁。如图 27。

5.2 加法及存储器寻址演示

该程序的代码为：

- 0000: IN A;IN=0101
- 0001: ADD A,IM;IM=1000

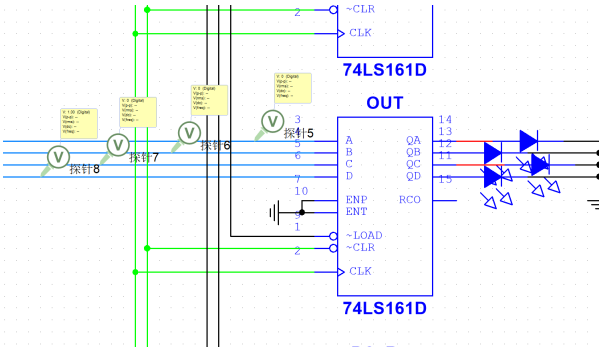

图 20: OUT 寄存器未接受输入

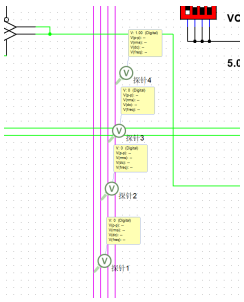

图 21: PC=0001

- 0010: MOV [IM],A;IM=0000

输入端为 0101。如图 28。

开始仿真，输入送到 A 寄存器。如图 29。

A=0101，IM=1000，作和结果为 1101，并送往 A 寄存器。如图 30。

将结果送往 Data Segment，RAM 内存中的首地址（IM=0000）中的内容为 0B，存储成功。如图 31。

5.3 CALL,RET,JMP 循环

该程序代码如下：

- 0000: OUT IM；IM=1111
- 0001: IN A;IN=0010
- 0010: CALL IM;IM=0100

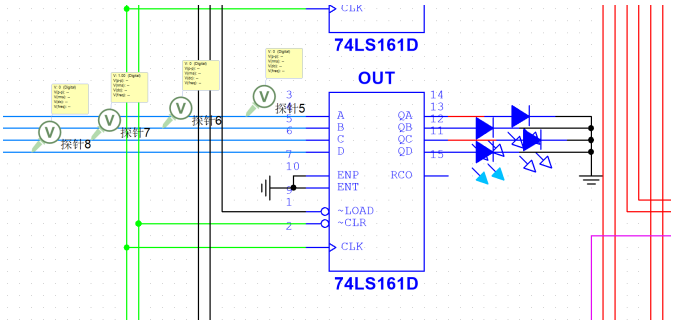

图 22: OUT 输出为 1000 输入为 0100

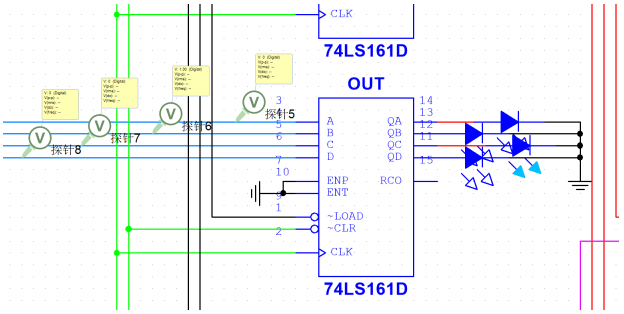

图 23: OUT 输出为 0100 输入为 0010

- 0011: JMP IM;IM=0000
- 0100: ADD A,IM;IM=1000
- 0101: OUT IM;IM=0000
- 0110: RET;RET 0011

在 ROM 中编写程序如下图 32:

执行 0000 指令，OUT 寄存器输出 1111，四个 LED 均亮。如图 33。

将输入送到 A 寄存器，此时输入为 0010，A 输出 0010。如图 34。

执行 CALL IM，PC 跳转到 0100，堆栈储存 PC 跳转前的值 +1，为 0011。如图 35。

子程序是 ADD A,IM，此时 IM=1000，A 输出为 1000+0010=1010。如图 36。

继续执行 OUT 0000。如图 37。

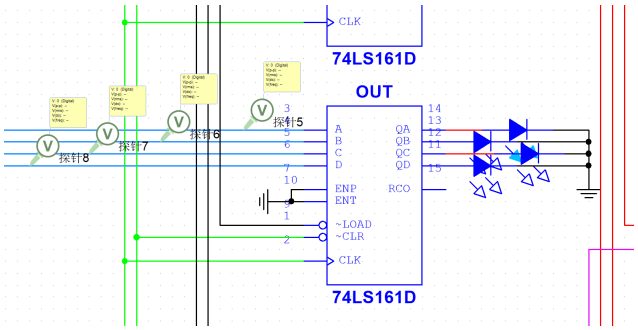

图 24: OUT 输出为 0010 输入为 0001

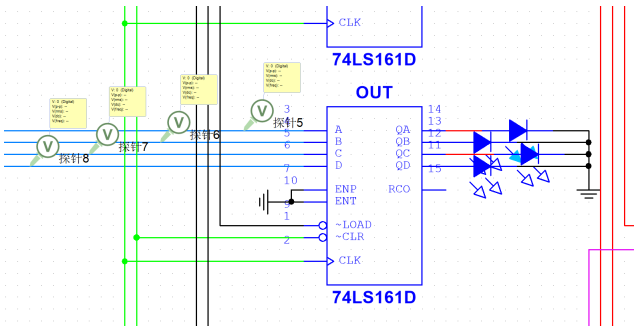

图 25: OUT 输出为 0001 输入为 0000

遇到返回指令 RET，此时 PC 跳转到堆栈寄存器存储的地址，PC=STACK=0011。如图 38。

0011 地址存储的指令为 JMP 0000。如图 39。

这样便跳转回 0000 地址的指令，实现了一次循环。

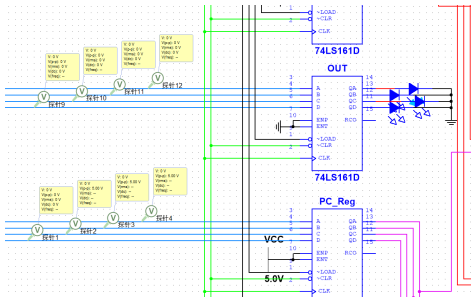

图 26: PC 输入为 0000，暂未跳转

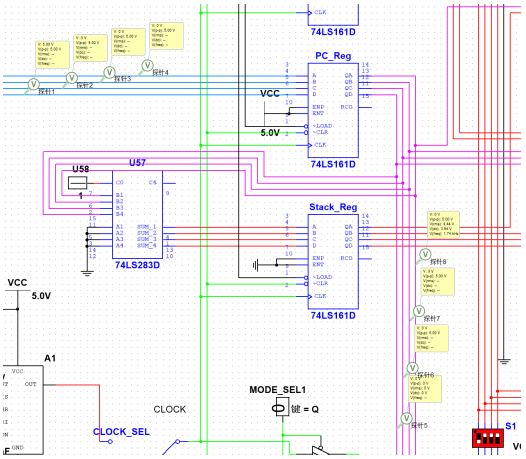

图 27: PC 跳转到 0000 实现循环

6 总结

本 CPU 的核心单元是 ALU 以及控制译码模块，每一个指令执行的过程都可以简化为如下的形式：

$$[OPS]_{SelectorA} + [OPS]_{SelectorB} \overset{ControlDecoder}{\longrightarrow} OPD \tag{2}$$

即通过两个控制逻辑在两个数据选择器中选择数据做和，将结果送往数据总线，并由控制逻辑选择接受数据总线数据上的数据的过程。

总体来说，本项目基本实现了 8086CPU 的功能，其他更复杂的功能可以基于本 CPU 来实现，同时本项目有许多不足之处以及一些可以改进的地方：

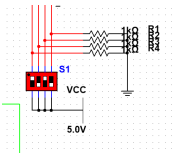

图 28: IN=0101

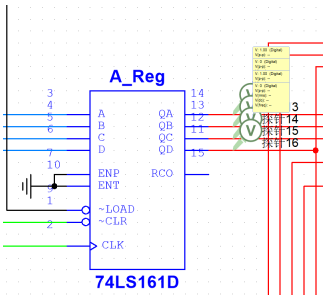

图 29: 输入送到 A 寄存器，A=0101

- 本项目的指令集只设计了 15 种指令（JNC 的两个分支算同一种），每一种都是独特的指令，种类齐全且没有重复功能的指令，但有一个问题在于我们设计的指令集中没有对 B 寄存器赋值的指令，可以从指令集的输出看到 $LOAD'_2$ 始终是高电平，即任何一个指令的运算结果都不会送到 B 寄存器，而 B 寄存器也没办法从外部赋值，致使 B 寄存器在 CPU 使用过程中输出一直为 0000。原因在于设计指令集时只考虑到设计出不同类型指令，没有考虑到需要给 B 寄存器赋值。

- 控制译码模块实际上可以选用 FPLA 进行编程，这样除了指令可以随时编写以外，指令集也可以随时编写，这样就可以解决了上面的问题，但我们在 Multisim14.0 软件中没有找到可以用的 FPLA，Multisim14.0 中的 FPGA 和 FPLA 都只是有封装而内部没有写函数，只能用来画电路而不能仿真。但做成实物时可以用 FPLA 来设计，这样可以实现的指令会非常多。

- CPU 中实际上有一个比较关键的问题是寄存器的置数是同步置数的，而数据存储模块相当于“异步”置数，这可能会造成在设计涉及到寄存器与存储模块的程序时造成预期以外的错误，目前想到的解决办法为可以将寄存器换为异步置数的控制逻辑（硬件改进），或者在涉及到存

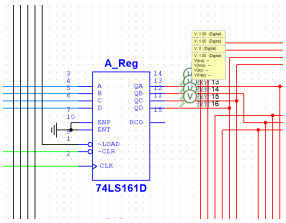

图 30: 结果送到 A 寄存器，A=1101

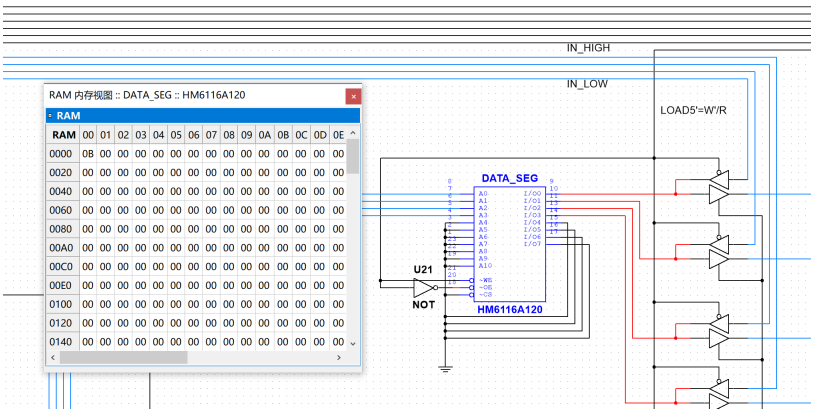

图 31: IM=0000,[IM]=0B, 数据存储成功

储模块的指令前后加入空指令，以保证存储模块与寄存器同步。

- 可以将堆栈寄存器改成堆栈 RAM 存储模块，这样就可以实现 POP X，PUSH X 以及子程序嵌套递归（不断保存跳转前的 PC 值，同时反序出栈）等功能。
- 该简易 CPU 可以改装成许多以用于许多实际的问题，本 CPU 的总输入端为四位拨码开关，输出为四个 LED 灯，可以将输入改为 A/D 模块，输出改为 D/A 模块，同时在代码段预先写好程序，然后封装，就可作为于诸如温湿度调节或报警等等问题的微控制器。
- 可以在电脑中设计编译器，这样就可以实现将 CPU 中的汇编逻辑变为高级语言，实现从硬件到软件的过程。

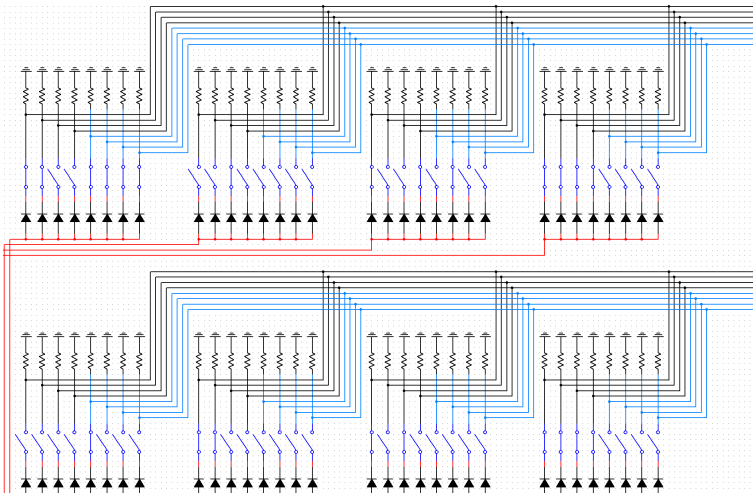

图 32: 第三段程序 ROM 编程

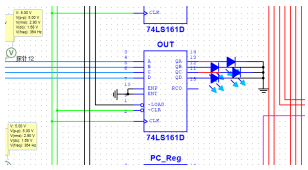

图 33: OUT=1111

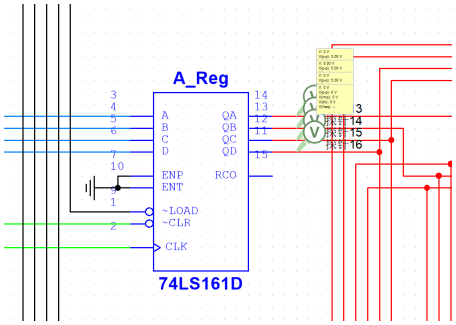

图 34: IN=0010，A=0010

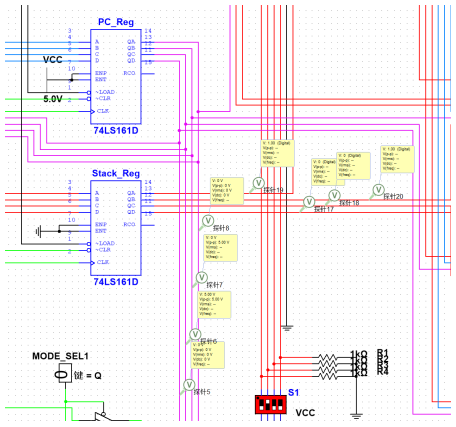

图 35: PC 跳转到 0100，堆栈存储 0011

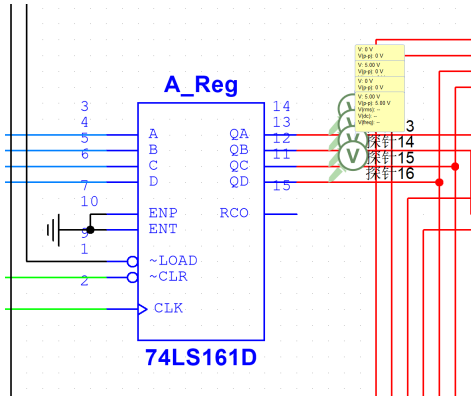

图 36: A=1010

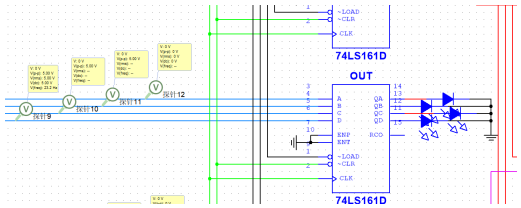

图 37: OUT=0000

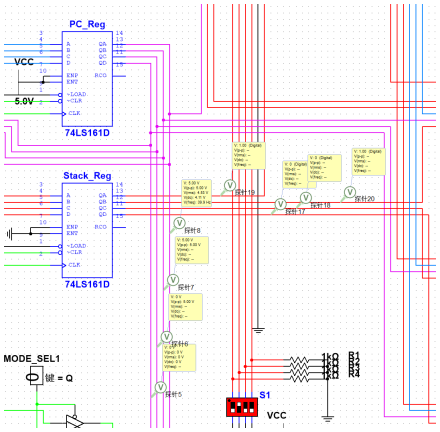

图 38: PC=STACK=0011

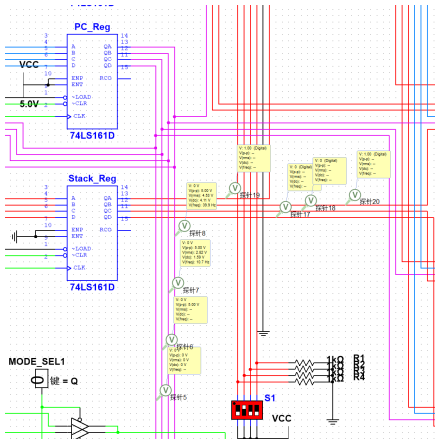

图 39: PC 跳转到 0000