

课程设计报告书

**基于微程序控制器的8位模型计算机设计**

**学 院 计算机科学与工程学院**

**专 业 计算机科学与技术**

**学生姓名 叶劲亨**

**学生学号 201830582180**

**指导教师 张齐**

**课程编号 045102071**

**课程学分 2.0**

**起始日期 2020/05/20**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

基于微程序控制器的8位模型计算机设计

## 一、选题背景

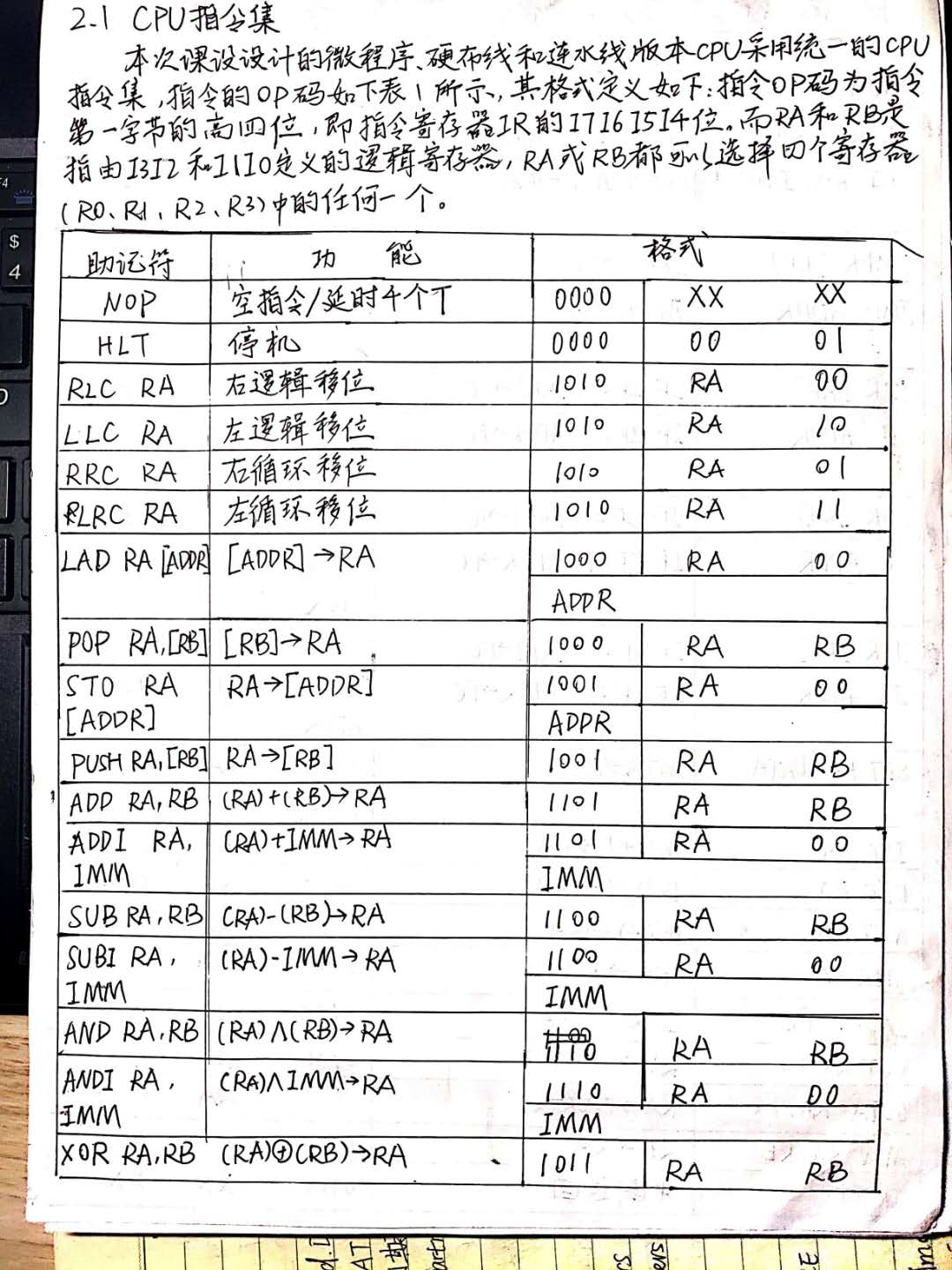
基于微程序控制器进行8位模型计算机的设计。

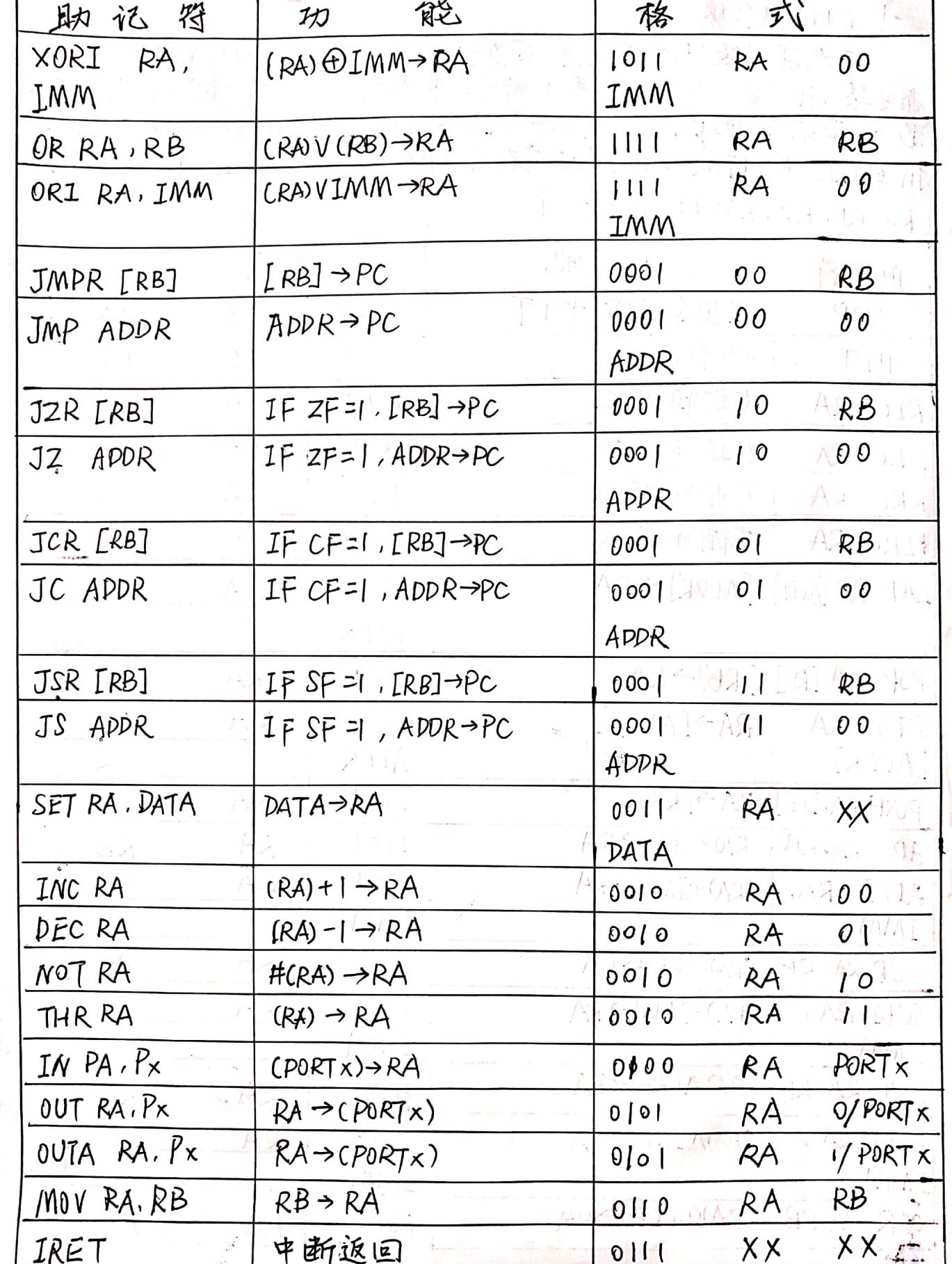
主要有以下四个方面的任务：

1. 根据所给出指令系统的要求对模板电路做进一步增删以形成最终的仿真电路，包括 CPU（运算器和控制器）、存储器（指令 ROM 和数据 RAM）、输入接口和输出接口。
2. 在已给出的指令系统的基础上，可以根据需求对指令系统加以扩展，但指令系统的指令条数不小于等于16条
3. 以最终的仿真电路和指令系统为基准
4. 设计微指令系统，基于模型机指令系统的用户程序设计

## 二、方案论证

### 2.1 CPU指令集





PS:

1. XX: 表示任意
2. RA\RB: 表示寄存器（实际在写程序的时候，不是用RA表示第一个寄存器，而是用R1表示一个寄存器，如MOV 3,R1;）
3. ADDR: 表示地址
4. DATA：表示数据，如 MOV 3,R1;，这里的3就是DATA
5. 指令PUSH/POP和JMP系列指令的JMPR/JCR/JZR/JER的RB只能是R1\R2\R3，不能是R0
6. IN和OUT指令的 汇编 要指明是对哪个外设端口操作，外设定义了四个端口 PORT0-3；其中PORT0和1 是双向端口即IN 和OUT都能操作的，PORT 2和3是单向输入端口只有IN指令。
7. IN指令的【IR1,IR0】指明操作的是哪个外设PORTx
8. OUT指令的IR0指明操作的外设PORTx，IR1指明传送外设的内容是数据还是地址（即IR1=0相当于ALE=0，向外设发送数据；IR1=1相当于ALE=1，向外设发送地址）。
9. HLT指令改为跟NOP指令相同OP码，然后IR0=1是 停机，IR0=0是NOP指令

### 2.2 微程序版CPU架构

如下图1所示，本实验的微程序版CPU由微程序控制器通路（CONTROLLER）、时序电路（CLOCK）及数据通路组成。数据通路包括：程序存储器ROM、数据存储器RAM及通用寄存器ROR3；IO接口；算术逻辑运算器（74LS181）及附带的移位寄存器（74LS194）；程序计数器（PC、ALU运算结果标志位寄存器（PSW）及其断点寄存器（BP\_PC、 BP\_PSW）.数据通路的所有部件都共同挂在一条位系统总线BUS上。

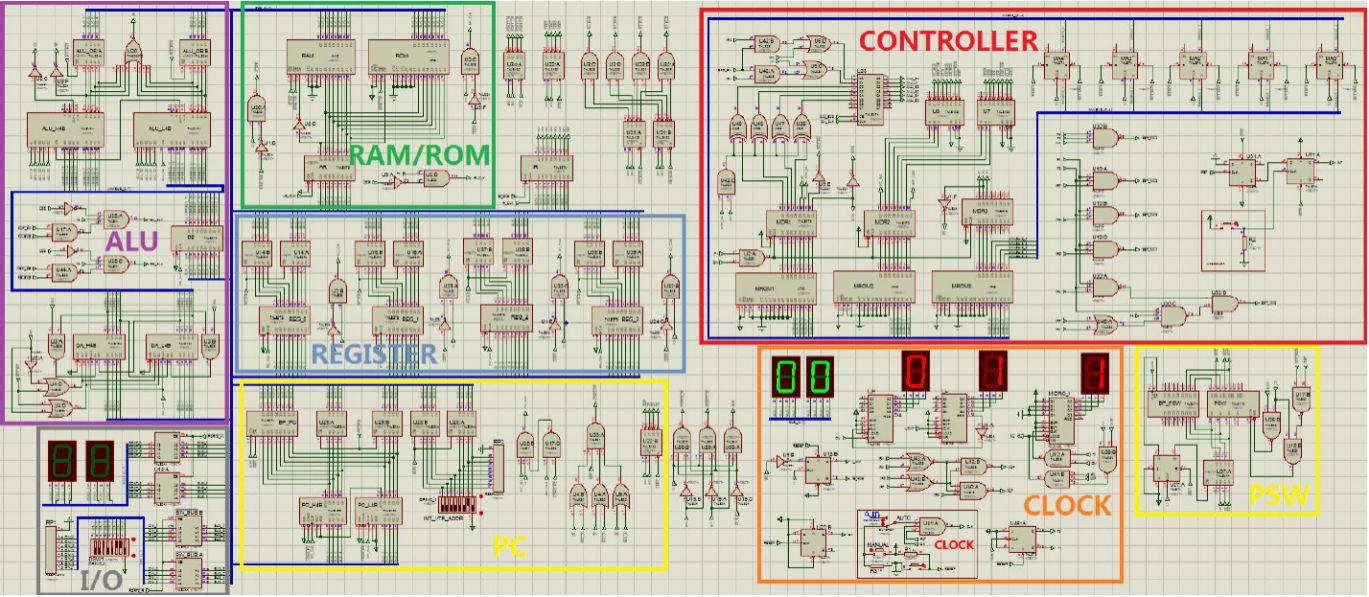
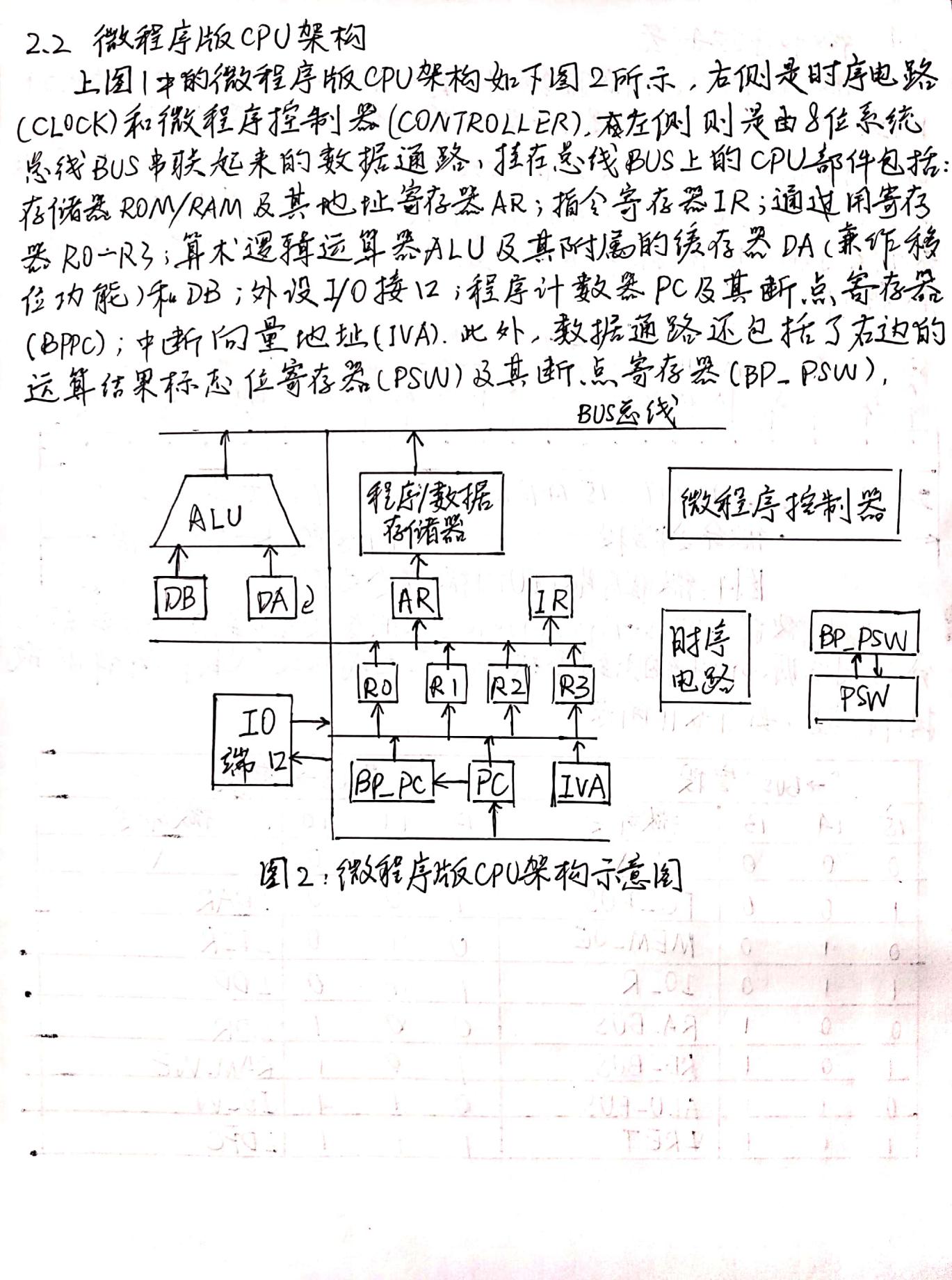


图1：微程序版CPU电路图



### 2.3 时序电路

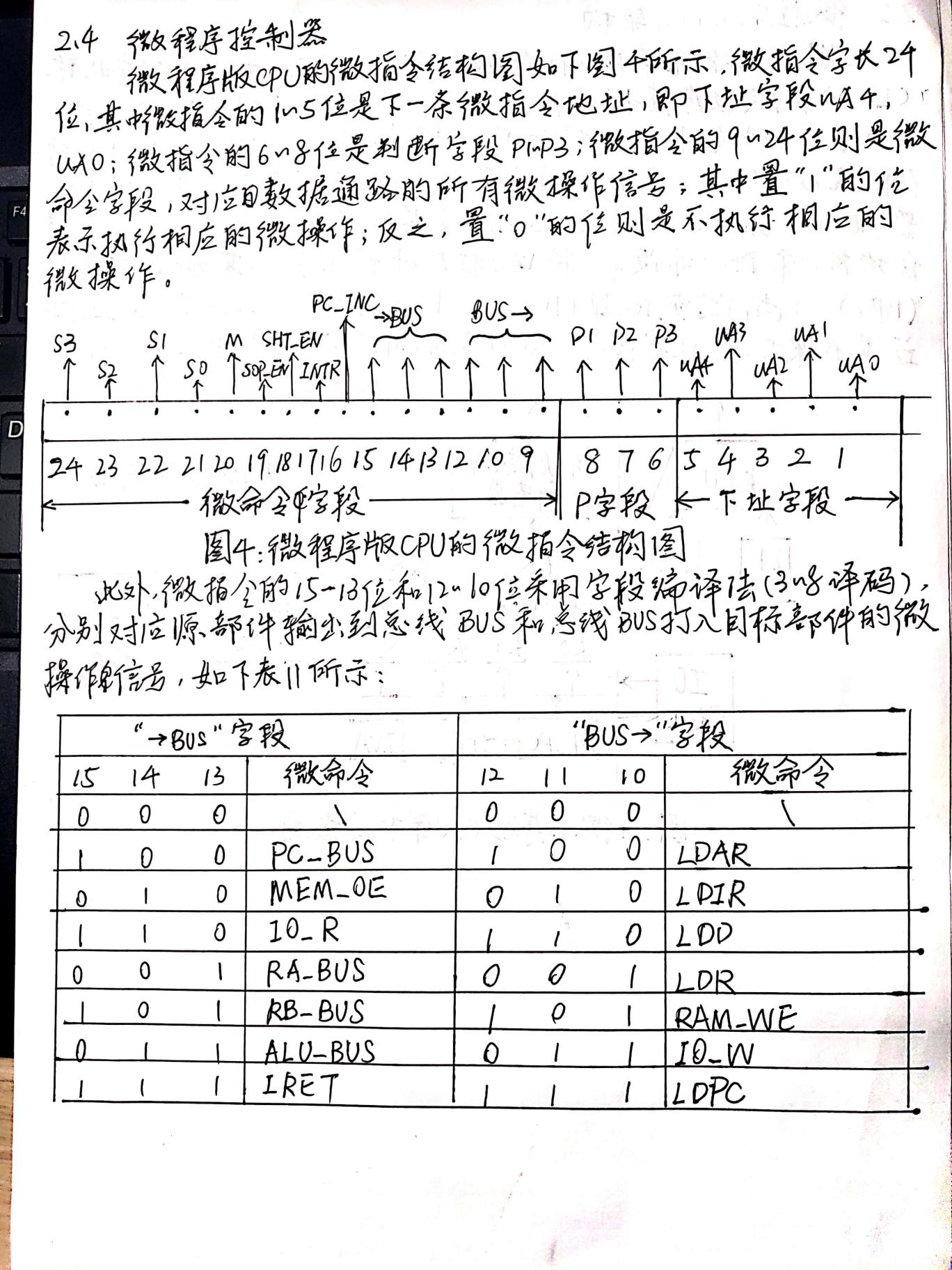


图3：微程序版CPU的时序电路

微程序版CPU的时序电路如上图3所示：用红色边框围起来的 CLOCK电路是CPU的基准时钟电路，系统时钟CLK可以由方波信号源AUTO-CLK提供（双击信号源可以自行选择方波信号频率）或者通过开关 MANUAL手动步进。当初始化信号ON=0或停机指令信号#HLT=0，时钟CLK阻塞（强制CLK=0），CPU停机。

CLOCK电路右侧是一个JK触发器74LS73实现的微指令状态机。由于微程序控制器和数据通路相互独立，两者操作可以并行执行，如下表10所示。所以，微程序版CPU的微指令周期只需要T1和T2两个状态，时钟信号CLK驱动微指令状态机循环输出节拍序列{T1，T2}，使状态顺序转移：T1->T2->T1->…

### 2.4 微程序控制器



基于上述微指令结构，本实验设计了如下图5所示的微程序控制器通路，包括三个8位ROM存储器2764组成的微指令存储器MROM1-3，三个寄存器74LS273组成的微指令寄存器MDR1-3，微指令译码电路，五位微地址寄存器MA0、MA4及微地址转移电路。

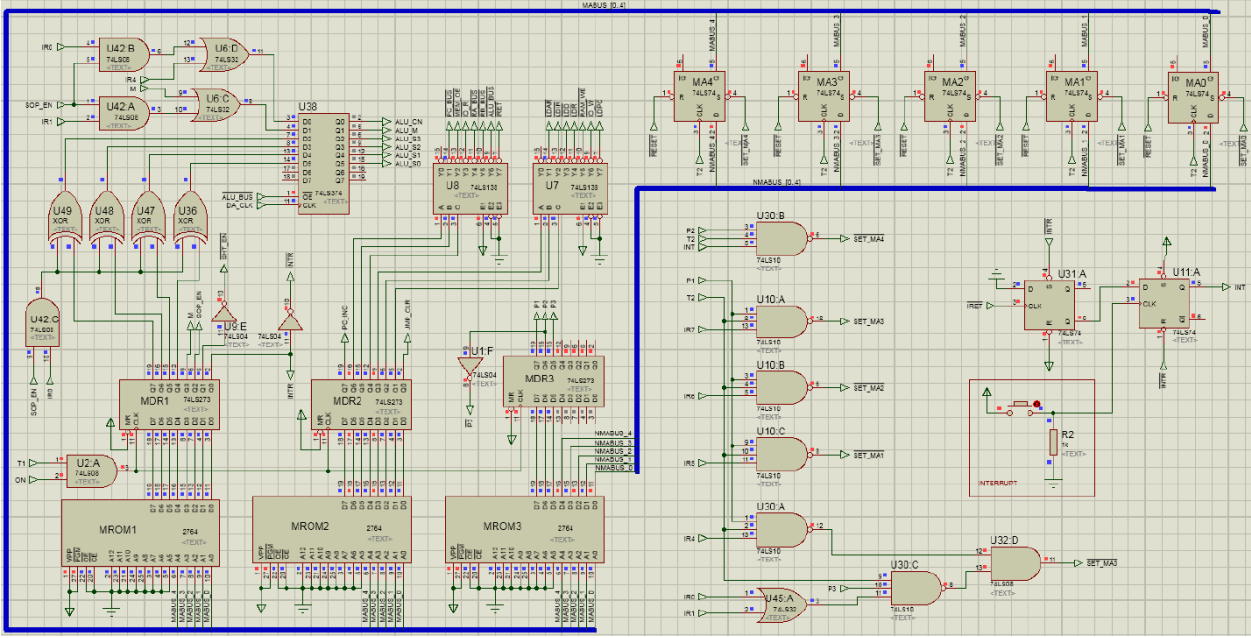


图5：微程序控制器通路

如下图6所示，24位微指令存储器MROM13共存放了32条微指令，所以微地址是 MABUS0\_4。当信号ON或T1节拍上升沿，微地址下址指定的微指令的微命令字段和P字段锁存到微指令寄存器MDR13，输岀微操作信号；而下址字段则送往微地址寄存器。

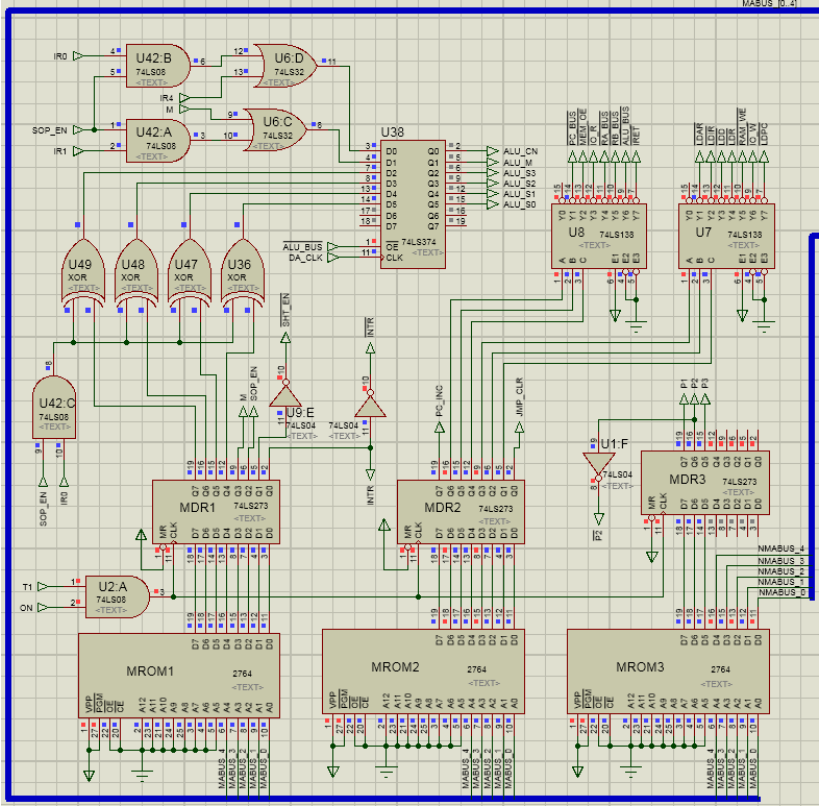


图6：微指令存储器、寄存器及其译码电路

其中，微指令寄存器MDR2的输出端采用了两个3-8译码器74LS138进行字段译码，分别实现把数据从源部件输出到总线BUS（译码器U8）和从总线BUS打入目标部件（译码器U7）的微操作。因为，在任何一条微指令中，仅有一条数据路径，该路径中只有一个源部件把数据打入总线BUS，其微操作信号互斥；也只有一个目标部件从总线BUS接收数据，其微操作信号也是互斥，因此，上述两种微操作信号可以分别用译码器实现。

因为本实验只有32条微指令，却需要实现38条机器指令。所以部分机器指令需要共用微指令。微指令寄存器MDRL的输岀译码电路采用了硬连线逻辑来实现以下单操作数指令：递增（INC）递减（DEC）、取反（NOT）和直通（THR），以及决定算术运算指令所需的CN操作信号（最低进位，用于补码运算的“求补+1”操作）

当执行指令SOP时，运算器74LS181控制端S3，S2，S1，s0，M，CN]=[0，0，0，0.0.0.因为 SOP EN=1，所以I1位修改M，I0位修改S3S0和CN（I4=0，不影响CN），如下表所示

OP码的I4则用来指定双操作数运算指令ADD和SUB的CN操作信号，如下表所示：

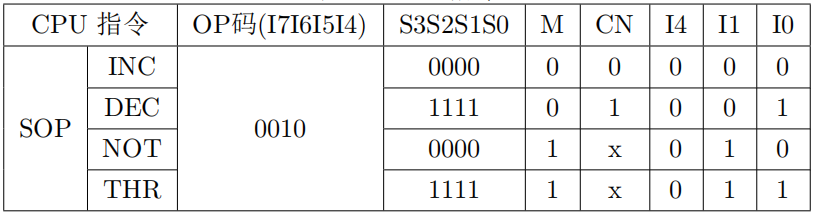


表12：SOP指令

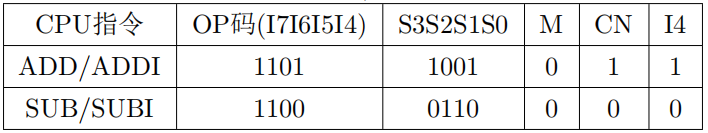


表13：ADD/SUB指令

此外，为了节省微指令地址，所有的双操作数和单操作数运算指令都在指令执行序列的第条微指令就锁存[S3,S2,S1,S0,M,CN]，待到最后一条微指令再打入运算器74LS181执行，从而得到运算结果输岀到总线BUS.因此，在第一条微指令的微操作信号DACⅠK上升沿跳变时刻把[S3,S2,S1,S0,M,CN]锁存到74LS374寄存器U38；在最后一条微指令周期，微操作信号#ALU BUS=0使能74LS374输出[S3,S2,S1,S0,M,CN]执行，再得到运算结果。

### 2.5 取指过程

除了空指令（NOP）和停机指令（H）以外，所有的CPU指令都包括了取指周期和执行周期。因为NOP指令OP码为“0000”，所以取指周期末尾PI（0765I4）译码的时候，直接返回取指周期（取下一条指令），没有执行周期。而HT指令与NOP指令完全相同，唯一不同是在取指周期后CPU硬件停机，需要手动 RESET“重启”才能跳出停机状态，进入下一条指令。此外，外部中断触发后，中断处理周期有专用的微指令使程序转向中断子程序。待到中断子程序末尾，最后条指令必须是中断返回指令（IRET），才能返回主程序。

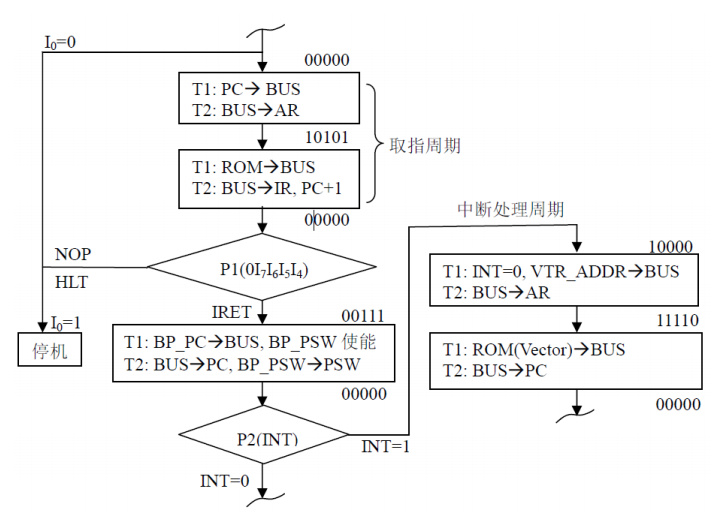


图8：取指周期、中断处理周期及系统指令的微程序流程图

上图8所示是取指周期、中断处理周期及NOP、HIT、IRET指令的微程序流程图，其中每个方框在时间上表示一个微指令周期，包括T1和T2两个节拍；在空间上表示数据从某个源部件经过总线BUS到达另一个目标部件的路径。每个方框的右上方是该微指令在控制存储器中的地址，右下方则是下一条微指令的地址。下表14则列出了上图8所对应的取指周期（即NOP/HLT指令）、中断处理周期及IRET指令的微指令代码：

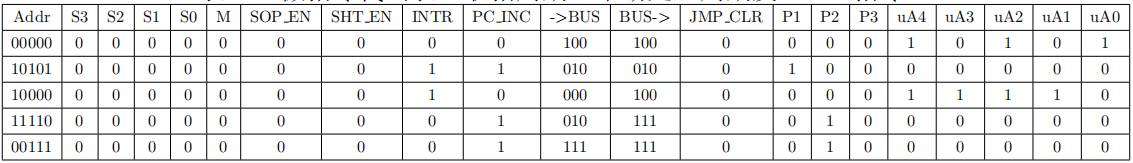


表14：微指令代码表（取指周期、中断处理周期及IRET指令）

本实验设计的存储器地址总线8位，地址空间256字节（0 OH FFH）.分配其中低半区（00H7FH）为ROM存储区（128字节），高半区（8 OH FFH）为RAM存储区（128字节），如下图39所示，存储器ROM和RAM共用一个地址寄存器AR，两个存储器共用 MEMOE信号作为存储器读信号，由地址最高位A7来作为两个存储器的片选信号。RAM存储器是可读写存储器，存放临时的数据。而ROM是只读存储器，存放程序和常量（采用堆栈操作指令访问）.因此，CPU程序和常量的存储容量最大是128字节，若程序和常量的代码量超过了128字节，则会越界出错同样的，因为只有存储器RAM允许写入，所以当存储器写信号RWE=0的时候，只有地址范围⑧ O.FFH是允许写入操作的；对地址范围0H，7FH的存储器单元进行写入操作是非法的。值得注意的是，上述存储器不同地址范围的读写差异必须由通过软件（程序员或汇编器）来判别。

### 2.6 寄存器及I/O 操作指令

寄存器操作指令包括一条单字节的寄存器间传送指令（MOⅣ）和一条双字节的寄存器赋值指令（ST）；Ⅰ/O操作指令包括三条单字节指令：输入指令（IN）、数据输出指令（OUT）及地址输出指令（OUTA）.下图13是MOV、SET指令和IN、OUT/OUTA指令的微程序流程图，其中（P1判断前）取指周期和若有中断触发的（P2判断后）中断处理周期参见上图8.此外，OUT和OUTA指令的微指令序列完全相同，由硬件逻辑区分。



图13：寄存器及I/O操作指令的微程序流程图

下表15列出了上图13对应的寄存器操作指令MOV、SET及I/O操作指令IN、OUT/OUTA的微指令代码。

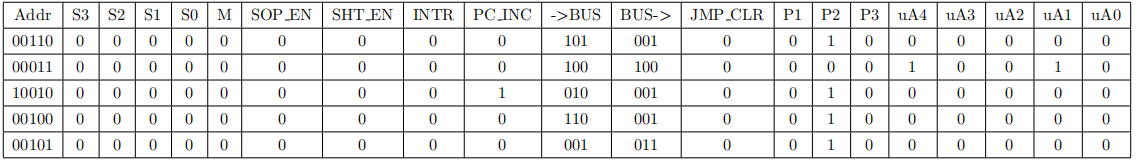


表15：微指令代码表（MOV/SET/IN/OUT指令）

### 2.7 存储器及堆栈操作指令

双字节存储器操作指令包括了取数指令LAD和存数指令STO，而单字节堆栈操作指令包括了岀栈指令POP和入栈指令PUSH，下图16是存储器操作指令LAD、STO和堆栈操作指令POP、PUSH的微程序流程图。从图中可以看出，POP指令只需要[11011和[11101]两条微指令就够了但是为了节省OP码，POP和LAD指令共用OP码“1000″，即共用第一条微指令[01000（即使POP指令其实并不需要微指令01000）.从而可以在第一条微指令的末尾用P3（Il0判断LAD和POP指令的不同路径：若II0=00，执行直接根据第二字节目标地址ADDR从存储器取数的双字节LAD指令；若II0≠00，则执行根据逻辑寄存器RB（RIR3）内容指定的目标地址从存储器取数的单字节POP指令。STO和PUSH指令的关系类似LAD和POP指令。

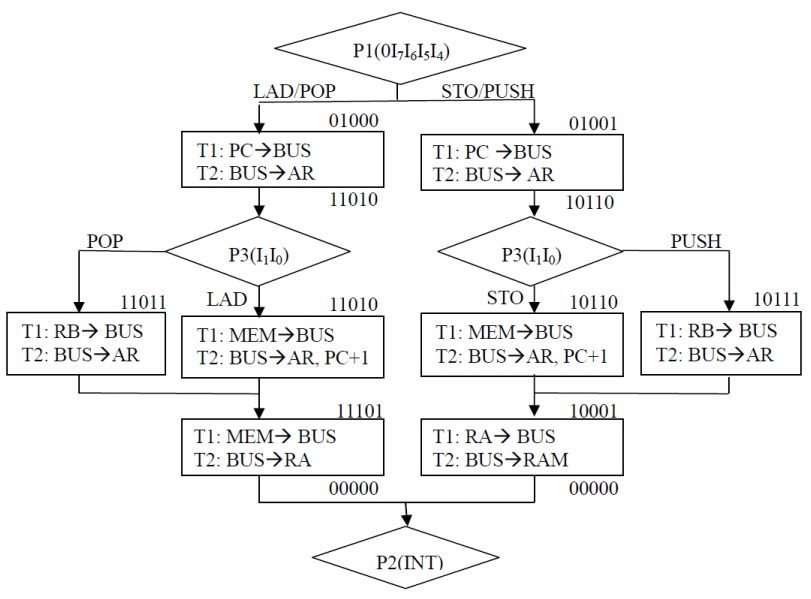


图16：存储器及堆栈操作指令的微程序流程图

下表16列出的存储器指令LAD/STO和堆栈指令POP/PUSH的微指令代码：

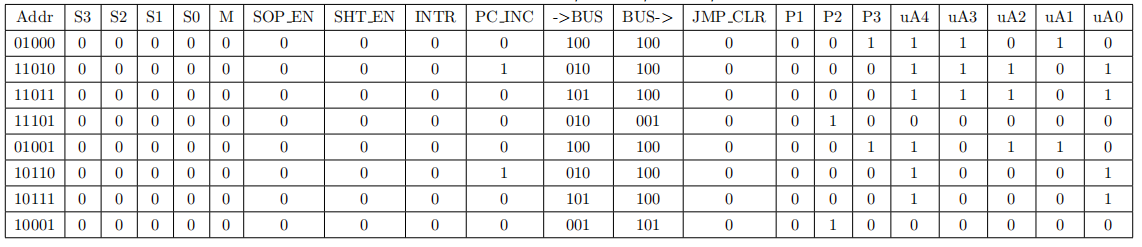


表16：微指令代码表（LAD/POP/ STO/PUSH指令）

### 2.8 跳转系列指令

下图17是JMPR/JxR指令和JMP/Jx指令的微程序流程图。从图中可以看出，JMPR/JxR指令只需要[111（指令就够了，但是为了节省OP码，两条跳转指令共用OP码“000”，即共用第一条微指令[0001（即使JMPR/JR指令其实并不需要[0001微指令）.从而可以在第一条微指令的末尾釆用P3（I⑩0）区分两种跳转指令的不同路径：若II0=00，执行直接根据第二字节目标地址ADDR跳转的双字节JMP/Jx指令；若II0≠00，则执行根据逻辑寄存器RB（R1R3）内容指定的目标地址跳转的单字节JMPR/JxR指令。

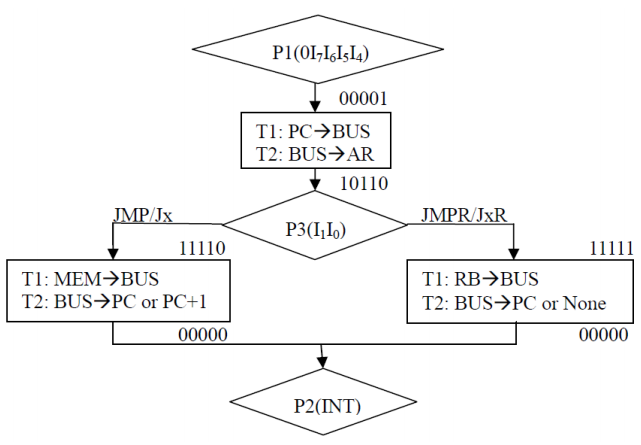


图17：跳转系列指令的微程序流程图

### 2.9 算术逻辑运算系列指令

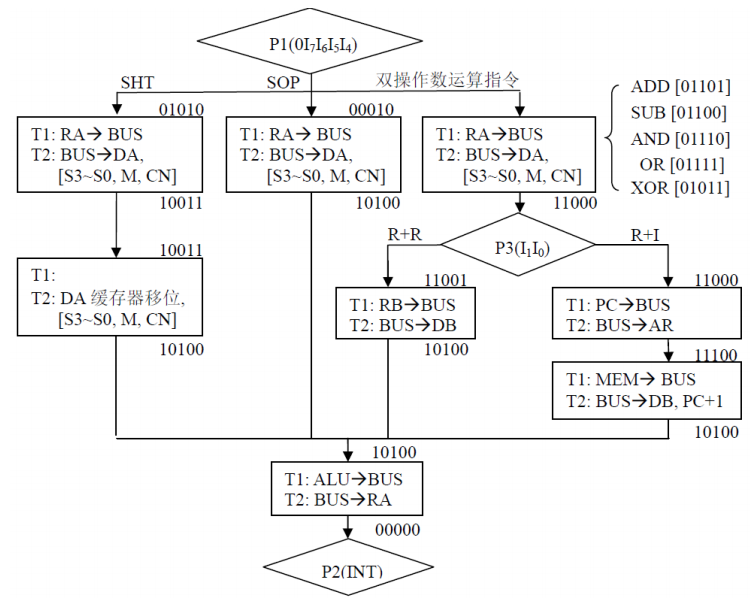


图19：算术逻辑运算系列指令的微程序流程图

算术逻辑运算系列指令包括了单字节的移位指令SH，单字节单操作数运算指令SOP，以及五条单字节的双操作数运算指令（ADD/SUB/AND/OR/XOR）和五条双字节的双操作数运算指令（ADDI/ SUBI/ANDI/ ORI/ XORI），其微程序流程图如上图19所示。为了节省微指令，所有的运算指令都在第一条微指令期间锁存74181运算器的控制端逻辑[S3.S2，S1，SO，M，CN.此外，五种双操作数的运算指令都采取在第一条微指令的末尾采用P3（II判断双字节和单字节指令的不同路径：若I0=00，执行操作数分别来自逻辑寄存器RA和指令第二字节（立即数IMM）的双字节指令；若II≠00，则执行操作数全部来自寄存器的单字节指令。上述算术逻辑运算指令的微指令代码表如下表18所示：

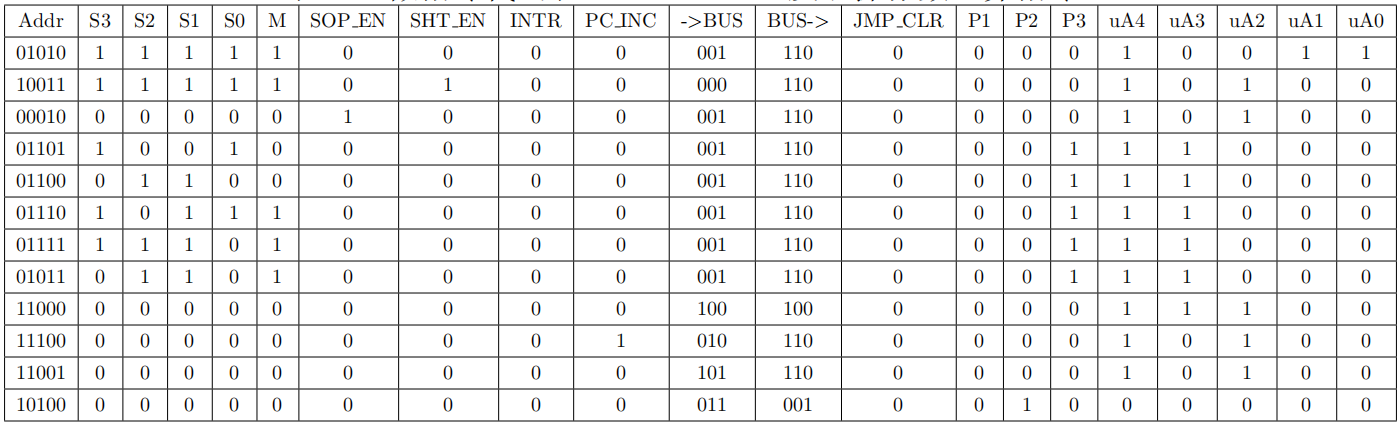
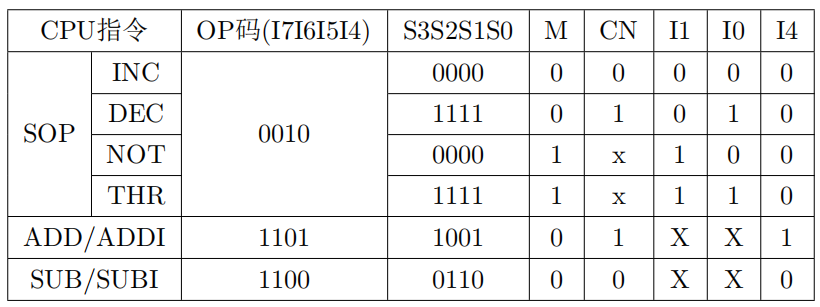


表18：微指令代码表（SHT、SOP及双操作数运算指令）

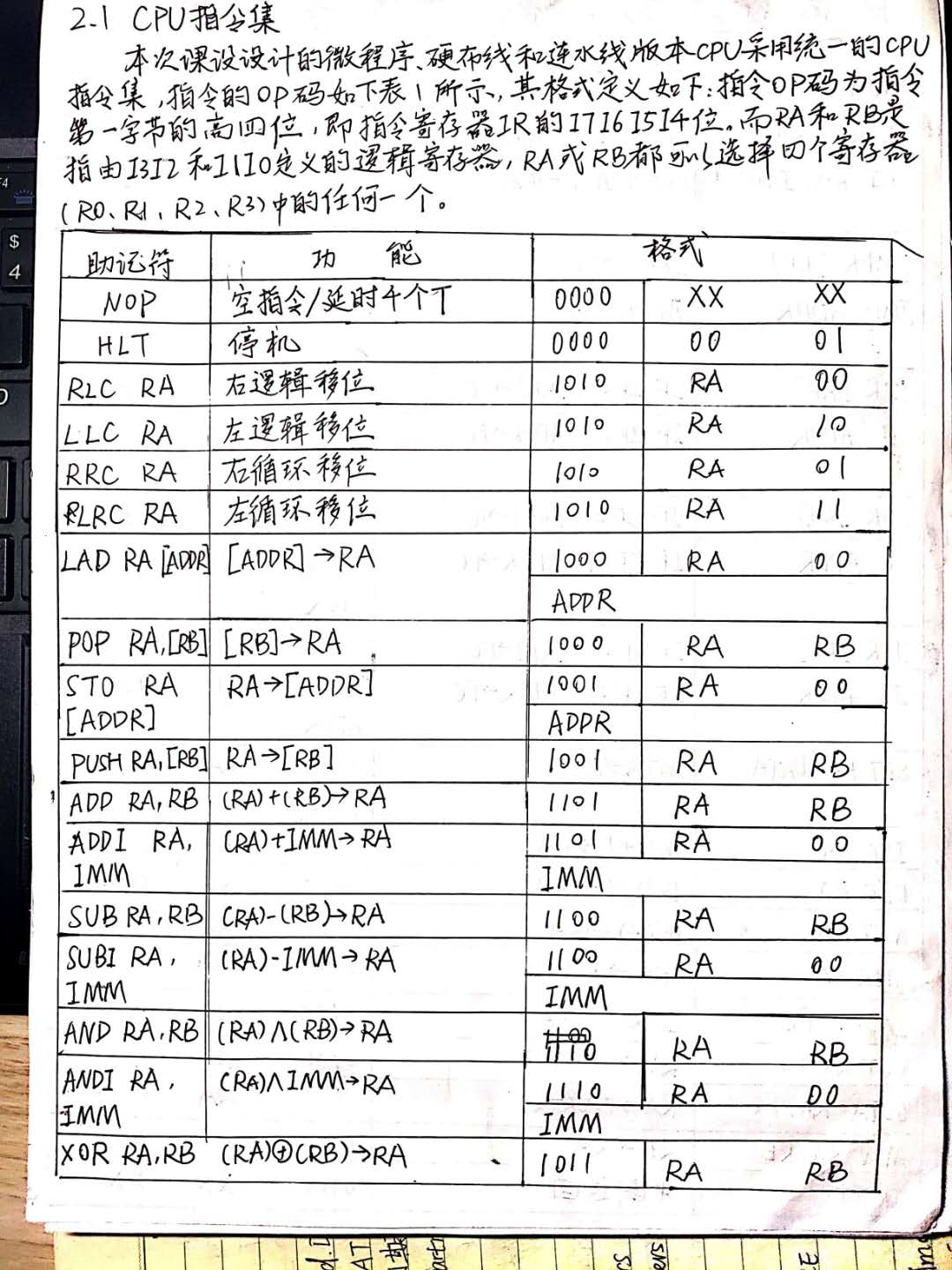
值得注意的是，单操作数运算指令SOP只有一条微指令[0001必须由上图18（右）所示的硬件逻辑电路根据SOP指令的I⑩0位修改运算器控制端S3，S2，S1，S0，M，CN，实现递增（INC）、递减（DEC）、取反（NOT）、直通（工HR）四个功能。同时，OP码的I4位则用来指定双操作数算术运算指令ADD和SUB的CN操作信号，如下所示

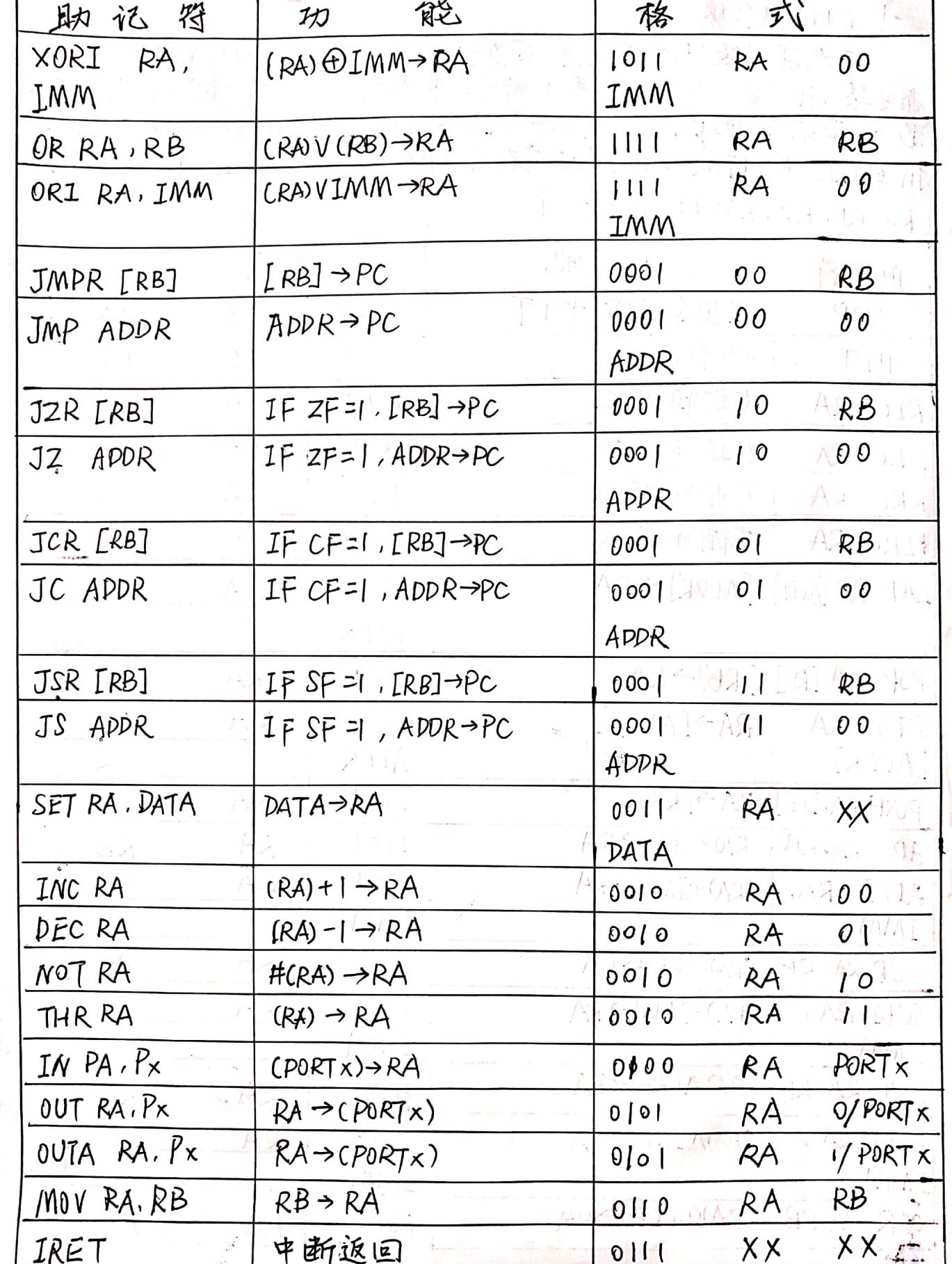


## 三、详细设计

### 3.1模型机指令系统设计

主要工作：根据任务1、2实现需求，完善已给出的指令系统。

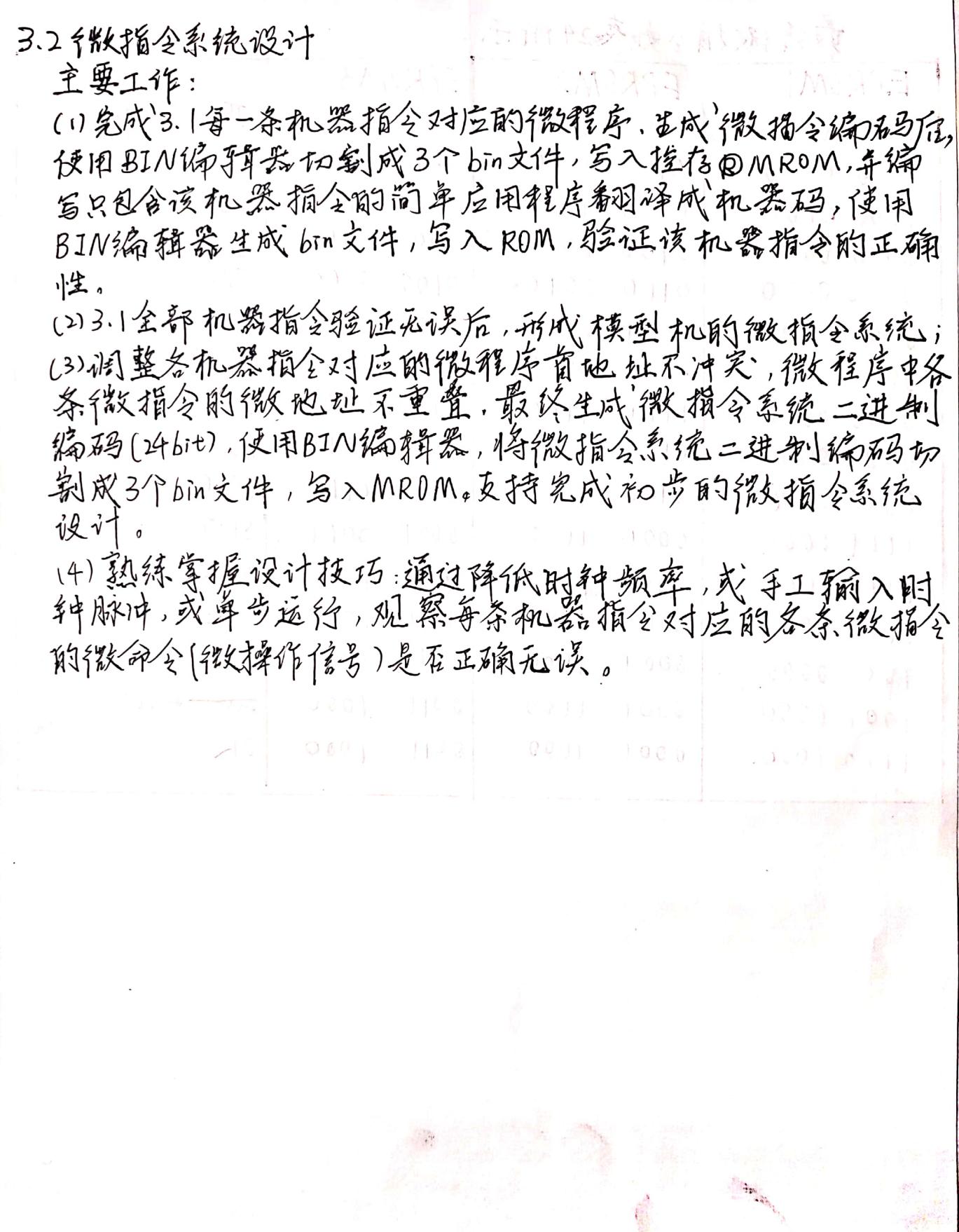




PS:

1. XX: 表示任意
2. RA\RB: 表示寄存器（实际在写程序的时候，不是用RA表示第一个寄存器，而是用R1表示一个寄存器，如MOV 3,R1;）
3. ADDR: 表示地址
4. DATA：表示数据，如 MOV 3,R1;，这里的3就是DATA
5. 指令PUSH/POP和JMP系列指令的JMPR/JCR/JZR/JER的RB只能是R1\R2\R3，不能是R0
6. IN和OUT指令的 汇编 要指明是对哪个外设端口操作，外设定义了四个端口 PORT0-3；其中PORT0和1 是双向端口即IN 和OUT都能操作的，PORT 2和3是单向输入端口只有IN指令。
7. IN指令的【IR1,IR0】指明操作的是哪个外设PORTx
8. OUT指令的IR0指明操作的外设PORTx，IR1指明传送外设的内容是数据还是地址（即IR1=0相当于ALE=0，向外设发送数据；IR1=1相当于ALE=1，向外设发送地址）。
9. HLT指令改为跟NOP指令相同OP码，然后IR0=1是 停机，IR0=0是NOP指令

### 3.2微指令系统设计



主要工作：

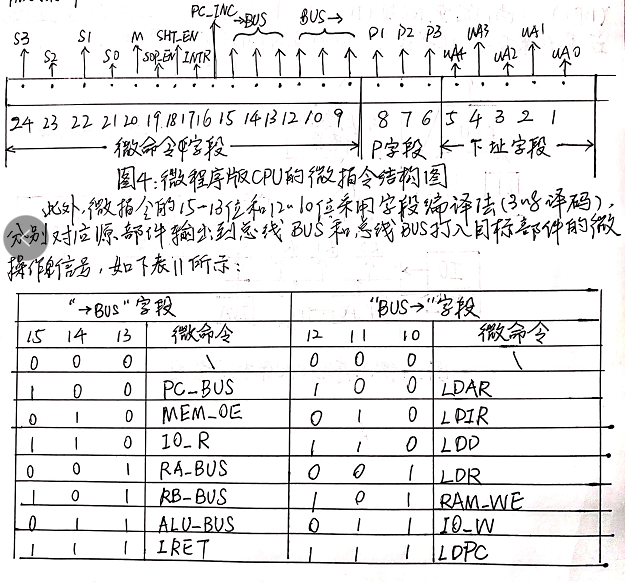
（1）完成3.1每一条机器指令对应的微程序，生成微指令编码后，使用BIN编辑器切割为3个bin文件，写入控存MROM，并编写只包含该机器指令的简单应用程序翻译成机器码，使用BIN编辑器生成bin文件，写入ROM，验证该机器指令的正确性。

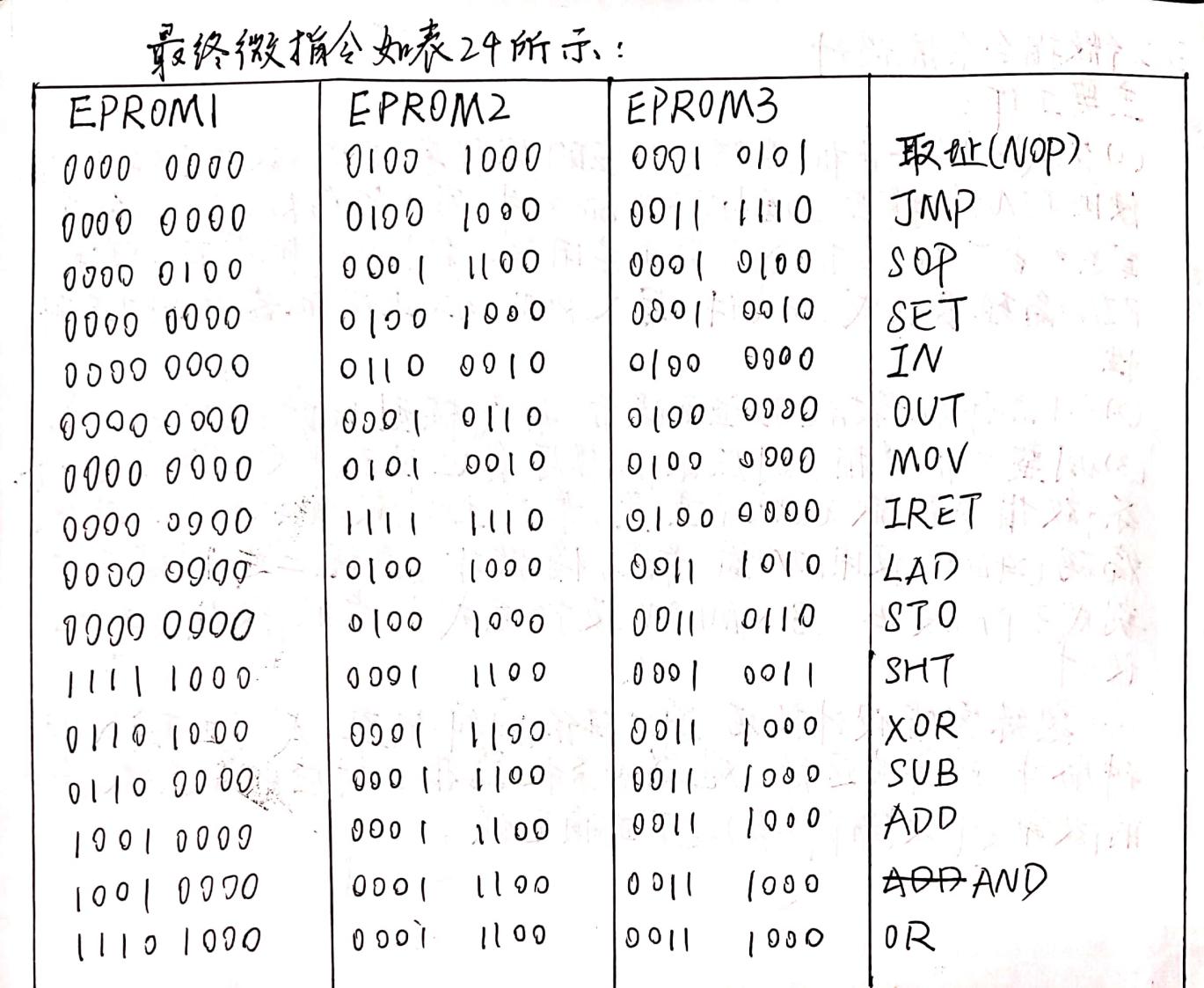
（2）3.1全部机器指令验证无误后，形成模型机的微指令系统；

（3）调整各机器指令对应的微程序首地址不冲突，微程序中各条微指令的微地址不重叠，最终生成微指令系统二进制编码（24bit），使用BIN编辑器，将微指令系统二进制编码切割成3个bin文件，写入MROM。支持完成初步的微指令系统设计。

（4）熟练掌握设计技巧：通过降低时钟频率，或手工输入时钟脉冲，或单步运行，观察每条机器指令对应的各条微指令的微命令（微操作信号）是否正确无误。

为尽可能少的修改模板电路这里直接采用模板电路的微指令格式，如图21和表2所示。





|  |  |  |  |
| --- | --- | --- | --- |
| EPROM1 | EPROM2 | EPROM3 |  |
| 0000 0000 | 0100 1000 | 0001 0101 | 取址(NOP) |
| 0000 0000 | 0100 1000 | 0011 1110 | JMP |
| 0000 0100 | 0001 1100 | 0001 0100 | SOP |
| 0000 0000 | 0100 1000 | 0001 0010 | SET |
| 0000 0000 | 0110 0010 | 0100 0000 | IN |
| 0000 0000 | 0001 0110 | 0100 0000 | OUT |
| 0000 0000 | 0101 0010 | 0100 0000 | MOV |
| 0000 0000 | 1111 1110 | 0100 0000 | IRET |
| 0000 0000 | 0100 1000 | 0011 1010 | LAD |
| 0000 0000 | 0100 1000 | 0011 0110 | STO |
| 1111 1000 | 0001 1100 | 0001 0011 | SHT |
| 0110 1000 | 0001 1100 | 0011 1000 | XOR |
| 0110 0000 | 0001 1100 | 0011 1000 | SUB |
| 1001 0000 | 0001 1100 | 0011 1000 | ADD |
| 1011 1000 | 0001 1100 | 0011 1000 | AND |
| 1110 1000 | 0001 1100 | 0011 1000 | OR |

表24：微指令集

## 四、基于模型机指令系统的应用程序设计

### 4.1 任务内容

**任务：**将“自己的学号（12位）”和 “自己的身份证号码后12位（最后1位为X的以0替换）”以压缩BCD格式（4位二进制表示1位十进制数）共12个字节保存在程序存储器ROM中**（用UltraEdit编辑可生成bin格式文件，或按照章节“2.3.3 ROM批量导入数据的技巧”所述生成的hex文件，将bin或hex加载到ROM中）**，程序启动后，根据拨码开关**DSW1输入值**执行下面操作：

DSW1输入值 % 3 = 0 将保存到程序存储器中的“自己的学号”写入数据存储器RAM中，写入RAM的**相对偏移地址**由拨码开关DSW1输入值（HEX）决定。

DSW1输入值 % 3 = 1 将保存到程序存储器中的“自己的身份证号”写入数据存储器RAM中，写入RAM的**相对偏移地址**由拨码开关DSW1输入值（HEX）决定。

DSW1输入值 % 3 = 2 将保存到程序存储器中的“自己的学号”和 “自己的身份证号”进行BCD码算术相加，然后将结果（6个字节的BCD码）写入数据存储器RAM中，写入RAM的**相对偏移地址**由拨码开关DSW1输入值（HEX）决定。

之后从RAM中读取该结果，送到DOUT显示，每个字节的显示时间不超过0.5秒。

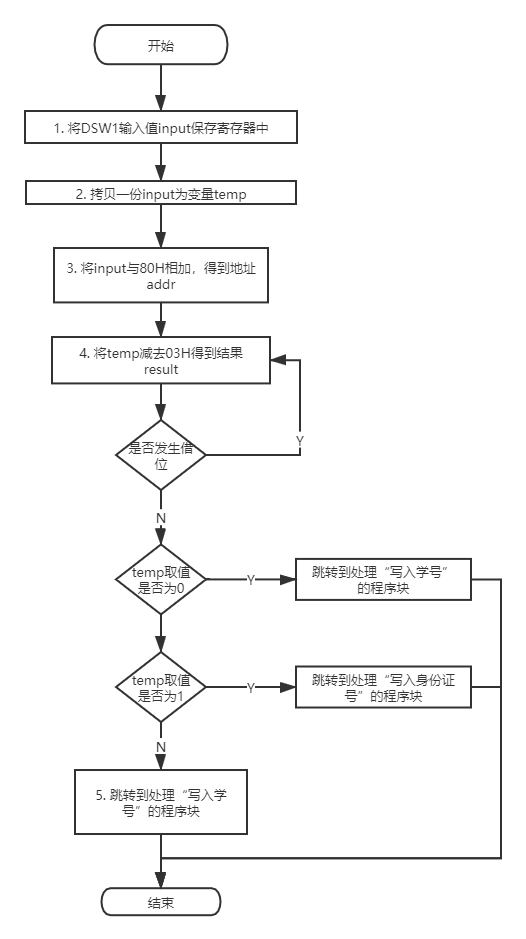
两数算术相加时要考虑到低位向高位的进位，如：

（学号）201830013463 + （身份证号）199809293216 = 401639306679

熟练掌握设计技巧：通过降低时钟频率，或手工输入时钟脉冲，或单步运行，观察每条机器指令对应的各条微指令的微命令（微操作信号）是否正确无误。

### 4.2 设计思路

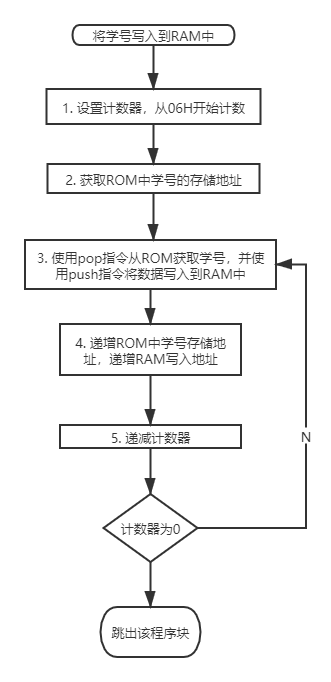
#### 4.2.1 根据拨码开关的值跳转到相应的程序块



伪代码

1. 将DSW1输入值input保存寄存器中；
2. 拷贝一份input为变量temp；
3. 将input与80H相加，得到地址addr；
4. 将temp减去03H得到结果result
5. 如果没有发生借位，则跳转到第4步，否则执行下一步；
6. 如果发生借位，此时temp是一个小于3的数，只可取值[0,1,2]；
7. 如果temp取值为0，则跳转到处理“写入学号”的程序块；
8. 如果temp取值为1，则跳转到处理“写入身份证号”的程序块；
9. 跳转到处理“学号和身份证号相加”的程序块。

#### 将学号写入到RAM中



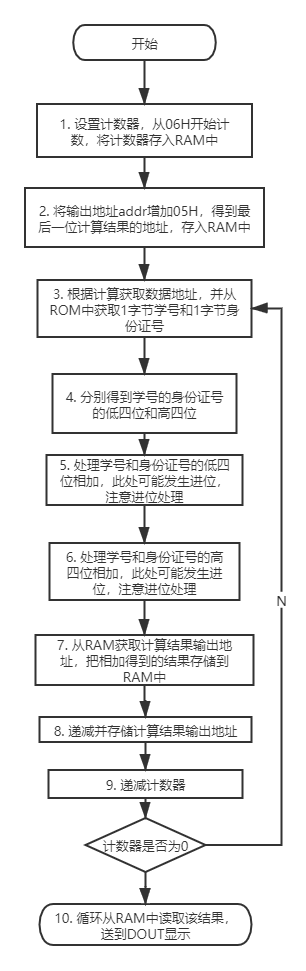
伪代码

1. 设置计数器，从06H开始计数；
2. 获取ROM中学号的存储地址；
3. 使用pop指令从ROM获取学号，并使用push指令将数据写入到RAM中；
4. 递增ROM中学号存储地址，递增RAM写入地址；
5. 递减计数器，如果计数器为0，则跳出该程序块，否则跳转到第3步

#### 4.2.3 将身份证号写入到RAM中

同4.2.2，本部分不再赘述

#### 两BCD码（学号和身份证号）相加



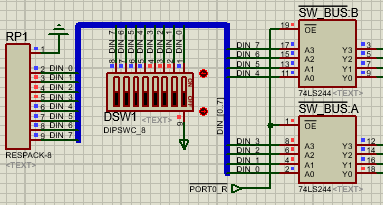
伪代码

1. 设置计数器，从06H开始计数，将计数器存入RAM中；
2. 将输出地址addr增加05H，得到最后一位计算结果的地址，存入RAM中；
3. 根据计算获取数据地址，并从ROM中获取1字节学号和1字节身份证号；
4. 分别得到学号的身份证号的低四位和高四位；
5. 处理学号和身份证号的低四位相加，此处可能发生进位，注意进位处理；
6. 处理学号和身份证号的高四位相加，此处可能发生进位，注意进位处理；
7. 从RAM获取计算结果输出地址，把相加得到的结果存储到RAM中；
8. 递减并存储计算结果输出地址；
9. 递减计数器，如果计数器为0，则执行下一步，否则跳转到第3步；
10. 循环从RAM中读取该结果，送到DOUT显示

## 五、本课程设计的验收方法

### 5.1 将学号写入RAM中

#### 5.1.1 拨动DSW1到0000 0000



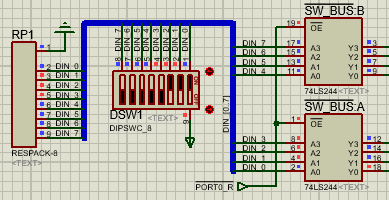
点击RESET按钮启动。

#### 5.1.2 查看RAM

### 

### 将学号写入RAM中

#### 5.2.1 拨动DSW1到0000 0111



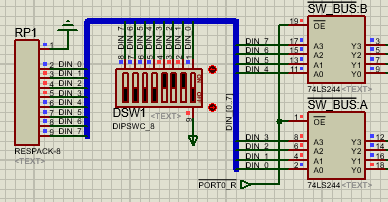
点击RESET按钮启动。

#### 5.2.2 查看RAM

### 

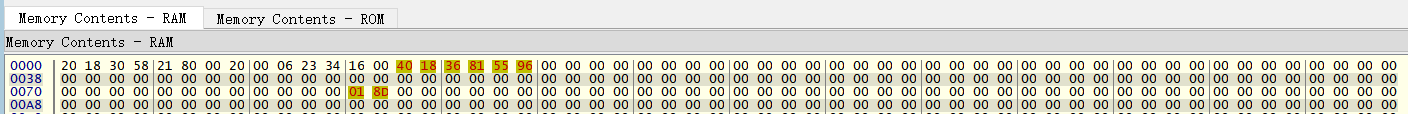
### 5.3 两BCD码（学号和身份证号）相加

#### 5.3.1 拨动DSW1到0000 1110

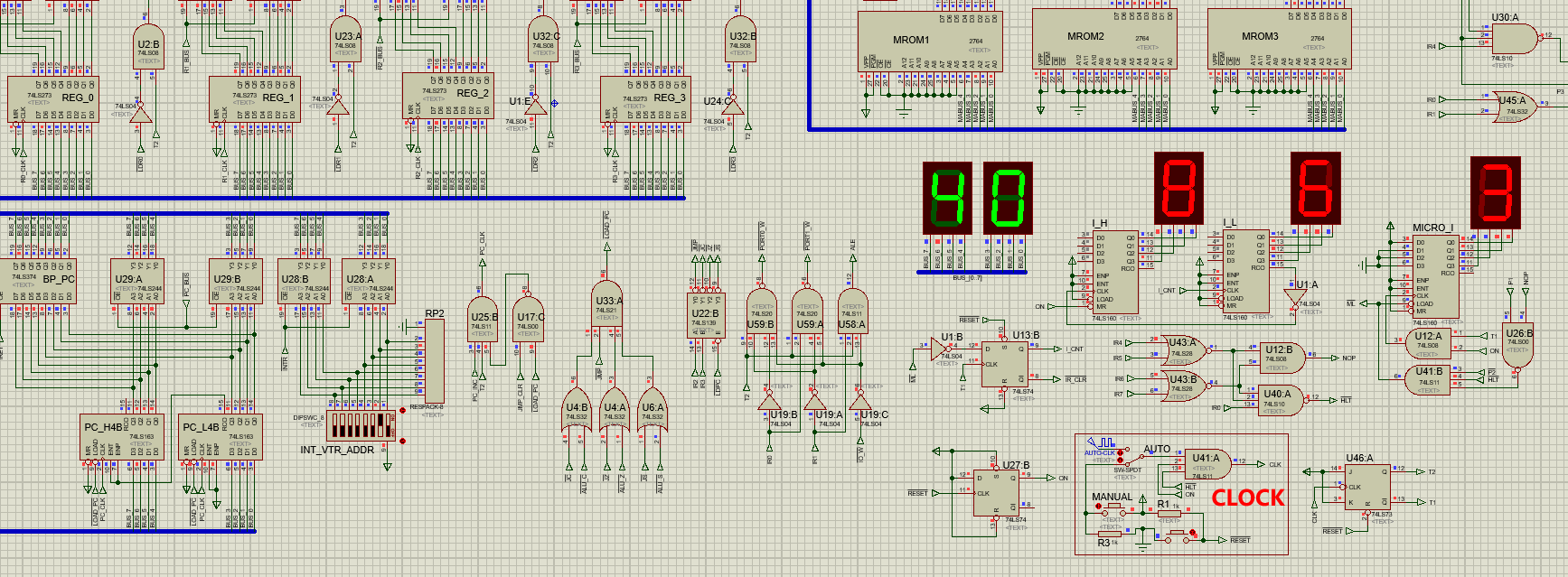


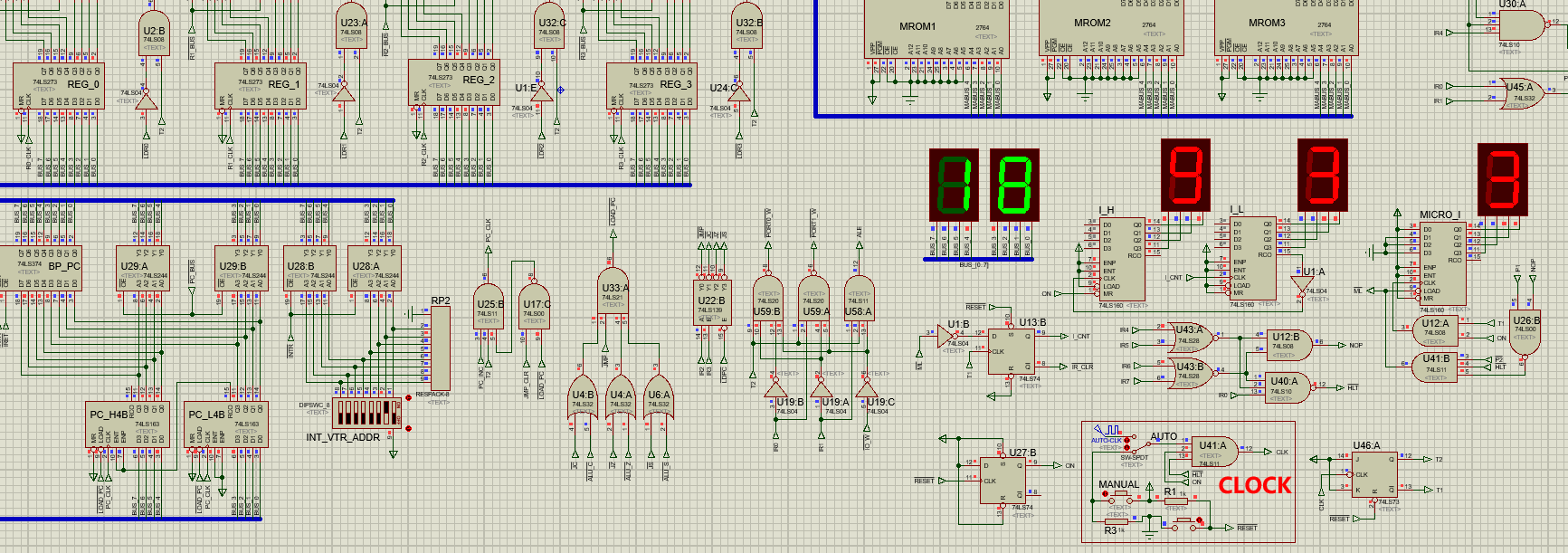
点击RESET按钮启动。

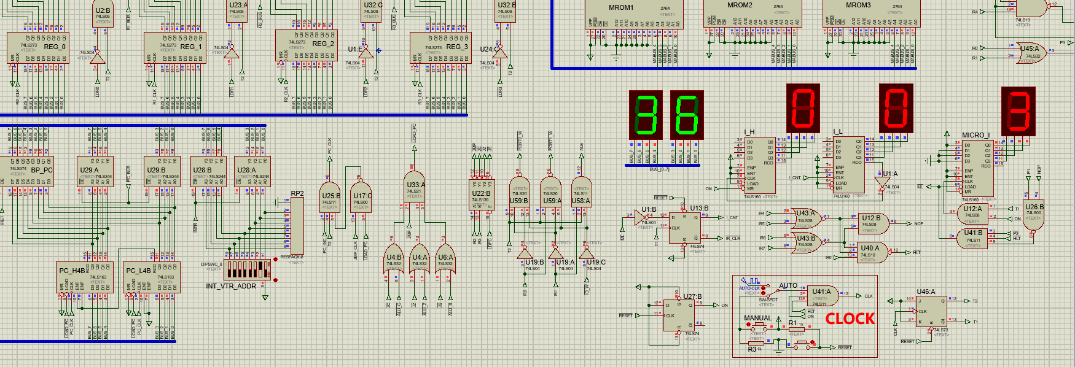
#### 5.3.2 查看RAM

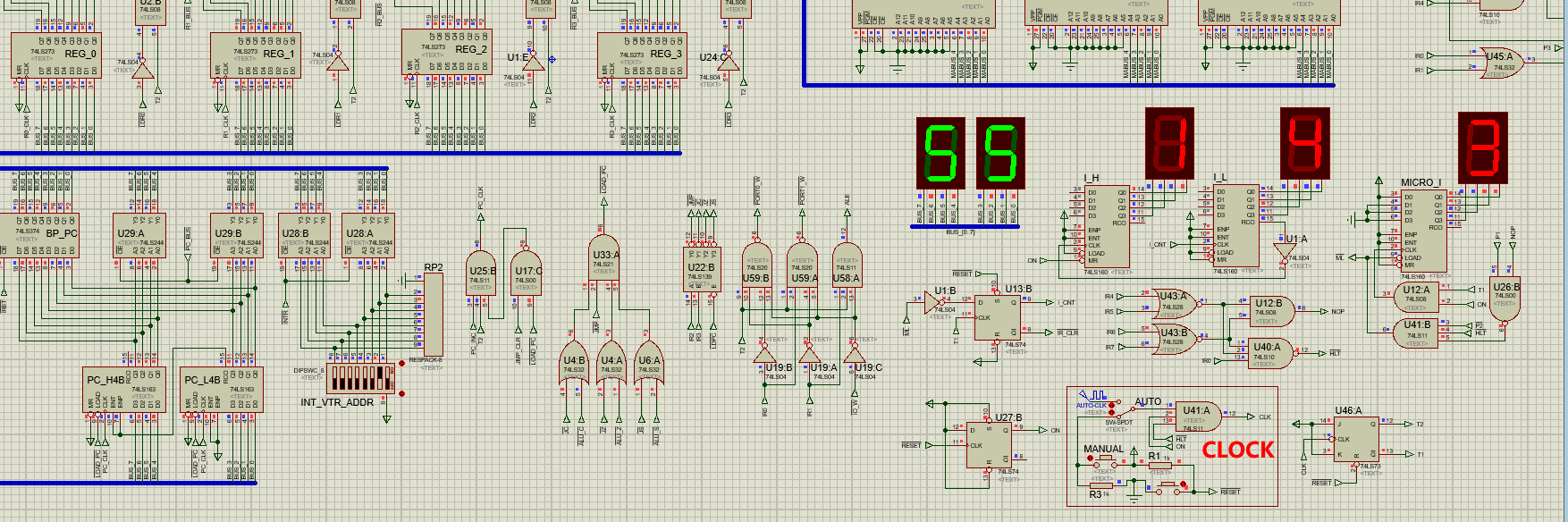
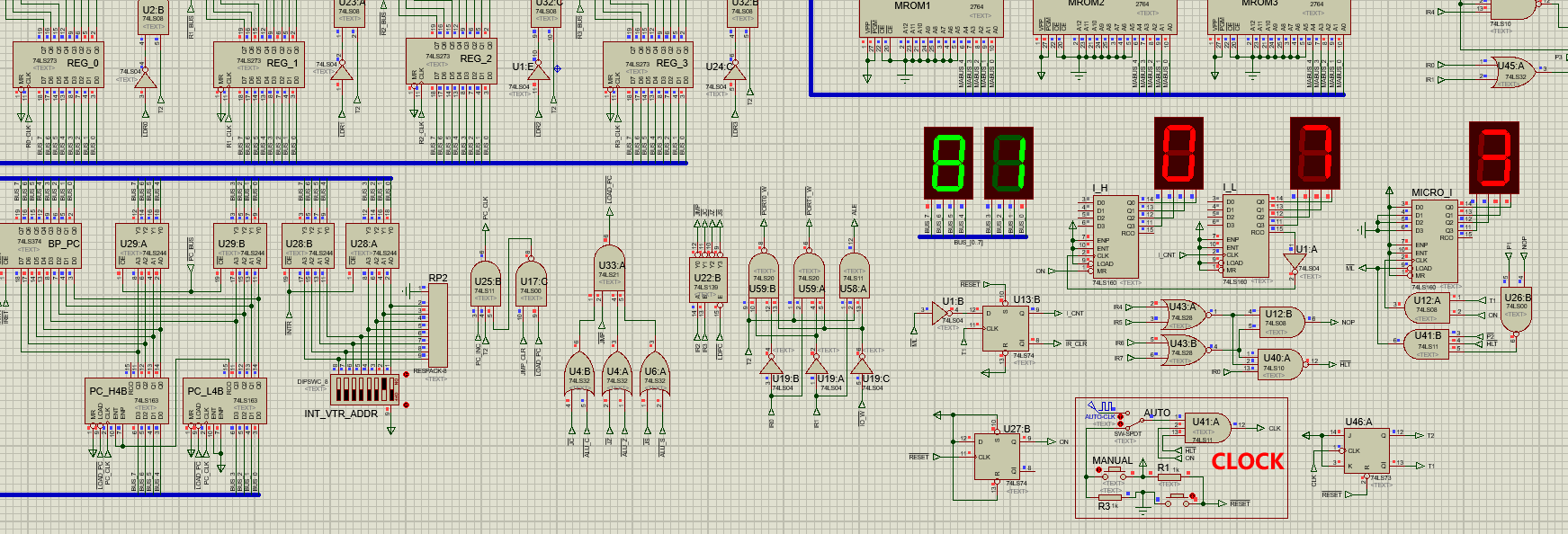


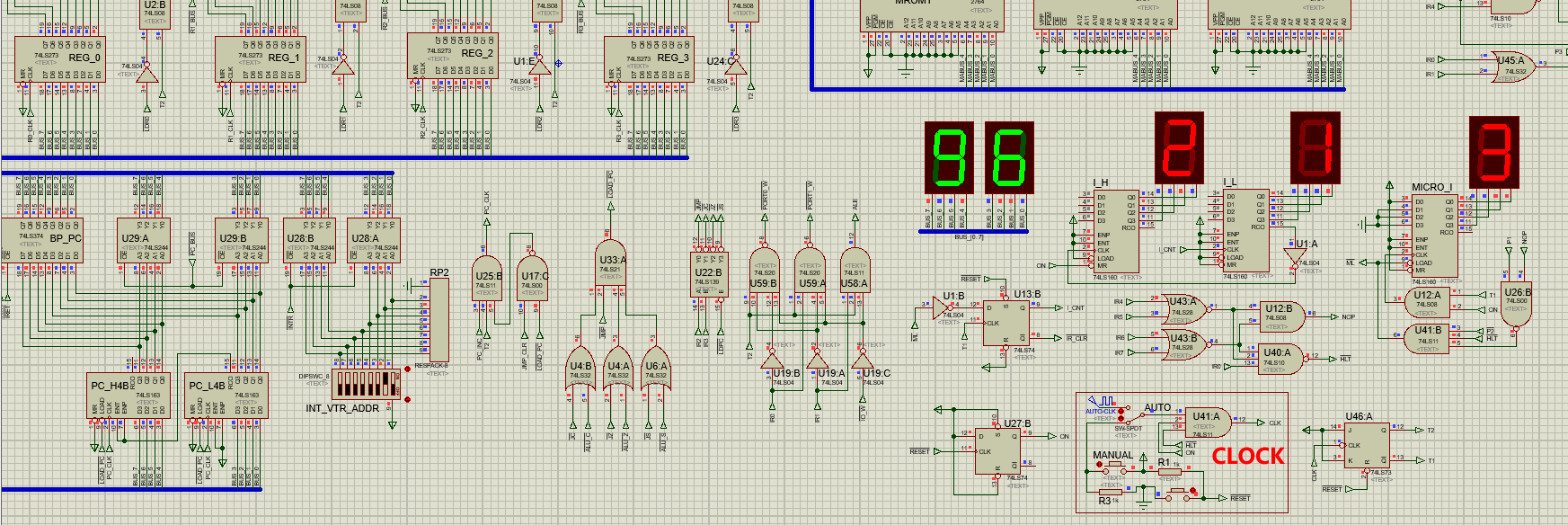
#### 循环从RAM中读取该结果，送到DOUT显示











## 六、课程设计总结

### 6.1 遇到的困难

#### 6.1.1 如何使用JMP/JC/JZ实现条件跳转

在4.2.1 根据拨码开关的值跳转到相应的程序块 这一步中，我一开始的思路是用求模指令计算出DSW1输入值%3的结果，然后根据结果跳转到响应的程序块。后来发现指令集中既没有求模指令，也没有像是if (x == 0){} if else (x == 1){} else {}这样的判断结果是否等于某个值才跳转的指令。再请教了老师之后，才明白应该采用AND、SUB等指令来替代求模从而实现跳转。

于是我花费了比较多的时间从网络上了解到了JMP/JC/JZ这三条指令的意义，着重了解了这三条指令的跳转条件以及3个控制标志位：

1. CF(进位标志位)：当执行一个加法（减法）运算时，最高位产生进位（或借位）时，CF为1，否则为0。
2. ZF零标志位：若当前的运算结果为零，则ZF为1，否则为0。
3. SF符号标志位：该标志位与运算结果的最高位相同。即运算结果为负，则SF为1，否则为0。



#### 6.1.2 如何实现两个BCD码加法运算

这一部分的程序可以说是困扰了我很久，也是花费时间最多的部分。我的大致思路就是分别取出两个BCD码的低四位和高四位，先处理低四位，这一过程中要特别注意如果运算结果大于9，就要向高四位进位；处理高四位相加也是相同思路，而且也要注意进位问题。

#### 6.1.3 谨防程序代码过多导致溢出问题

因为ROM的访问空间只有00-7F，所以指令+数据的空间至多只能有128字节，如果超出，则会发生不可预料的结果。

### 6.2 收获

#### 6.2.1 理解了微程序型CPU的基本工作原理

本次课设通过编写指令程序代码，实现将机器指令转化成微指令程序，从而完成一些比较简单的功能。看似简单的功能背后却是控制器、运算器、存储器、IO接口等多个设备联合工作的结果，一旦某个设备出现问题，就会引发不可预料的结果，因此CPU的制作实现是一个专业、严谨、耐心的过程。

#### 6.2.2 掌握了指令的编写

本次课设的任务主要是基于模型机指令系统的应用程序设计，将编写的代码载入ROM，CPU便会从首地址开始执行指令。许多平时看似简单的功能，在CPU层面的实现却花费了大量的时间和精力，这让我领悟到一个道理：将底层的指令集封装成高级编程语言，可以大幅提高开发人员的工作效率。

## 七、参考文献

计2018级2班课程设计任务书.pdf

课程设计案例参考.pdf