# 实验日志-控制器

## 一，模拟器中指令的执行情况载图

### 1.开机


（1）开始为初始状态，所有数值均为 0

（2）执行当前地址 M_ADDR==01H 下的指令：PC→AR; PC+1；

（3）经过此步骤 PC == 01H; M_NXT_ADDR == 02H

### 2.读取 LDA 指令


（4）执行当前地址 M_ADDR==02H 下的指令：RAM→IR;功能跳转到 OP==02H 时的操作 LDA；经过此步骤，OP == 20H; M_NXT_ADDR == 09H

（5）执行当前地址 M_ADDR==09H 下的指令：PC→AR; PC+1;

（6）经过此步骤，PC == 02H; M_NXT_ADDR == 15H; AR == 01H

（7）执行当前地址 M_ADDR==15H 下的指令：RAM→AR; 此时 01H 地址的值为 0DH，因此 AR 被赋值为 0DH；经过此步骤，M_NXT_ADDR == 16H; AR == 0DH

（8）执行当前地址 M_ADDR==16H 下的指令：RAM→R5; 此时 0DH 地址的值为 55H，因此 R5 被赋值为 55H；经过此步骤，M_NXT_ADDR == 01H; R5 == 55H

（9）执行当前地址 M_ADDR==01H 下的指令：PC→AR; PC+1；

（10）经过此步骤，PC == 03H; M_NXT_ADDR == 02H; AR == 02H

## 3.读取并执行指令 ADD

（11）执行当前地址 M_ADDR==02H 下的指令：RAM→IR;功能跳转到 OP==C0H 时的操作 ADD；经过此步骤，OP == C0H; M_NXT_ADDR == 0E

（12）执行当前地址 M_MXT_ADDR == 0E 下的指令：PC→AR; PC+1;

（13）经过此步骤，PC == 04H; M_NXT_ADDR == 03H; AR == 03H

（14）执行当前地址 M_ADDR==03H 下的指令：RAM→AR; 此时 03H 地址的值为 0EH，因此 AR 被赋值为 0EH

```
PC:03 OP:C0 M_ADDR:02 M_NXT_ADDR:0E
AR:02 DR1:00 DR2:00 R5:55

PC:04 OP:C0 M_ADDR:0E M_NXT_ADDR:03
AR:03 DR1:00 DR2:00 R5:55

PC:04 OP:C0 M_ADDR:03 M_NXT_ADDR:04
AR:0E DR1:00 DR2:00 R5:55

PC:04 OP:C0 M_ADDR:04 M_NXT_ADDR:05
AR:0E DR1:00 DR2:8A R5:55

PC:04 OP:C0 M_ADDR:05 M_NXT_ADDR:06
AR:0E DR1:55 DR2:8A R5:55

PC:04 OP:C0 M_ADDR:06 M_NXT_ADDR:01
AR:0E DR1:55 DR2:8A R5:DF

PC:05 OP:C0 M_ADDR:01 M_NXT_ADDR:02
AR:04 DR1:55 DR2:8A R5:DF
```

（15）经过此步骤，M_NXT_ADDR == 04H; AR == 0EH

（16）执行当前地址 M_ADDR==04H 下的指令：RAM→DR2; 此时 0EH 地址的值为 8AH，因此 DR2 被赋值为 8AH；经过此步骤，M_NXT_ADDR == 05H; DR2 == 8AH

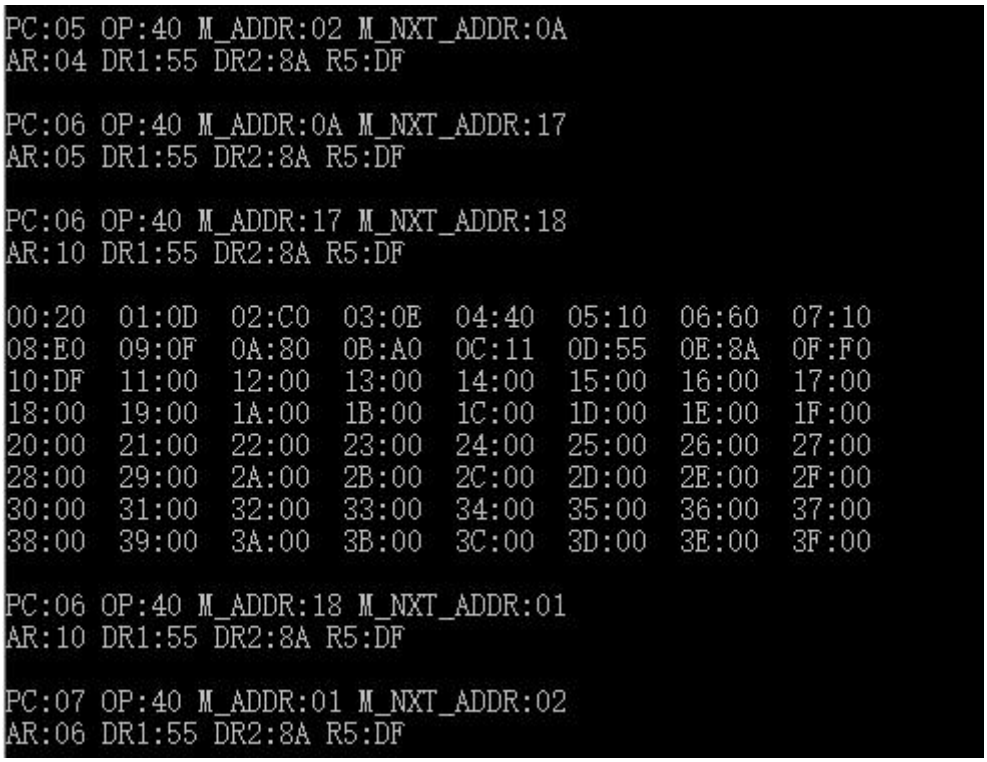
（17）执行当前地址 M_ADDR==05H 下的指令:R5→DR1; 此时 R5 的值为 55H,因此 DR1 被赋值为 55；经过此步骤，M_NXT_ADDR == 06H; DR1 == 55H

（18）执行当前地址 M_ADDR==06H 下的指令：ALU→R5; 此时 ALU 进行加法运算后的的值为 DFH，因此 R5 被赋值为 DFH；经过此步骤，M_NXT_ADDR == 01H; R5 == DFH

（19）执行当前地址 M_ADDR==01H 下的指令：PC→AR; PC+1;

（20）经过此步骤，PC == 05H; M_NXT_ADDR == 02H; AR == 04H

## 4.读取并执行指令 STA




（21）执行当前地址 M_ADDR==02H 下的指令：RAM→IR;功能跳转到 OP==40H 时的操作 STA；经过此步骤，OP == 40H; M_NXT_ADDR == 0AH

（22）执行当前地址 M_ADDR==0AH 下的指令：PC→AR; PC+1;

（23）经过此步骤，PC == 06H; M_NXT_ADDR == 17H; AR == 05H

（24）执行当前地址 M_ADDR==17H 下的指令：RAM→AR; 此时 05H 地址的值为 10H，因此 AR 被赋值为 10H；经过此步骤，M_NXT_ADDR == 18H; AR == 10H

（25）执行当前地址 M_ADDR==18H 下的指令：R5→RAM; 此时 R5 的值为 DFH，因此 RAM 的 10E 位置被赋值为 DFH；经过此步骤，M_NXT_ADDR == 01H; RAM 中 地址为 10E 的值变为 DFH

（26）执行当前地址 M_ADDR==01H 下的指令：PC→AR; PC+1;

（27）经过此步骤，PC == 07H; M_NXT_ADDR == 02H; AR == 06H

## 5.读取并执行指令 OUT

（28）执行当前地址 M_ADDR==02H 下的指令：RAM→IR;功能跳转到 OP==60H 时的操作 OUT；经过此步骤，OP == 60H; M_NXT_ADDR == 0BH

（29）执行当前地址 M_ADDR==0BH 下的指令：PC→AR; PC+1;

（30）经过此步骤，PC == 08H; M_NXT_ADDR == 19H; AR == 07H

（31）执行当前步骤 M_ADDR==19H 下的指令：RAM→AR;此时 07H 地址的值为 10H，因此 AR 被赋值为 10H；经过此步骤，M_NXT_ADDR == 1AH; AR == 10H

（32）执行当前步骤 M_ADDR == 1AH 下的指令：RAM→BUS;此时 10H 地址的值为 DFH，因此向总线输出 DF；经过此步骤，M_NXT_ADDR == 01H

（33）执行当前步骤 M_ADDR == 01H 下的指令：PC→AR; PC+1;

（34）经过此步骤，PC = 09; M_NXT_ADDR == 02H; AR == 08H

```
PC:07 OP:60 M_ADDR:02 M_NXT_ADDR:0B
AR:06 DR1:55 DR2:8A R5:DF

PC:08 OP:60 M_ADDR:0B M_NXT_ADDR:19
AR:07 DR1:55 DR2:8A R5:DF

PC:08 OP:60 M_ADDR:19 M_NXT_ADDR:1A
AR:10 DR1:55 DR2:8A R5:DF

---OUT---   MEM[10]:DF

PC:08 OP:60 M_ADDR:1A M_NXT_ADDR:01
AR:10 DR1:55 DR2:8A R5:DF

PC:09 OP:60 M_ADDR:01 M_NXT_ADDR:02
AR:08 DR1:55 DR2:8A R5:DF
```

## 6.读取并执行指令 AND

```
PC:09 OP:E0 M_ADDR:02 M_NXT_ADDR:0F
AR:08 DR1:55 DR2:8A R5:DF

PC:0A OP:E0 M_ADDR:0F M_NXT_ADDR:1D
AR:09 DR1:55 DR2:8A R5:DF

PC:0A OP:E0 M_ADDR:1D M_NXT_ADDR:1E
AR:0F DR1:55 DR2:8A R5:DF

PC:0A OP:E0 M_ADDR:1E M_NXT_ADDR:1F
AR:0F DR1:55 DR2:F0 R5:DF

PC:0A OP:E0 M_ADDR:1F M_NXT_ADDR:07
AR:0F DR1:DF DR2:F0 R5:DF

PC:0A OP:E0 M_ADDR:07 M_NXT_ADDR:01
AR:0F DR1:DF DR2:F0 R5:D0

PC:0B OP:E0 M_ADDR:01 M_NXT_ADDR:02
AR:0A DR1:DF DR2:F0 R5:D0
```

（35）执行当前地址 M_ADDR==02H 下的指令：RAM→IR;功能跳转到 OP==E0H 时的操作 AND；经过此步骤，OP == E0H; M_NXT_ADDR == 0F

（36）执行当前地址 M_MXT_ADDR == 0F 下的指令：PC→AR; PC+1;

（37）经过此步骤，PC == 0AH; M_NXT_ADDR == 1DH; AR == 09H

（38）执行当前地址 M_ADDR==1DH 下的指令：RAM→AR; 此时 09H 地址的值为 0FH，因此 AR 被赋值为 0FH；经过此步骤，M_NXT_ADDR == 1EH; AR == 0FH

（39）执行当前地址 M_ADDR==1EH 下的指令：RAM→DR2; 此时 0FH 地址的值为 F0H，因此 DR2 被赋值为 F0H；经过此步骤，M_NXT_ADDR == 1FH; DR2 == F0H

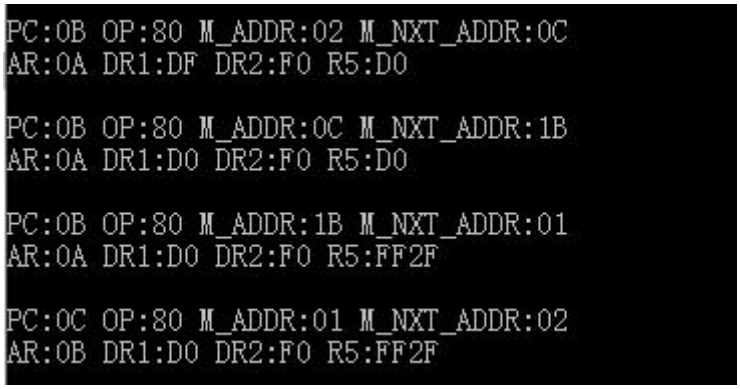
（40）执行当前地址 M_ADDR==1FH 下的指令:R5→DR1; 此时 R5 的值为 DFH,因此 DR1 被赋值为 DFH; 经过此步骤, M_NXT_ADDR == 07H; DR1 == DFH

（41）执行当前地址 M_ADDR==07H 下的指令: ALU→R5; 此时 ALU 进行 AND 运算后的的值为 D0H, 因此 R5 被赋值为 D0H; 经过此步骤, M_NXT_ADDR == 01H; R5 == D0H

（42）执行当前地址 M_ADDR==01H 下的指令: PC→AR; PC+1;

（43）经过此步骤, PC == 0BH; M_NXT_ADDR == 02H; AR == 0AH

## 7.读取并执行指令 COM




（44）执行当前地址 M_ADDR==02H 下的指令: RAM→IR;功能跳转到 OP==80H 时的操作 COM; 经过此步骤, OP == 80H; M_NXT_ADDR == 0C

（45）执行当前地址 M_ADDR==0CH 下的指令: R5→DR1,R5 的值为 D0H, 因此 DR1 被赋值为 D0H; 经过此步骤, M_MXT_ADDR == 1BH; DR1 == D0H

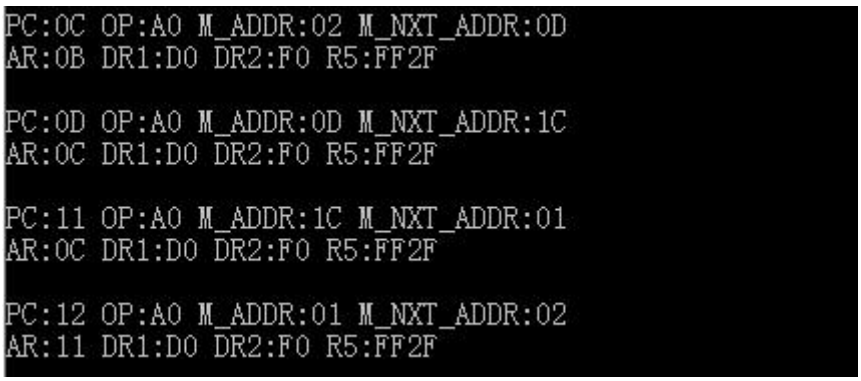
（46）执行当前地址 M_ADDR==1BH 下的指令: /ALUR5→R5,R5 的值变为 FF2FH

（47）经过此步骤, M_MXT_ADDR == 01H; R5 == FF2FH

（48）执行当前地址 M_ADDR==01H 下的指令: PC→AR; PC+1;

（49）经过此步骤, PC == 0CH; M_NXT_ADDR == 02H; AR == 0BH

## 8.读取并执行指令 JMP




（50）执行当前地址 M_ADDR==02H 下的指令: RAM→IR;功能跳转到 OP==A0H 时的操作 JMP; 经过此步骤, OP == A0H; M_NXT_ADDR == 0D

（51）执行当前地址 M_ADDR==0DH 下的指令: PC→AR; PC+1;

（52）经过此步骤, PC == 0DH; M_NXT_ADDR == 1CH; AR == 0CH

（53）执行当前地址 M_ADDR==1CH 下的指令：RAM→PC；此时 0CH 地址的值为 11H，因此 PC 被赋值为 11H；经过此步骤，PC == 11；M_NXT_ADDR == 01H

（54）执行当前地址 M_ADDR==01H 下的指令：PC→AR；PC+1；

（55）经过此步骤，PC == 12H；M_NXTADDR == 02H；AR == 11H

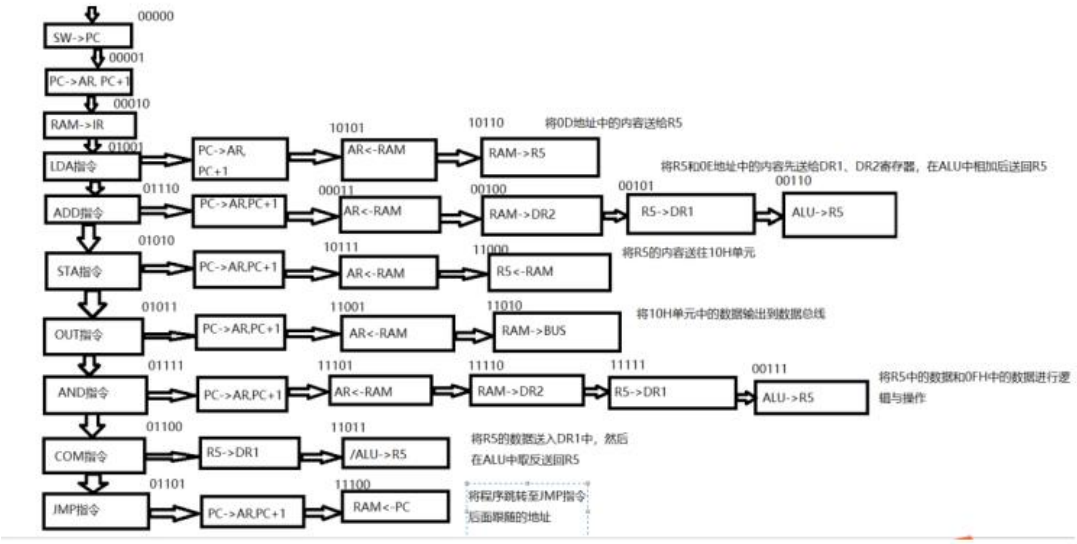
## 9.结束程序

```
PC:12 OP:00 M_ADDR:02 M_NXT_ADDR:08
AR:11 DR1:D0 DR2:F0 R5:FF2F

---OVER---
PC:12 OP:00 M_ADDR:08 M_NXT_ADDR:08
AR:11 DR1:D0 DR2:F0 R5:FF2F
```

（56）执行当前地址 M_ADDR==02H 下的指令：：RAM→IR；已无对应功能，程序结束

# 二，流程图




模拟器执行的指令是由上往下的执行顺序，横向顺序是每一条指令对应的具体机器操作，由于指令是在地址中连续存储的，所以模拟器也是连续执行的，每一条指令后面都有文字描述具体的操作

# 三，模拟器执行复合运算的最后结果截图

```
Memory:
00:20  01:0D  02:C0  03:0E  04:40  05:10  06:60  07:10
08:E0  09:0F  0A:80  0B:A0  0C:11  0D:55  0E:8A  0F:F0
10:DF  11:00  12:00  13:00  14:00  15:00  16:00  17:00
18:00  19:00  1A:00  1B:00  1C:00  1D:00  1E:00  1F:00
20:00  21:00  22:00  23:00  24:00  25:00  26:00  27:00
28:00  29:00  2A:00  2B:00  2C:00  2D:00  2E:00  2F:00
30:00  31:00  32:00  33:00  34:00  35:00  36:00  37:00
38:00  39:00  3A:00  3B:00  3C:00  3D:00  3E:00  3F:00
```

最后运行结果，将 ADD 指令计算出的值写入 10 地址，故 10:00->10:DF

# 四，ROM 模块的设计

说明：根据不同的指令需求来设置信号，如在 address 为“00001”时，要执行 PC→AR;PC+1;的指令，按照该需求分别设置 28 位信号，同时其接下来跳变为“00010”根据跳变来一步步设计信号，直到包括了所有的信号。

zjw_kongzhiqi.bdf | rom.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY rom IS
 PORT
(
      address  : IN     STD_LOGIC_VECTOR (4 DOWNTO 0);
               q  : OUT STD_LOGIC_VECTOR (27 DOWNTO 0));
END rom;
ARCHITECTURE SYN OF rom IS
 SIGNAL sub_wire0 : STD_LOGIC_VECTOR (27 DOWNTO 0);
BEGIN
 sub_wire0<=
 "101011110000000100000000001"   WHEN address=  "00000" ELSE
 "111111100000100100000000010"   WHEN address=  "00001" ELSE
 "100111110000010100000101000"   WHEN address=  "00010" ELSE
 "101011110000000100000010011"   WHEN address=  "01000" ELSE
 "111111100000100100000010101"   WHEN address=  "01001" ELSE
 "100111110000100100000101010"   WHEN address=  "10101" ELSE
 "100111111000000100000001000"   WHEN address=  "10110" ELSE
 "111111100000100100000010111"   WHEN address=  "01010" ELSE
 "100111110000100100000011000"   WHEN address=  "10111" ELSE
 "100110110000010000010000001"   WHEN address=  "11000" ELSE
 "111111100000100100000011001"   WHEN address=  "01011" ELSE
 "100111110000100100000101010"   WHEN address=  "11001" ELSE
 "100111110000000100000010001"   WHEN address=  "11010" ELSE
 "100110110001000100000011011"   WHEN address=  "01100" ELSE
 "100111011000001000000000001"   WHEN address=  "11011" ELSE
 "111111100000100100000011100"   WHEN address=  "01101" ELSE
 "101111110000000100000010001"   WHEN address=  "11100" ELSE
 "111111100000100100000000011"   WHEN address=  "01110" ELSE
 "100111110000100100000010001"   WHEN address=  "00011" ELSE
 "100111110010000100000010101"   WHEN address=  "00100" ELSE
 "100110110001000100000000110"   WHEN address=  "00101" ELSE
 "100111011000000110010000001"   WHEN address=  "00110" ELSE
 "111111100000100100000011101"   WHEN address=  "01111" ELSE
 "100111110000100100000101110"   WHEN address=  "11101" ELSE
 "100111100100001000001011111"   WHEN address=  "11110" ELSE
 "100110110001000100000000111"   WHEN address=  "11111" ELSE
 "100111011000001010110000001"   WHEN address=  "00111" ELSE
 "111111100000100100000010100"   WHEN address=  "10011" ELSE
 "100111110000000100000101011"   WHEN address=  "10100" ELSE
 "101011110000000100000010001"   WHEN address=  "10000" ELSE
 "111111100000100100000010010"   WHEN address=  "10001" ELSE
 "100011110000000100010100001"  ;
  q    <= sub_wire0(27 DOWNTO 0);
END SYN;
```

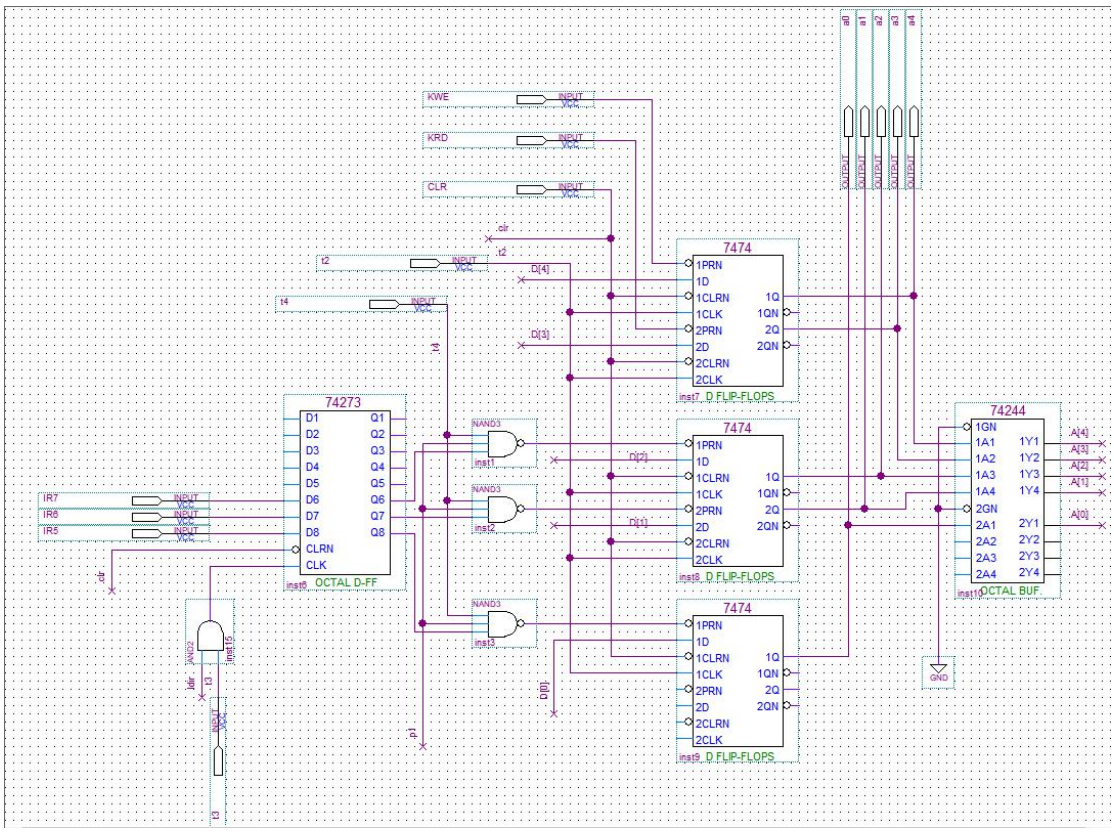
# 五，微程序控制器电路图截图

## 1.ROM 输出以下

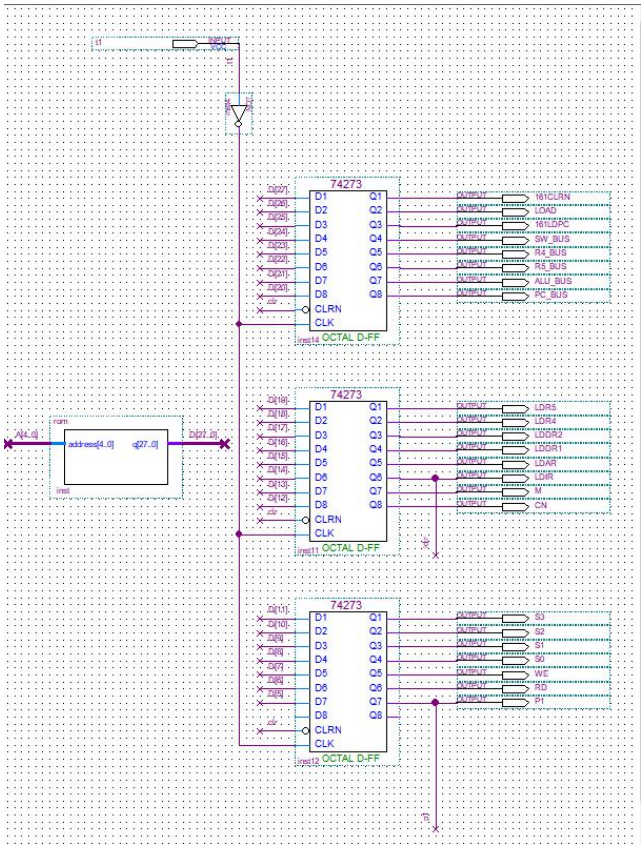
最左边的 74273 模块用于保存输入的 IR7,IR6,IR5，这三个输入在 001~111 个输入情况下分别表示 LDA,STA,OUT,COM,JMP,ADD,AND 等命令。74273 的时钟信号由节拍电路的 T3 和 LDIR 控制，即取指令。

74273 模块和右边的三个三输入与非门（这三个与非门为地址转移逻辑）将 IR7,IR6,IR5 送给中间的三个 7474 模块，这三个模块为微地址寄存器，会根据输入的指令以及输入的 ROM 输出的 4-0 为（即下址），将下址将下址送给后面的 74244 模块。微地址寄存器的时钟信号由节拍电路的 T2 控制，即给出下址。

ROM 左边的 74244 模块 8 位缓冲器，其作用是将左边三个 7474 送过来的下址转变为当前地址并送给 ROM。




2.ROM 输出以上




ROM 右边的三个 74273 模块为微命令寄存器，根据 ROM 输出的微指令来控制信号。

上面的 74273 控制 161CLRN,LOAD,161LDPC,PC_BUS,LDADR,WE,RD,SW_BUS。
中间的 74273 控制 LDRS,LDDR1,R5_BUS,M,S3,S2,S1,S0。
下面的 74273 控制 P1,LDIR,LDDR2,R4_BUS,LDR4,ALU_BUS,CN。
微命令寄存器的时钟部分为节拍电路的 T1，即给出当前微地址的微命令。

# 六，分析机器指令的执行过程

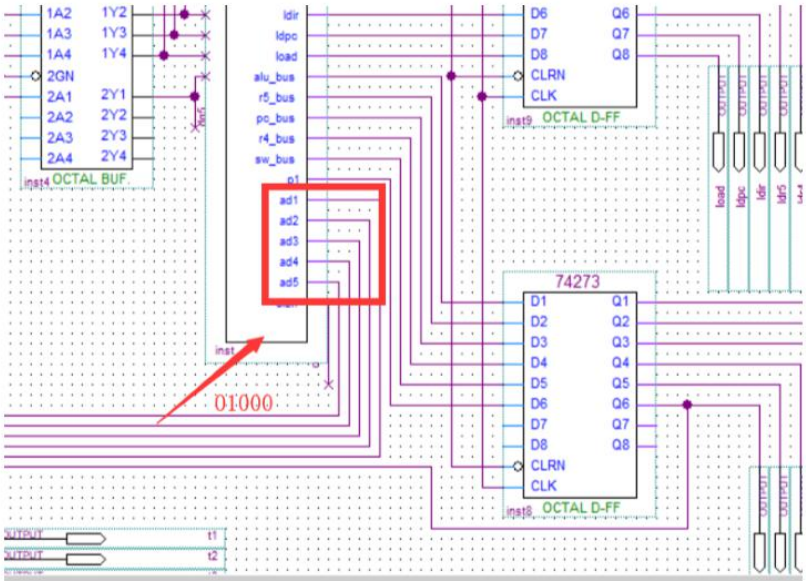
## 以 ADD 指令为例

## 1.

| 00010：RAM→IR，下址 01000 | | | | |
|---|---|---|---|---|
| 指令码 | 二进制 | IR7，IR6，IR5 | 入口地址 | 机器指令 |
| C0H | 11000000 | 110 | 01110 | ADD |

## 2. T1/T2

微指令寄存器即 3 个 34273 送出微指令 P1=1
绝对地址 01000

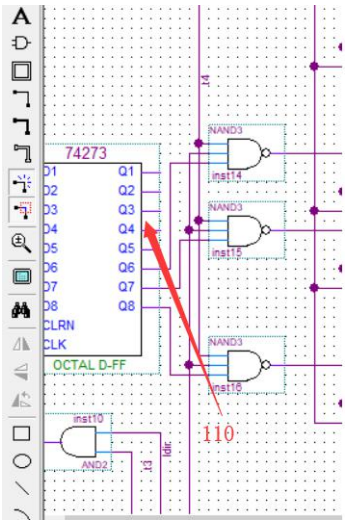



## 3. T2

微地址寄存器读取
绝对地址 01000

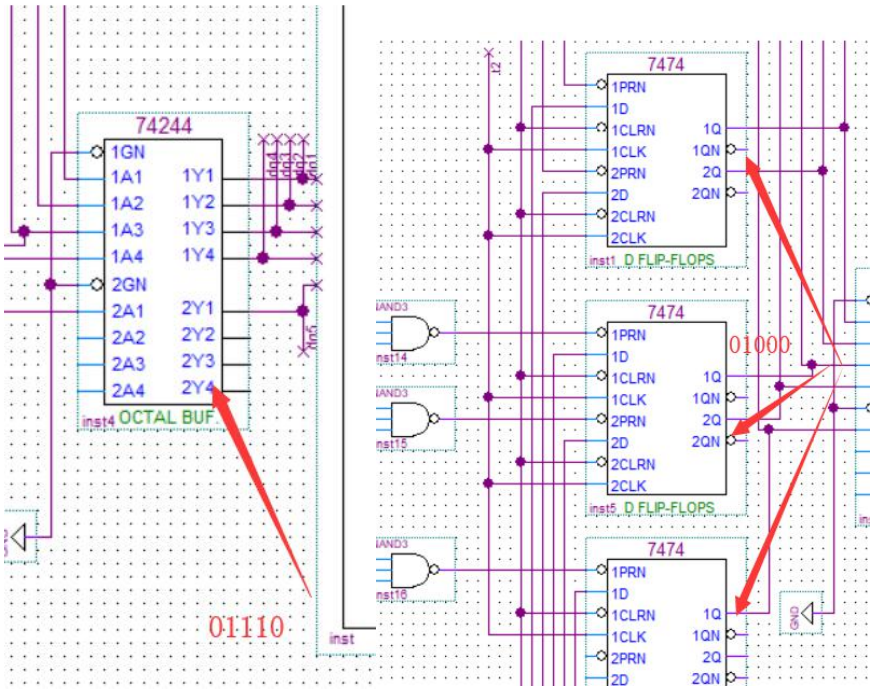
## 4. T3

由数据通路中的 RAM 读取 IR【7..5】到 IR (74273)




## 5. T4

产生新的下址到 74244 中 即从 01000 变为 01110 进入 ADD 操作的流程




具体分析

（1）构造初始化状态，使 CLR == 0

（2）启动时序电路，使 qd == 0

（3）执行 00000 对应的操作：SW→PC,并产生下址 00001 此时 SW_BUS==0

（4）执行 00001 对应的操作：PC→AR,PC+1,并产生下址 00010，此时 pc_BUS==0，LDAR == 1；PC_SEL == 111

（5）执行 00010 对应的操作：RAM→IR 跳转到了 LDA 指令，产生下址 01001

（6）执行 01001 对应的操作：PC→AR,PC+1,产生下址 10101

（7）执行 01001 对应的操作：AR→RAM,产生下址 10110