1. 模拟器执行情况

1) 初始化

```
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
10:00  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

---START---
PC:00 OP:00 M_ADDR:00 M_NXT_ADDR:00
AR:00 DR1:00 DR2:00 R5:00
```

2) 由上到下依次为:PC->AR,PC+1； RAM->IR

```
PC:01 OP:00 M_ADDR:01 M_NXT_ADDR:02
AR:00 DR1:00 DR2:00 R5:00

PC:01 OP:20 M_ADDR:02 M_NXT_ADDR:09
AR:00 DR1:00 DR2:00 R5:00
```

3) 执行 LDA 0DH 指令，将 0D 的数据 55 加载到 R5 中

```
PC:02 OP:20 M_ADDR:09 M_NXT_ADDR:15
AR:01 DR1:00 DR2:00 R5:00

PC:02 OP:20 M_ADDR:15 M_NXT_ADDR:16
AR:0D DR1:00 DR2:00 R5:00

PC:02 OP:20 M_ADDR:16 M_NXT_ADDR:01
AR:0D DR1:00 DR2:00 R5:55
```

依执行顺序，依次为:PC->AR,PC+1； RAM->AR； RAM->R5

4) 地址跳转到 00001，再次执行 2)

5) 执行 ADD 0EH 指令，将 0E 地址中的 8A 与 R5 相加得到 DF 送于 R5 中

```
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
```

依执行顺序，依次为:PC->AR,PC+1； RAM->AR； RAM->DR2； R5->DR1； DR1+DR2->R5

6) 地址跳转到 00001，再次执行 2)

7) 执行 STA 10H 指令，将 R5 的数据送到地址 10H 单元

```
PC:06 OP:40 M_ADDR:0A M_NXT_ADDR:17
AR:05 DR1:55 DR2:8A R5:DF

PC:06 OP:40 M_ADDR:17 M_NXT_ADDR:18
AR:10 DR1:55 DR2:8A R5:DF

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

PC:06 OP:40 M_ADDR:18 M_NXT_ADDR:01
AR:10 DR1:55 DR2:8A R5:DF
```

依执行顺序，依次为:PC->AR,PC+1；RAM->AR；R5->RAM

可以再存储单元里看出，地址 10H 单元的值已经写入了 R5 的值 DF

8) 地址跳转到 00001，再次执行 2)

9) 执行 OUT 10H 指令，将地址 10H 单元的数据输出到总线

```
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
```

依执行顺序，依次为:PC->AR,PC+1；RAM->AR；RAM->BUS，可从—OUT—一行得到输出结果

10) 地址跳转到 00001，再次执行 2)

11) 执行 AND 0FH 指令，将地址 0FH 单元的数据与 R5 数据相与，结果 D0 送入 R5

```
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
```

依执行顺序，依次为:PC->AR,PC+1；RAM->AR；RAM->DR2；R5->DR1；DR1 AND

DR2->R5

12) 地址跳转到 00001，再次执行 2）

13) 执行 COM 指令，将 R5 的数据取反，得到 FF2F

```
PC:0B OP:80 M_ADDR:0C M_NXT_ADDR:1B
AR:0A DR1:D0 DR2:F0 R5:D0

PC:0B OP:80 M_ADDR:1B M_NXT_ADDR:01
AR:0A DR1:D0 DR2:F0 R5:FF2F
```

依执行顺序，依次为:R5->DR1；NOT DR1->R5

14) 地址跳转到 00001，再次执行 2）

15) 执行 JMP 00H 指令，将地址转移到 00H 单元

```
PC:0D OP:A0 M_ADDR:0D M_NXT_ADDR:1C
AR:0C DR1:D0 DR2:F0 R5:FF2F

PC:11 OP:A0 M_ADDR:1C M_NXT_ADDR:01
AR:0C DR1:D0 DR2:F0 R5:FF2F
```

依执行顺序，依次为:PC->AR,PC+1；RAM->PC;

16) 地址跳转到 00001

```
PC:12 OP:A0 M_ADDR:01 M_NXT_ADDR:02
AR:11 DR1:D0 DR2:F0 R5:FF2F

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

08H 单元无操作，且下址为 08

17) 结束进程

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

由于篇幅问题，将原问题 3：模拟器执行复合运算最终结果+简单文字描述置于前

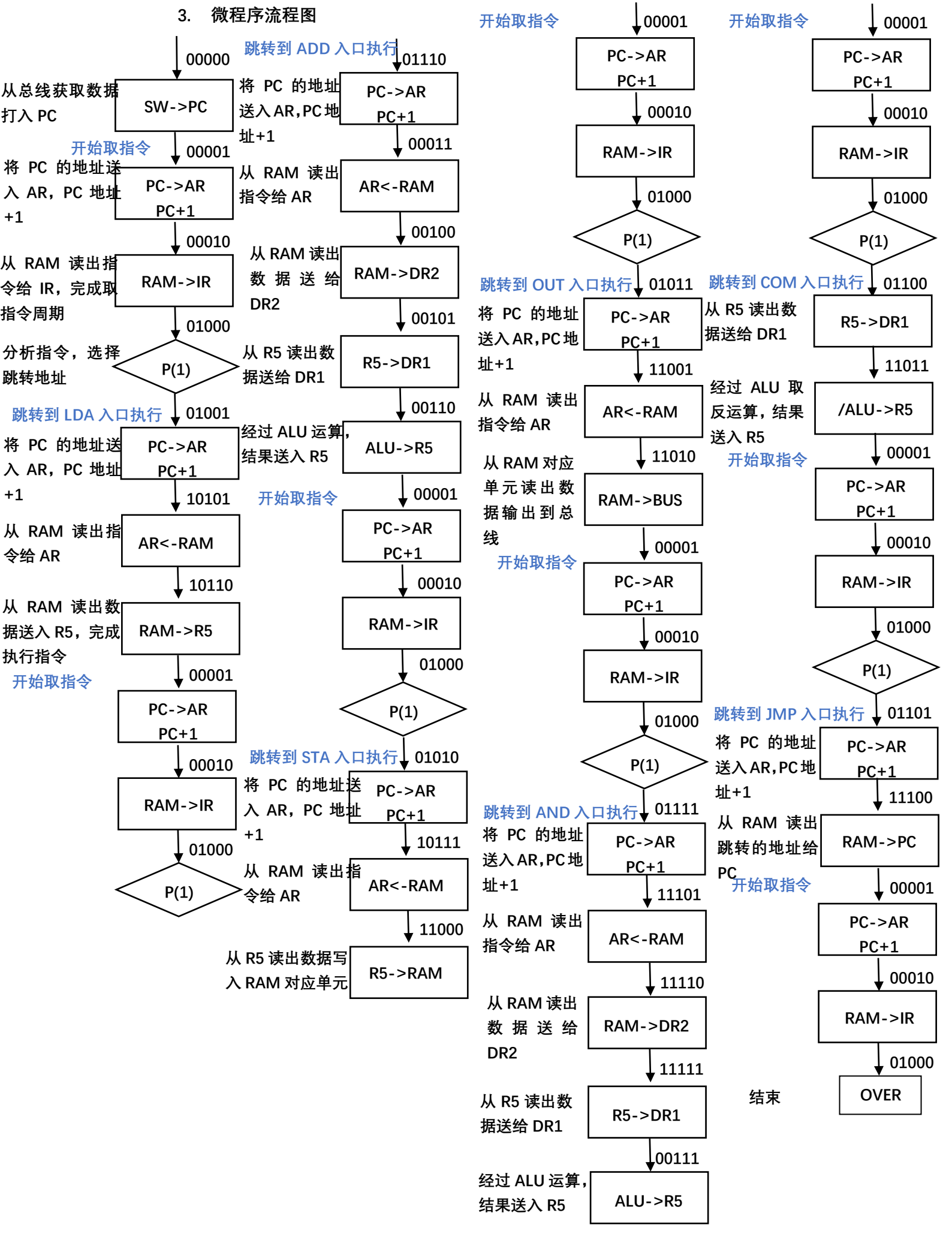
2. 模拟器执行复合运算最终结果

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

```
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
```

最终结果保存在 R5 内，复合运算为 NOT((55H 加 8AH)AND F0H)，最终结果为 FF2F

3. 微程序流程图


00000
从总线获取数据打入 PC
SW->PC
开始取指令
00001
将 PC 的地址送入 AR，PC 地址+1
PC->AR PC+1
00010
从 RAM 读出指令给 IR，完成取指令周期
RAM->IR
01000
分析指令，选择跳转地址
P(1)
跳转到 LDA 入口执行
01001
将 PC 的地址送入 AR，PC 地址+1
PC->AR PC+1
10101
从 RAM 读出指令给 AR
AR<-RAM
10110
从 RAM 读出数据送入 R5，完成执行指令
RAM->R5
开始取指令
00001
PC->AR PC+1
00010
RAM->IR
01000
P(1)
跳转到 ADD 入口执行
01110
将 PC 的地址送入 AR，PC 地址+1
PC->AR PC+1
00011
从 RAM 读出指令给 AR
AR<-RAM
00100
从 RAM 读出数据送给 DR2
RAM->DR2
00101
从 R5 读出数据送给 DR1
R5->DR1
00110
经过 ALU 运算，结果送入 R5
ALU->R5
开始取指令
00001
PC->AR PC+1
00010
RAM->IR
01000
P(1)
跳转到 STA 入口执行
01010
将 PC 的地址送入 AR，PC 地址+1
PC->AR PC+1
10111
从 RAM 读出指令给 AR
AR<-RAM
11000
从 R5 读出数据写入 RAM 对应单元
R5->RAM
开始取指令
00001
PC->AR PC+1
00010
RAM->IR
01000
P(1)
跳转到 OUT 入口执行
01011
将 PC 的地址送入 AR，PC 地址+1
PC->AR PC+1
11001
从 RAM 读出指令给 AR
AR<-RAM
11010
从 RAM 对应单元读出数据输出到总线
RAM->BUS
开始取指令
00001
PC->AR PC+1
00010
RAM->IR
01000
P(1)
跳转到 AND 入口执行
01111
将 PC 的地址送入 AR，PC 地址+1
PC->AR PC+1
11101
从 RAM 读出指令给 AR
AR<-RAM
11110
从 RAM 读出数据送给 DR2
RAM->DR2
11111
从 R5 读出数据送给 DR1
R5->DR1
00111
经过 ALU 运算，结果送入 R5
ALU->R5
开始取指令
00001
PC->AR PC+1
00010
RAM->IR
01000
P(1)
跳转到 COM 入口执行
01100
从 R5 读出数据送给 DR1
R5->DR1
11011
经过 ALU 取反运算，结果送入 R5
/ALU->R5
开始取指令
00001
PC->AR PC+1
00010
RAM->IR
01000
P(1)
跳转到 JMP 入口执行
01101
将 PC 的地址送入 AR，PC 地址+1
PC->AR PC+1
11100
从 RAM 读出跳转的地址给 PC
RAM->PC
开始取指令
00001
PC->AR PC+1
00010
RAM->IR
01000
结束
OVER

先进行初始化，之后 PC 数据送到 AR 后 PC+1，从 RAM 中读取数据送到指令寄存器 IR，在 P1=1 时依据 IR 的高三位来选择微操作的入口，第一次选择了 LDA 操作，进入 LDA 操作后，依次执行各项微操作，最后一项微操作执行完后下址给为 00001；

再次重复执行 PC 数据送入 AR，PC+1，从 RAM 中读取数据送到指令寄存器 IR，并在 P1 作用下选择新的微操作的入口，执行第二个微操作 ADD；

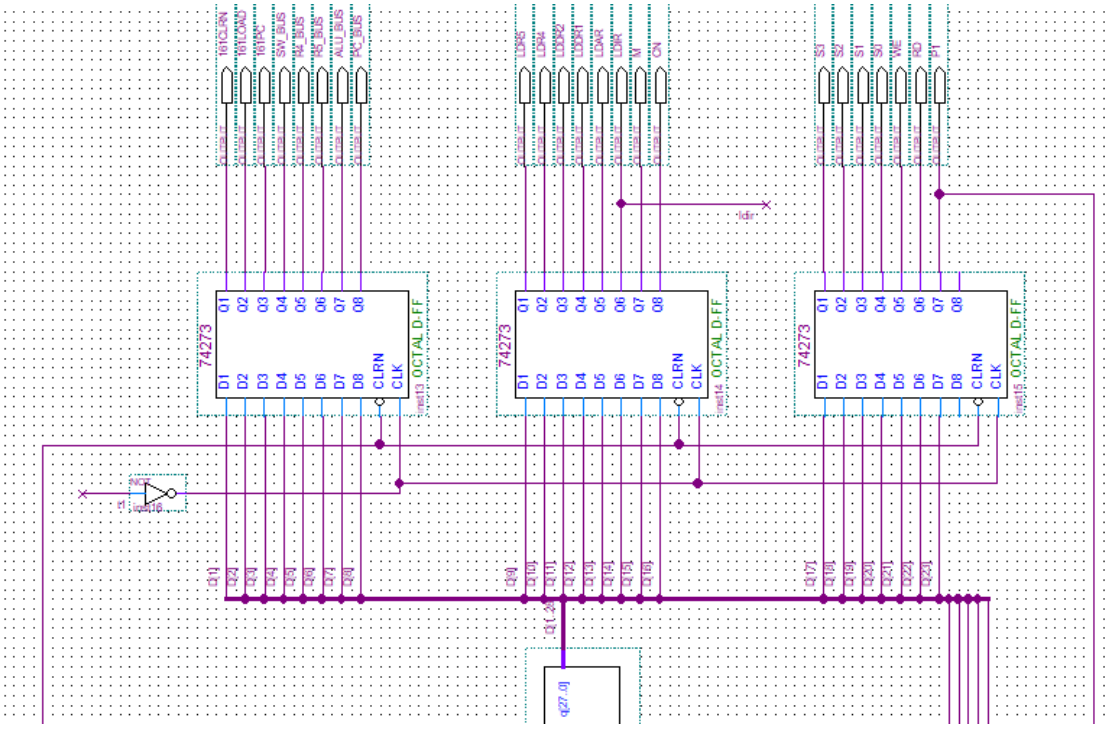
之后步骤依次重复执行完 STA，OUT，AND，COM，由 JMP 转移到 00H 地址，最终进入 01000 强读程序，结束进程。

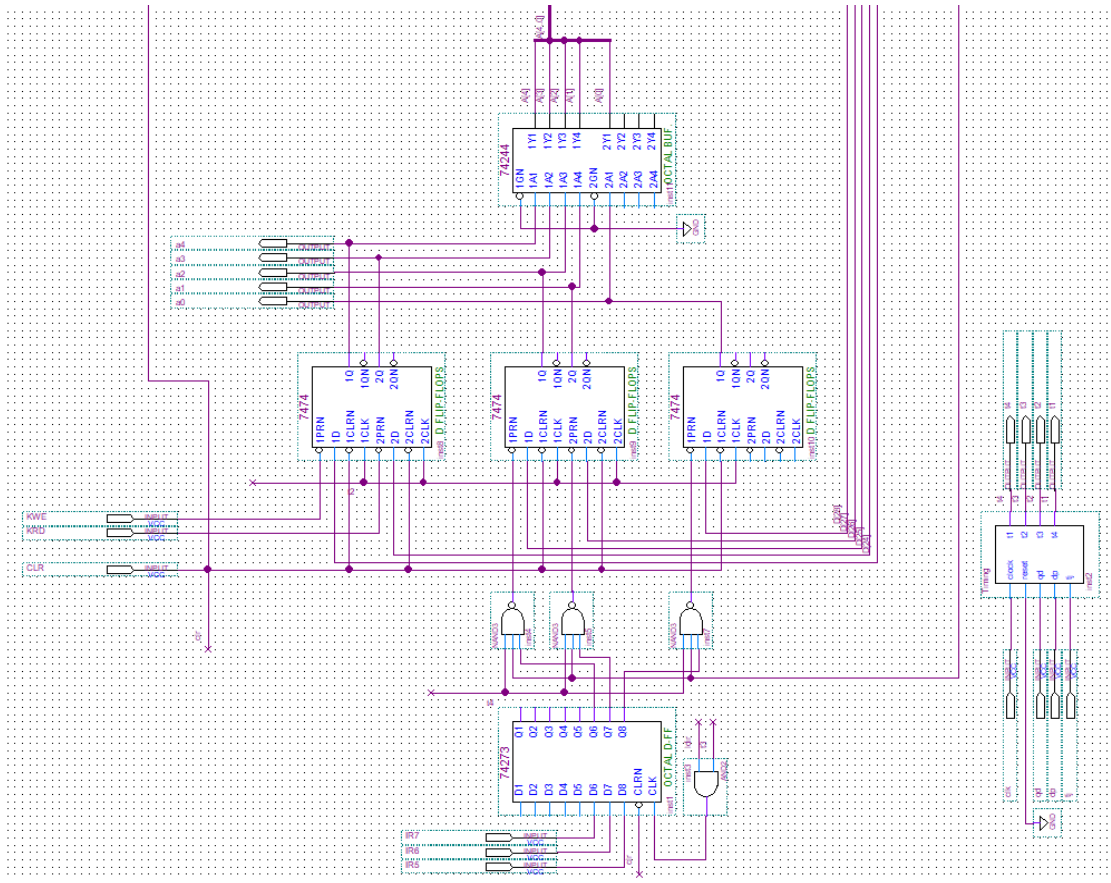
4. ROM 模块的设计

```
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<=
"101011110000000000000000001"   WHEN address=  "00000" ELSE
"111111100001000000000000010"   WHEN address=  "00001" ELSE
"100111110000010000000011010000"   WHEN address=  "00010" ELSE
"111111100001000000000010101"   WHEN address=  "01001" ELSE
"100111110000100000000010110"   WHEN address=  "10101" ELSE
"100111111000000000000001000001"   WHEN address=  "10110" ELSE
"111111100001000000000010111"   WHEN address=  "01010" ELSE
"100111110000100000000011000"   WHEN address=  "10111" ELSE
"100110110000000000010000001"   WHEN address=  "11000" ELSE
"111111100001000000000011001"   WHEN address=  "01011" ELSE
"100111110000100000000011010"   WHEN address=  "11001" ELSE
"100111110000000000000001000001"   WHEN address=  "11010" ELSE
"100110110001000000000011011"   WHEN address=  "01100" ELSE
"100111011000001000000000001"   WHEN address=  "11011" ELSE
"111111100001000000000011100"   WHEN address=  "01101" ELSE
"101111110000000000000001000001"   WHEN address=  "11100" ELSE
"111111100001000000000000011"   WHEN address=  "01110" ELSE
"100111110000100000000001000100"   WHEN address=  "00011" ELSE
"100111110010000000000001000101"   WHEN address=  "00100" ELSE
"100110110001000000000000000110"   WHEN address=  "00101" ELSE
"100111011000000011001000000001"   WHEN address=  "00110" ELSE
"111111100001000000000011101"   WHEN address=  "01111" ELSE
"100111110000100000000010111110"   WHEN address=  "11101" ELSE
"100111110010000000000010111111"   WHEN address=  "11110" ELSE
"100110110001000000000000000111"   WHEN address=  "11111" ELSE
"100111011000001011110000000001"   WHEN address=  "00111" ELSE
"101011110000000000000000010001"   WHEN address=  "10000" ELSE
"111111100001000000000010010"   WHEN address=  "10001" ELSE
"100011110000000000010010001"   WHEN address=  "10010" ELSE
"101011110000000000000000010011"   WHEN address=  "01000" ELSE
"111111100001000000000010100"   WHEN address=  "10011" ELSE
"100111110000000000000001010011"  ;
 q    <= sub_wire0(27 DOWNTO 0);
END SYN;
```

通过填写“5-总结构图&微代码表 VHDL&时序信号&微指令 RTL.xls”表中的信号，将生成的代码中 YH 改为“完成 vhdl 的编写，其中，各个控制信号的顺序依次为 161CLR、161LD、161PC、SW_BUS、R4_BUS、R5_BUS、ALU_BUS、PC_BUS、LDR5、LDR4、LDDR2、LDDR1、LDAR、LDIR、M、CN、S3、S2、S1、S0、WE、RD、P1。

5. 微程序控制器电路图





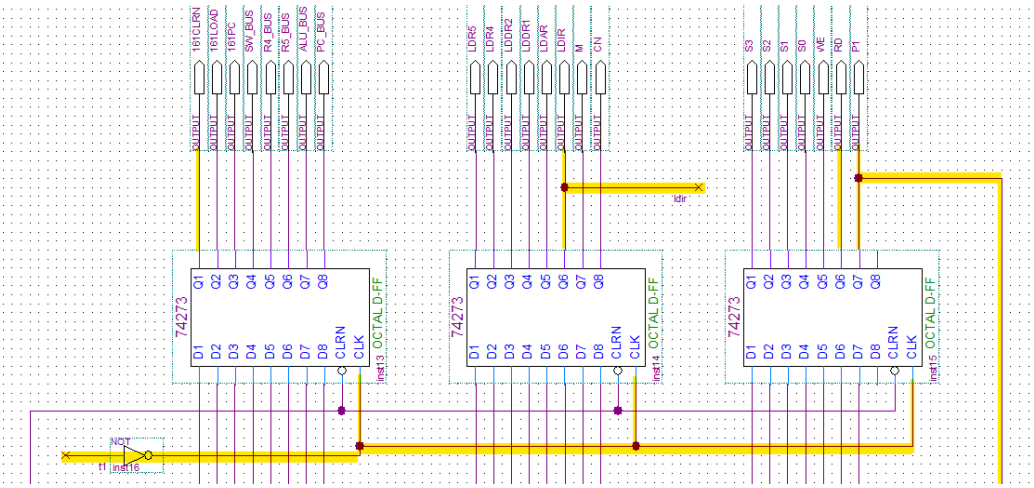
① ROM 的输出以上部分：其中各控制信号的顺序为上文中在 vhdl 中的顺序

② ROM 的输出以下部分：右下角为时序电路部分信号由左到右依次为 T4，T3，T2，T1。整图基本按照给定电路图设计。

6. 举例分析机器指令执行情况

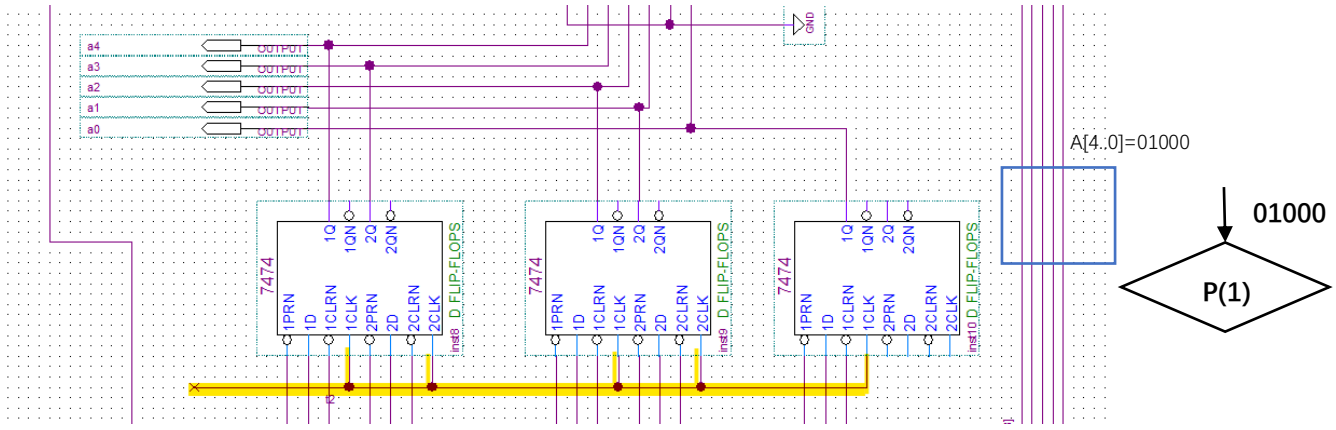
以 AND 为例

T1：给出当前微地址的微命令



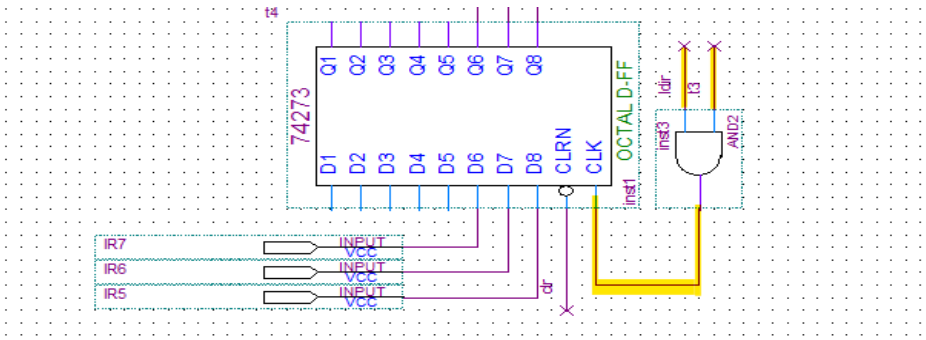
通过微命令寄存器 74273*3 送出的有效信号为 161CLRN，LDIR，P1

T2：给出下址



从 RAM 读出下一当前地址的指令输送到总线；同时微地址寄存器 7474*3 送出绝对地址 01000

T3：取指令



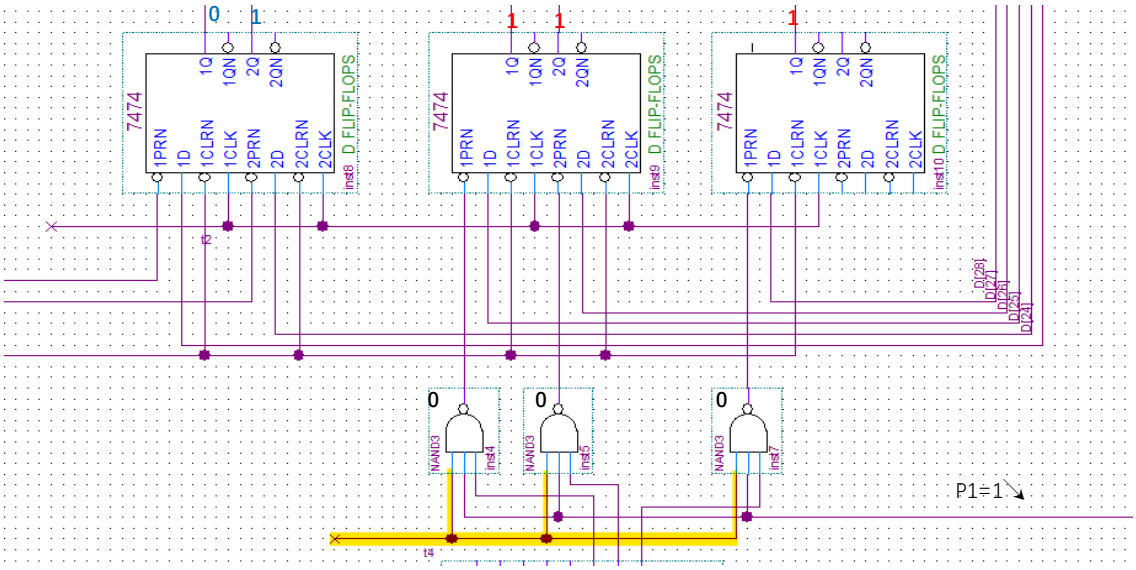
AR 从总线上读取数据，PC+1.

之前送出的 LDIR 为 1，当 T3 到达时，74273 的 clk 为 1 有效，并由数据通路的 RAM 读取 IR[7..0]

其中 IR7，IR6，IR5 为 111

经指令寄存器 74273 送出

T4：译码产生新下址



由于三个 NAND3 的输入端均为 1，则输出 0，送到 7474 中，对应的 PRN 输入端为低电平有效，将数据置 1，如图红字。最左的 7474 给出 01，组合成新的下址 01111 送入 74244，产生了 AND 的入口。

以上，完成取指令周期，跳转到地址 01111

接下来执行 AND

01111： T1：通过当前地址 01111 译出各个控制信号，送到数据通路

T2：给出下址 11101

T3：PC 的地址装载给 AR，PC+1

T4：由于 P1 无效，经最下方 7474*3 送出新下址 11101

11101： T1：由当前地址 11101 译出各个控制信号送到数据通路

T2：从 RAM 读出数据送入总线，给出下址 11110

T3：AR 从总线读入数据

T4：由于 P1 无效，直接给出新下址 11110

11110： T1：由当前地址 11110 译出各个控制信号送到数据通路

T2：从 RAM 读出数据送入总线，给出下址 11111，DR2 从总线接受数据

T3：LDIR 无效 74273 未译出新下址

T4：由于 P1 无效，直接给出新下址 11111

11111： T1：由当前地址 11111 译出各个控制信号送到数据通路，R5 输出数据到总线

T2：给出下址 00111

T3：DR2 从总线读入 R5 之前输出到总线数据

T4：由于 P1 无效，直接给出新下址 00111

00111： T1：由当前地址 00111 译出各个控制信号送到数据通路，ALU 进行运算并输出到总线

T2：给出下址 00001

T3：R5 从总线接受 ALU 的运算结果

T4：由于 P1 无效，直接给出新下址 00001