山东大学
SHANDONG UNIVERSITY

《计算机组成与设计》
课程设计报告

课程设计题目：微程序实现模型设计

班　　级：2020 级 3 班

姓　　名：李云飞

学　　号：202000130083

完 成 日 期：2022 年 12 月 11 日

目录

一. 指令组

1. 指令组及其功能

1) 读指令

读指令指令通过将 RAM 中的指令读入 IR 中，利用 QJP 微操作跳转到相应的指令入口。QJP 将 uPC 高四位设置为 IR 的高四位，低四位默认为 0，即可完成相应的跳转，例：如果读指令的入口在 10，当 IR 为 1x 时（x 为任意），uPC 执行 QJP 时会运行到 10 入口，从而执行读指令的操作。

2) Load R0 -> RAM

读取 RAM 数据到 R0 寄存器中。首先将 PC 地址送入 MAR，然后 PC 自加一，RAM 将 MAR 地址中的数据送给 R0，PC 地址再送到 MAR，最后进行 JP，跳转到读指令。

3) Load R1 -> RAM

读取 RAM 数据到 R1 寄存器中。首先将 PC 地址送入 MAR，然后 PC 自加一，RAM 将 MAR 地址中的数据送给 R1，PC 地址再送到 MAR，最后进行 JP，跳转到读指令。

4) **Add R0 + R1 -> RAM**

将 R0 和 R1 相加后存入 RAM。首先将 PC 地址送入 MAR，然后 PC 自加一，从 RAM 中读取一个地址送入 MAR，该地址即为得数需要存入的地址，然后将 R0 和 R1 中的数据送到 ALU 进行运算相加 R0 + R1，将运算结果送入 RAM 中（MAR）地址，最后将 PC 地址重新送入 MAR。

5) **Sub R0 - R1 -> RAM**

将 R0 和 R1 相减（R0 - R1）后存入 RAM。首先将 PC 地址送入 MAR，然后 PC 自加一，从 RAM 中读取一个地址送入 MAR，该地址即为得数需要存入的地址，然后将 R0 和 R1 中的数据送到 ALU 进行运算相减，R0 – R1，将运算结果送入 RAM 中（MAR）地址，最后将 PC 地址重新送入 MAR。

6) **Mul R0 * R1 -> RAM**

将 R0 和 R1 进行乘法运算，后存入 RAM。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入 MAR，用于决定计算结果存入的地址。然后将 R0 和 R1 送入阵列乘法器进行乘法运算，将运算结果 R0 * R1 送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR，用于下一个操作。

7) **And R0 & R1 -> RAM**

将 R0 和 R1 进行与运算 R0 & R1，将结果存入 RAM。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入 MAR，用于决定计算结果存入的地址。然后将 R0 和 R1 送入 ALU 进行与运算，将运算结果 R0 & R1 送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR，用于下一个操作。

8) **Or R0 | R1 -> RAM**

将 R0 和 R1 进行或运算 R0 | R1，将结果存入 RAM。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入 MAR，用于决定计算结果存入的地址。然后将 R0 和 R1 送入 ALU 进行或运算，将运算结果 R0 | R1 送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR，用于下一个操作。

9) **NAND ~(R0 & R1) -> RAM**

将 R0 和 R1 进行与非运算~（R0 & R1），将结果存入 RAM。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入 MAR，用于决定计算结果存入的地址。然后将 R0 和 R1 送入 ALU 进行与非运算，将运算结果~（R0 & R1）送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR，用于下一个操作。

10) **XOR R0 ^ R1 -> RAM**

将 R0 和 R1 进行异或运算 R0 ^ R1，将结果存入 RAM。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入

MAR，用于决定计算结果存入的地址。然后将 R0 和 R1 送入 ALU 进行异或运算，将运算结果 R0 ^ R1 送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR，用于下一个操作。

11) Adds R0 + 1 -> RAM

将 R0 寄存器中的数据进行加 1，存入 RAM。该操作也可以将结果送入 R0，从而完成 R0 的自加一。之所以送入 RAM 是为了方便验证，可以直接查看 RAM 进行验证。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入 MAR，用于决定计算结果存入的地址。然后将 R0 送入 ALU 进行加 1 运算，将运算结果 R0+1 送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR。

12) Subs R0 – 1 -> RAM

将 R0 寄存器中的数据进行减 1，存入 RAM。该操作也可以将结果送入 R0，从而完成 R0 的自加一。之所以送入 RAM 是为了方便验证，可以直接查看 RAM 进行验证。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入 MAR，用于决定计算结果存入的地址。然后将 R0 送入 ALU 进行减 1 运算，将运算结果 R0 -1 送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR。

13) Nots ~R0 -> RAM

将 R0 中的数据取反，存入 RAM。将输出端设置为 R0 即可实现自反。首先将 PC 的地址输入到 MAR，然后 PC 自加一待定。从 RAM 中读取一个地址，送入 MAR，用于决定计算结果存入的地址。然后将 R0 送入 ALU 进行取反运算，将运算结果~R0 送到 RAM 中 MAR 地址处，最后将 PC 地址重新送入 MAR。

14) Lsr R0 <- 1 -> RAM

将 R0 中的数据左移 1 位，送到 RAM。通过将输入改为 R0 即可完成 R0 的左移。首先将 PC 地址送到 MAR，用与取出需要存放的地址，然后将 RAM 中的 MAR 地址处的数据取出存入 MAR，即为左移后存入 RAM 的地址。将 R0 中的数据送到移位器进行左移，将得到的结果送入 RAM 相应的位置。将 MAR 重新设置为 PC 的地址。

15) Mov @RAM

间接寻址，将 R0 送到间接寻址得到的地址。首先将 PC 送入 MAR，RAM 从 MAR 地址得到第一个地址 d1，将 d1 送到 MAR 后，再选择 d1 地址中的数据，将数据再次送入 MAR，即为间接寻址得到的地址。简而言之即执行两次 RAM->MAR 即可。

16）HALT

停机指令，即 IR3 为 1，停止启动器。

2. 指令格式

指令基本字长为 8 位，一条微指令由 3 个字长，即 24 位构成，即 uIR_{23}~uIR_0。

其中

uIR_{23} 和 uIR_{22} 分别为乘法和移位运算符。

uIR_{21}~uIR_{16} 为运算器控制位，分别对应 ALU 的输入端 M，S_3，S_2，S_1，S_0，C_0。通过对 uIR_{23}~uIR_{16} 的控制，来决定不同的运算。

$uIR_{15,14}$ 决定 ALU 的输入数据 a 是来自 RAM 还是来自 R0。10 表示为 R0，01 表示为 RAM。$uIR_{15,14}$ 经过 2-4 译码器后可以扩展到 4 位，最多可以选择 4 个寄存器。

$uIR_{13,12}$ 决定 ALU 的输入数据 b 是来自 R1 还是来自 PC。10 表示为 PC，01 表示为 R1。$uIR_{15,14}$ 经过 2-4 译码器后可以扩展到 4 位，最多可以选择 4 个寄存器。

$uIR_{11,\ 10,9}$ 决定 ALU 输出端。$uIR_{11\sim9}$ 经过 3-8 译码器后可以决定 8 个输出端，经过 3-8 译码后，1 代表输出到 R0，2 代表输出到 R1，3 代表输出到 PC，4 代表输出到 IR，5 代表输出到 MAR。

uIR_8 决定移位操作时移位的方向。

$uIR_{7\sim5}$ 待定。

uIR_4 决定 RAM 的读和写。1 写 0 读。

uIR_3 为停机标识，1 标识停机，停止启动器。

$uIR_{2\sim0}$ 为后续地址形成方式。经过 3-8 译码器后可以为 8 种地址生成方式。其中 1 为 PC+1 顺序执行，2 位 JP 为无跳转转移，根据 $uIR_{15\sim8}$ 跳转，3 位 QJP，根据操作码 IR 的高 4 位跳转，低 4 位默认为 0,4 为 YJP，给定高 4 位，低 4 位按照源寻址方式进行转移，5 为 MJP，给定高 4 位，低 4 位按照目寻址方式转移。

操作码基础为 6 位，即 ALU 的输入端对应的 $uIR_{21\sim16}$，在进行乘法或者移位操作时，会判断 $uIR_{23,22}$ 以及 uIR_8，后续地址生成 $uIR_{2\sim0}$ 也会影响指令的执行，因此也可以认为操作码为 6+3+3=12 位。地址码基础为 7 位，即 A 选择地址 2 位，B 选择地址 2 位，以及输出地址 3 位。如果跳转为 JP，则 $uIR_{15\sim8}$ 为地址码，为 8 位。并且 RAM 的地址也为 8 位，因此也可以将地址码看作为 8 位。

3. 操作码

1) RAM->IR

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 0011 1110 0 001，执行直传 A 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 10 00 100，表示 A 输入为 RAM，B 无输入，输出为 IR 寄存器。

2) RAM->R0

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 0011 1110 0 001，执行直传 A 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 10 00 001，表示 A 输入为 RAM，B 无输入，输出为 R0 寄存器

3) RAM->R1

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 0011 1110 0 001，执行直传 A 操作, 下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 10 00 010，表示 A 输入为 RAM，B 无输入，输出为 R1 寄存器

4) PC->MAR

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 0011 0100 0 001，执行直传 B 操作, 下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 00 01 101，表示 A 无输入，B 输入为 PC，输出为 MAR 寄存器

5) PC+1->PC

操作码 $uIR_{23\sim16,8,\ 2\sim0}$ 为 0001 0010 0 001，执行 B+1 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 00 01 011，表示 A 无输入，B 输入为 PC，输出为 PC

6) R0+R1->RAM

操作码 $uIR_{23\sim16,8,\ 2\sim0}$ 为 0001 0011 0 001，执行 A+B 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 01 10 000，RAM 读写标志位 1，表示 A 输入为 R0，B 输入为 R1，输出写入到 RAM

7) R0-R1->RAM

操作码 $uIR_{23\sim16,8,\ 2\sim0}$ 为 0000 1100 0 001，执行 A-B 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 01 10 000，RAM 读写标志位 1，表示 A 输入为 R0，B 输入为 R1，输出写入到 RAM

8) R0*R1->RAM

操作码 $uIR_{23\sim16,8,\ 2\sim0}$ 为 0100 0000 0 001，执行 A*B 操作。uIR_{22} 为 1 表示为乘法运算，数据送入阵列乘法器，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 01 10 000，RAM 读写标志位 1，表示 A 输入为 R0，B 输

入为 R1，输出写入到 RAM

9) R0&R1->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$为 0011 0110 0 001，执行 A&B 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$为 01 10 000，RAM 读写标志位 1，表示 A 输入为 R0，B 输入为 R1，输出写入到 RAM

10) R0|R1->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$为 0011 1100 0 001，执行 A|B 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$为 01 10 000，RAM 读写标志位 1，表示 A 输入为 R0，B 输入为 R1，输出写入到 RAM

11) ~(R0+R1)->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$为 0010 0010 0 001，执行~（A+B）操作, 下地址为顺序执行

地址码 $uIR_{15\sim9}$为 01 10 000，RAM 读写标志位 1，表示 A 输入为 R0，B 输入为 R1，输出写入到 RAM

12) R0^R1->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$为 0010 1100 0 001，执行 A^B 操作, 下地址为顺序执行

地址码 $uIR_{15\sim9}$为 01 10 000，RAM 读写标志位 1，表示 A 输入为 R0，B 输入为 R1，输出写入到 RAM

13) R0+1->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$为 0000 0000 0 001，执行 A+1 操作, 下地址为顺序执行

地址码 $uIR_{15\sim9}$为 01 00 000，RAM 读写标志位 1，表示 A 输入为 R0，B 无输入，输出写入到 RAM

14) R0-1->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$为 0001 1111 0 001，执行 A-1 操作, 下地址为顺序执行

地址码 $uIR_{15\sim9}$为 01 00 000，RAM 读写标志位 1，表示 A 输入为 R0，B 无输入，输出写入到 RAM

15) ~R0->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$为 0010 0000 0 001，执行~A 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 01 00 000，RAM 读写标志位 1，表示 A 输入为 R0，B 无输入，输出写入到 RAM

16) (R0<-1)->RAM

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 1000 0000 1 001，执行 A<- 1 操作. 下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 01 00 000，RAM 读写标志位 1，表示 A 输入为 R0，B 无输入，输出写入到 RAM

17) RAM->MAR,RAM->MAR

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 0011 1110 0 001，执行直传 A 操作，下地址为顺序执行

地址码 $uIR_{15\sim9}$ 为 10 00 101，表示 A 输入为 R0，B 无输入，输出写入到 MAR。因为经过了两次运行，可以实现间接寻址。第一个 RAM 得到的数据为数据地址所在的地址，第二次访问地址时得到的才是数据真正需要传输的地址。

18) QJP

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 0000 0000 0 011，ALU 不工作，下地址为 IR 的高四位 + 0000

地址码 $uIR_{15\sim9}$ 为 00 00 000，没有输入输出

19) JP

操作码 $uIR_{23\sim16,8,2\sim0}$ 为 0000 0000 0 010，ALU 不工作，下地址为 $uIR_{15\sim8}$，即跳转到 00 地址

地址码 $uIR_{15\sim9}$ 为 00 00 000，没有输入输出

二． 总体结构与数据通路

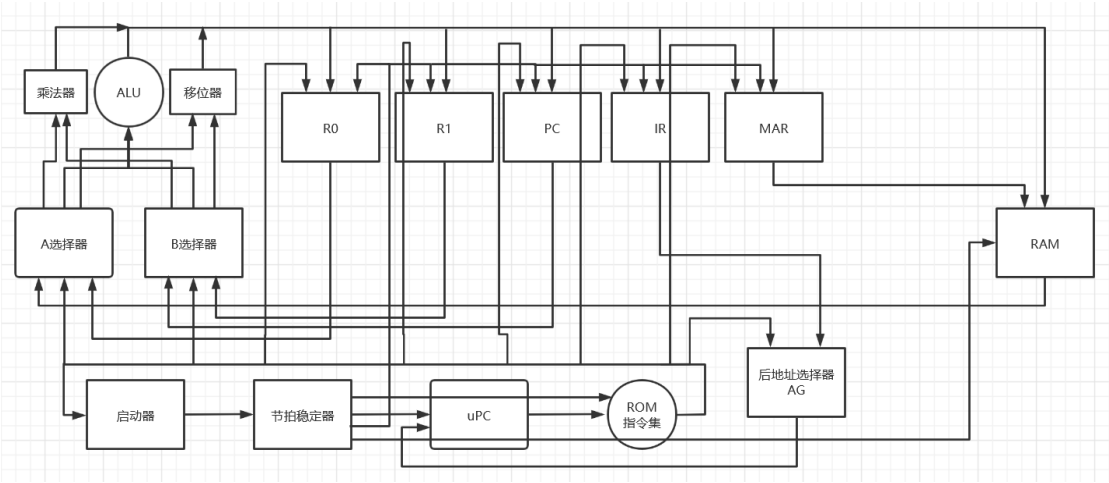


图 2.0 总体结构框图

1. 启动器

启动器，功能是通过输入选择手动输入、脉冲输入、开关以及停止，并且输出脉冲信号。作用是用于控制整个机器的运行和停止，发射脉冲信号决定模型运行的频率。

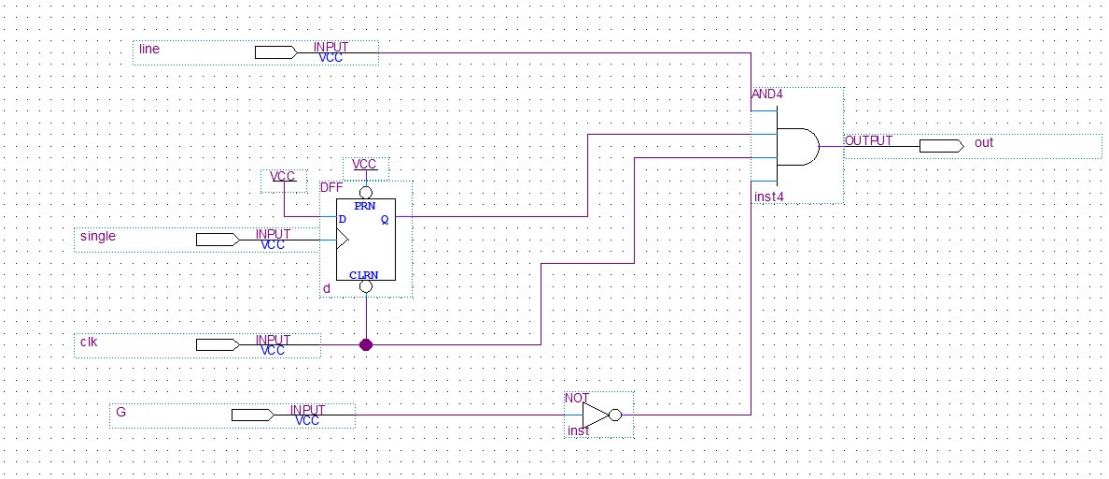


图 2.1 启动器原理图

当 G 输入为 0 时，结果才可能为 1，因此只要 G 端有输入，输出端就必定为 0，即没有输出，实现停机。clk 为总开关，为 1 时才可能启动。Single 为单脉冲信号，用于手动调试，调用 D 触发器，保存。Line 为时钟脉冲输入端，用于程序自动执行。

2. 节拍稳定器

节拍稳定器可以将 4 个输出按照 J0,J1,J2,J3 的顺序依次送出，作用是使不同器件按照顺序执行，避免因为数据在通路上的传输导致数据时钟紊乱，预防可能出现的数据接收错误。按照输出控制、PC、ROM、RAM 的顺序依次进行运行，保证每个器件运行时都能得到当前周期内的正确数据。

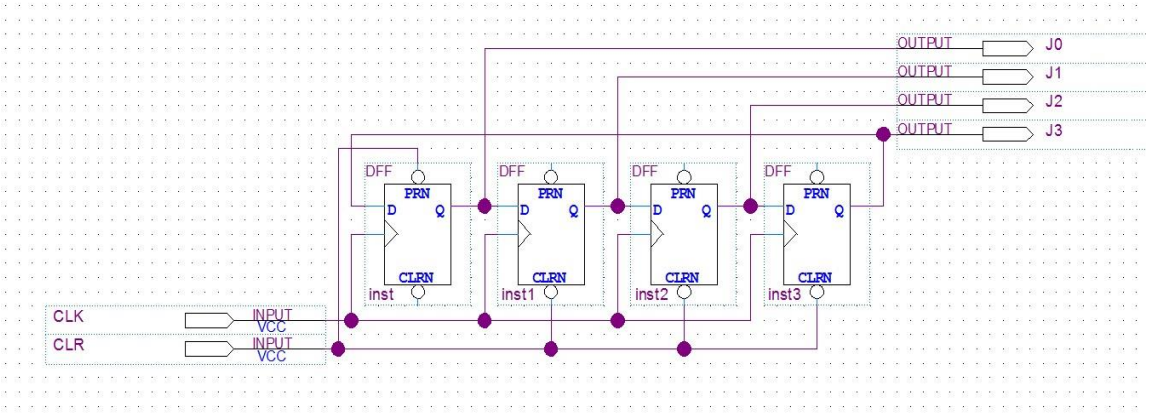


图 2.2 节拍稳定器

CLK 输入通过 4 个 D 触发器，并且在每个触发器之后分别进行输出，保证了 4 个输出为顺序输出的。

3. 指令计数器 PC

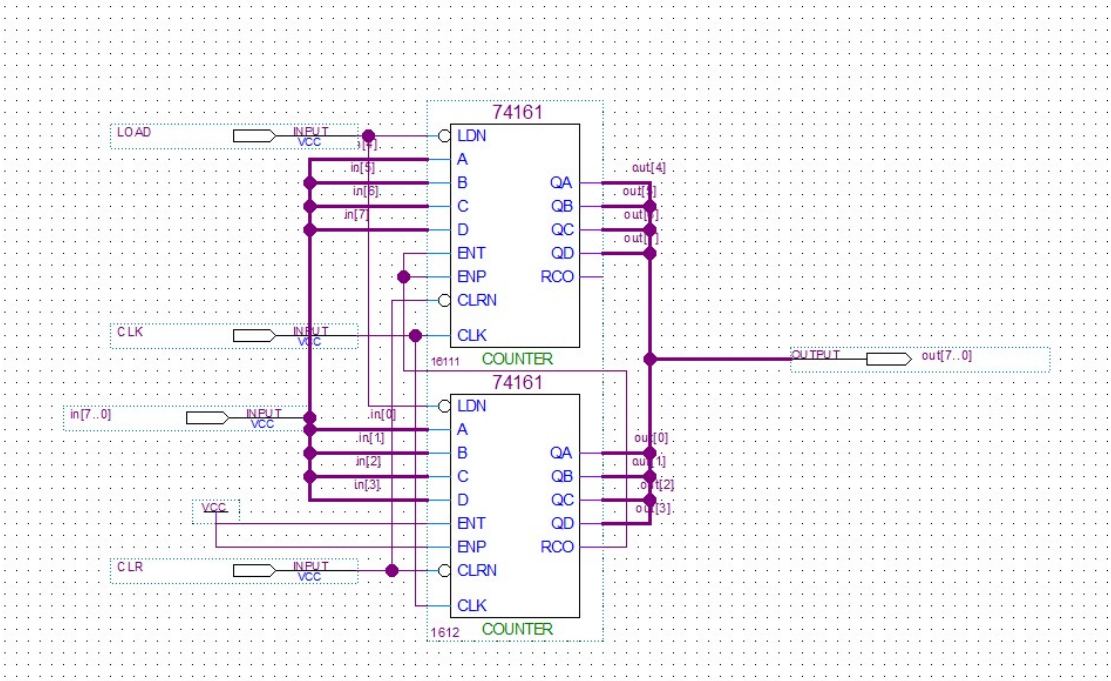


图 2.3 PC

指令计数器 PC 可以在有 CLK 输入时，对 8 位数据进行加 1 处理，或者在 LOAD 有输入时，将数据置换为 in[7..0]的数据并进行输出。作用是可以每次对

自身数据加 1 来得到下一个数据或者地址的地址，或者根据 in[7..0]来指定数据，进行地址的跳转。

如图 CLR 为清零端，当 CLR 为 0 时，可以将数据 $Q_{7\sim0}$ = 0；LOAD 为置数端，当 LOAD == 0 时，CLK 的上升沿会将 PC 的数据置换为数据数据 in[7..0]；当 CLR=LOAD=ET=EP=1 时，CLK 的上升沿会将 PC 的数据进行自增一。

4. 后续地址选择器 AG

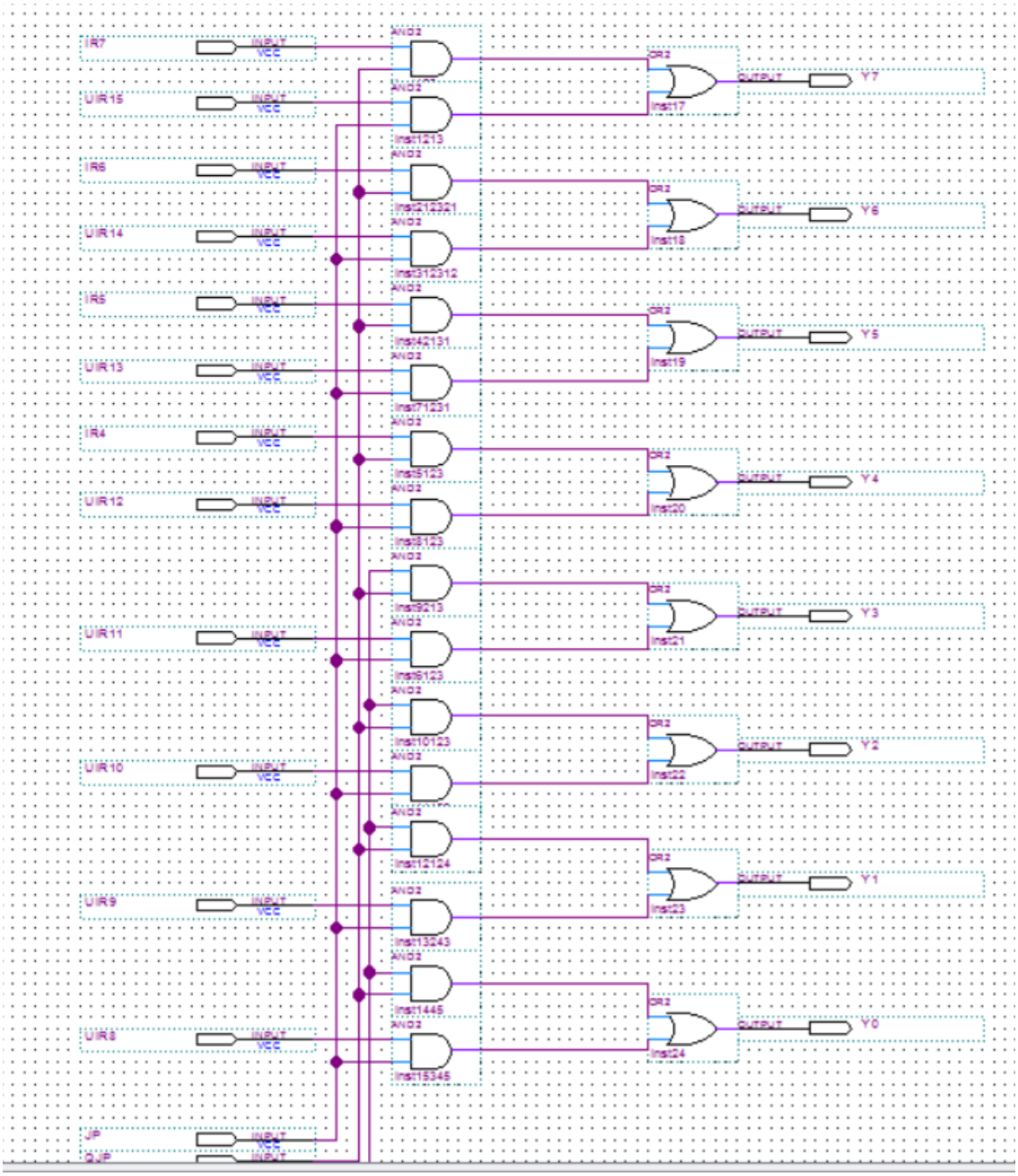


图 2.4 后续地址选择器 AG

后续地址选择器 AG 会根据 JP 和 QJP 两个输入，选择将 $IR_{7\sim4}$ 还是 $UIR_{15\sim8}$ 进行输出。当 JP=0，QJP=1 时，输出 $IR_{7\sim4}$；当 JP=1，QJP=0 时，输出 $UIR_{15\sim8}$。

作用是在进行 uPC 的地址跳转时，根据 JP 或者 QJP 来决定跳转到哪个地址。JP 可以直接进行跳转，一般用来跳转到 00 地址进行读指令操作；而 QJP 可以根据高四位进行跳转，一般用来跳跃到指令集的入口，如 10,20,30 地址等，然后顺序执行以完成一系列微指令。

实现原理即为通过 JP 或者 QJP 与相应的 UIR 或者 IR 的数据进行与运算，从而实现通过 JP 或者 QJP 来选择数据，最后将两者统一到 OUTPUT 进行输出。

5. 数据选择器 chooser

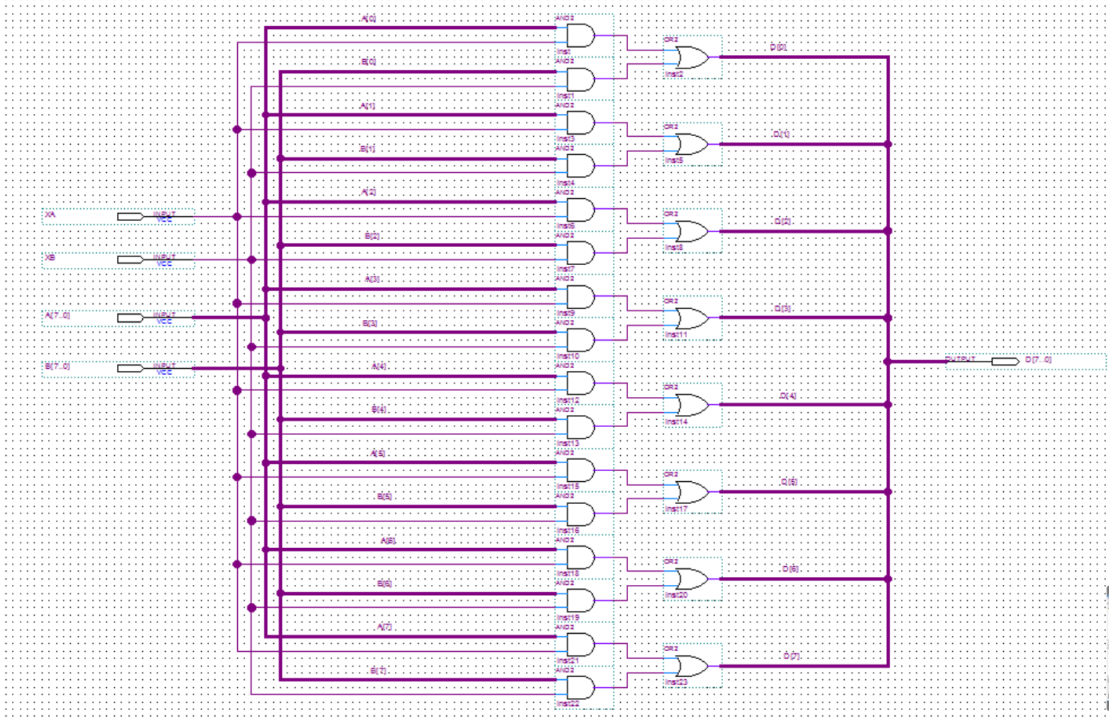


图 2.5 数据选择器 chooser

数据选择器 chooser 根据 XA 或者 XB 的输入，选择是将 A[7..0]还是 B[7..0]进行输出。当 XA == 1，XB == 0 时，OUTPUT 输出为 A[7..0]；当 XB == 1，XA == 1 时，OUTPUT 输出为 B[7..0]。通过对 XA 或者 XB 的指定，选择输送到 ALU 的数据来自何处。在进行扩展时，也可以将选择的数据增多，如增加 XC，XD 等数据，增大灵活性。

实现原理即为 A[7..0]和 B[7..0]数据分别与 XA 和 XB 进行与运算，只有 XA 或者 XB 为 1 时，AND2 才会有输出 A 或者 B，然后经过将 A 或者 B 统一到 OUTPUT 进行输出即可。

6. 运算器 ALU

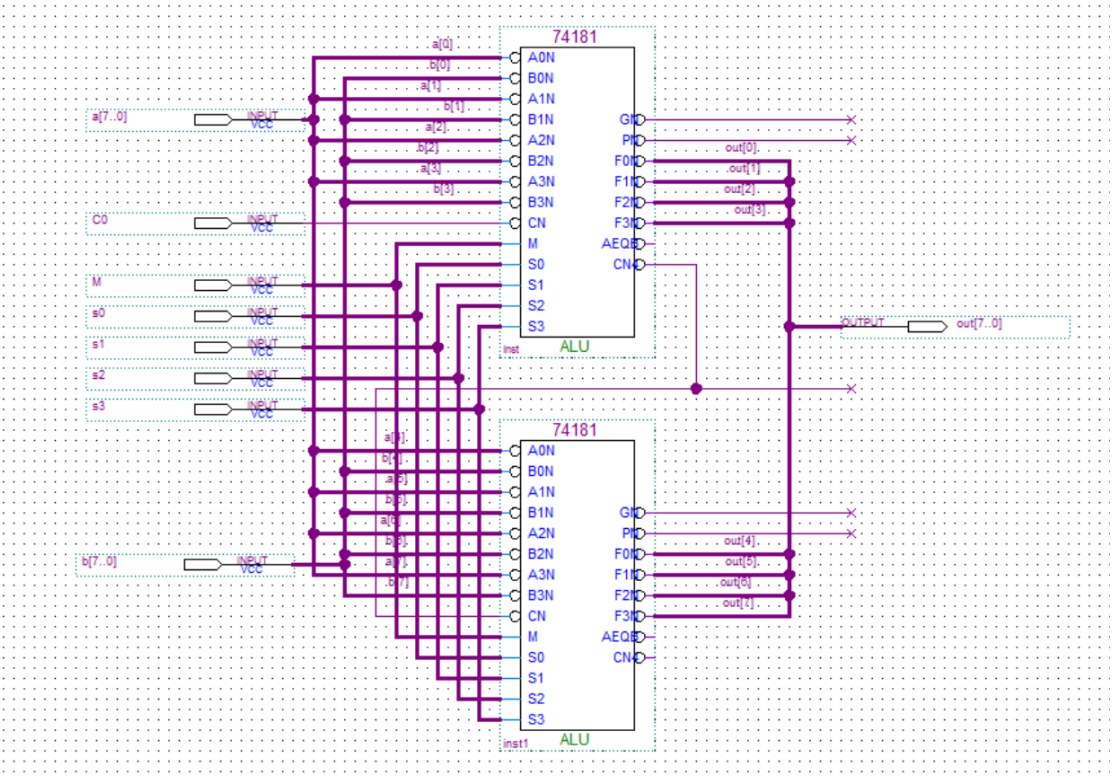


图 2.6 运算器 ALU

运算器 ALU 的实现主要为为两个 74181 元件。本 ALU 的运算位数为 8 位，所以采用了两片 74181，控制输入 M，S_0，S_1，S_2，S_3，和进位 C_0，数据分别为 a[7..0]和 b[7..0]，低位的如果有进位则通过 CN 输送到高位的 C_0 段进行进位。ALU 具有运算功能，可以根据 M 和 $S_{3\text{-}0}$ 选择运算形式，然后将运算结果进行整合输出，实现运算。在 CPU 中即为运算器作用，对数据进行大部分的处理以及直传等，也具有汇总数据的功能。

输入 a[7..0]和 b[7..0]与输出 out[7..0]都进行了封装处理，采用总线输入输出。

功 能 表				
工作方式选择 输入 $S_3S_2S_1S_0$	负逻辑输入或输出		正逻辑输入或输出	
	逻辑运算 (M=1)	算术运算 (M=0) (C_{-1}=0)	逻辑运算 (M=1)	算术运算 (M=0) (C_{-1}=1)
0000	$\overline{A}$	A 减 1	$\overline{A}$	A
0001	$\overline{AB}$	AB 减 1	$\overline{A+B}$	$A+B$
0010	$\overline{A}+B$	$A\overline{B}$ 减 1	$\overline{A}B$	$A+\overline{B}$
0011	逻辑 1	减 1	逻辑 0	减 1
0100	$\overline{A+B}$	A 加 $(A+\overline{B})$	$\overline{AB}$	A 加 $A\overline{B}$
0101	$\overline{B}$	AB 加 $(A+\overline{B})$	$\overline{B}$	$(A+B)$ 加 $A\overline{B}$
0110	$\overline{A\oplus B}$	A 减 B 减 1	$A\oplus B$	A 减 B 减 1
0111	$A+\overline{B}$	$A+\overline{B}$	$A\overline{B}$	$A\overline{B}$ 减 1
1000	$\overline{A}B$	A 加 $(A+B)$	$\overline{A}+B$	A 加 AB
1001	$A\oplus B$	A 加 B	$\overline{A\oplus B}$	A 加 B
1010	B	$A\overline{B}$ 加 $(A+B)$	B	$(A+\overline{B})$ 加 AB
1011	$A+B$	$A+B$	AB	AB 减 1
1100	逻辑 0	A 加 A^*	逻辑 1	A 加 A^*
1101	$A\overline{B}$	AB 加 A	$A+\overline{B}$	$(A+B)$ 加 A
1110	AB	$A\overline{B}$ 加 A	$A+B$	$(A+\overline{B})$ 加 A
1111	A	A	A	A 减 1

(1) 1=高电平；0=低电平；(2) *表示每一位均移到下一个更高位，即 $A^*=2A$

如图为 74181 功能表，根据该功能表可以实现相应的功能。C_{-1} 即为 uIR 中的 C_0 端，M 和 S_{0-3} 对应 uIR 的 M 和 S_{0-3} 即可。

7. 寄存器 Register

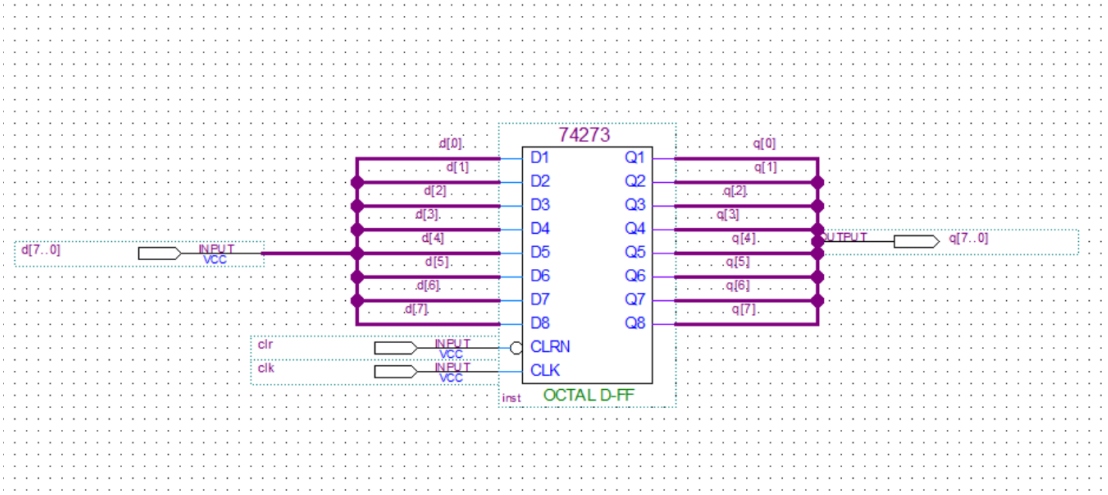


图 2.7 寄存器 Register

寄存器 Register 的功能为寄存数据，将数据暂存，方便进行输出计算，调取速度较快。寄存器在 CPU 模型中为不可或缺的部分，用于总线中数据的传输和计算等，MAR 寄存器还会决定 RAM 的地址选择。

寄存器可以直接采用 74273 寄存器，但是为了方便实用，将 74273 进行封装，输入输出均采用总线处理。

8. 比较器 compare

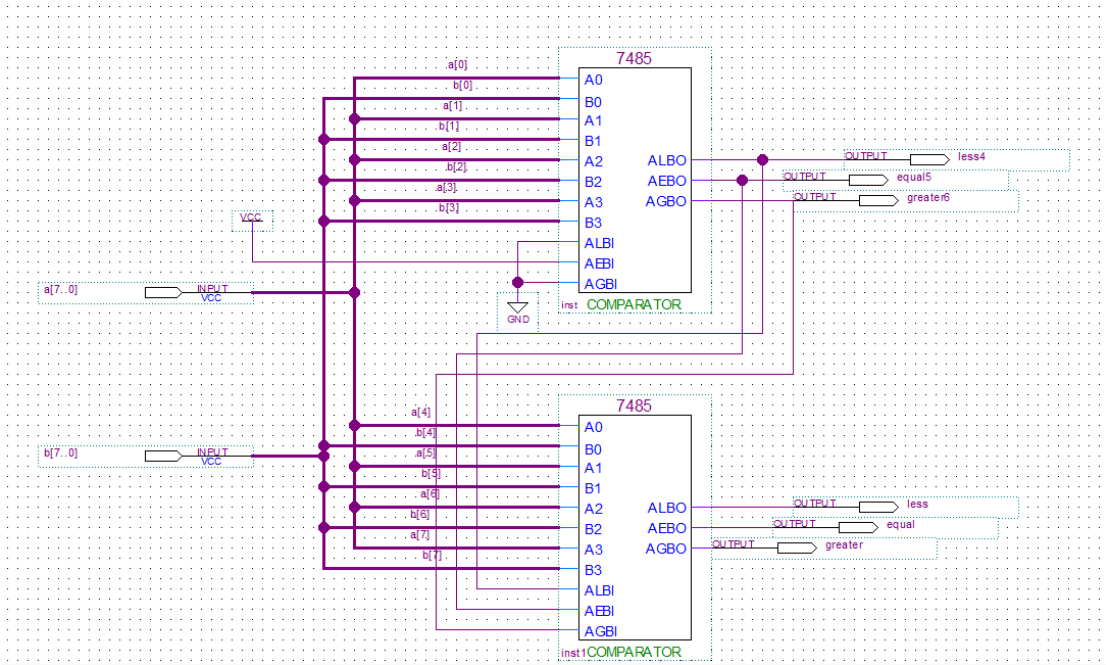


图 2.8 比较器 Compare

比较器 compare，功能是将输入的数据 a[7..0]和 b[7..0]进行比较，得到 a 和 b 的大小关系，可以用于条件判断。

比较器采用 7485 比较元件，对 $A_{3\text{-}0}$ 和 $B_{3\text{-}0}$ 进行按位比较，例如当 $A_3 < B_3$ 时，ALBO 输出为 1，AEBO 和 AGBO 输出为 0；当 $A_{3\text{-}0} = B_{3\text{-}0}$ 时，输出按照输入端的 ALBI、AEBI 和 AGBI 表示。

利用比较器，先进行高位 $a_{7..4}$ 和 $b_{7..4}$ 进行比较，如果不相等则直接输出，相等则根据低位的输出决定。低位比较结果的输出端 O 接入高位的 I 端，用于决定最后的输出。经过这样拓展之后可以比较 8 位的数据大小，可以用于比较判断，比如当 a > 0 时循环运行等，多用于循环操作或者条件转移，当 $R_0 = 5$，判断 $R_0 > 0$ 时循环执行一段 ROM，每次运行都会令 R_0 自减 1，即可实现 5 次的循环执行。或者根据 $R_0 = 0$ 时进行另一段指令的跳转也可。Compare 可以实现大于，小于和等于的判断，在使用时较为方便。

9. 2-4 译码器

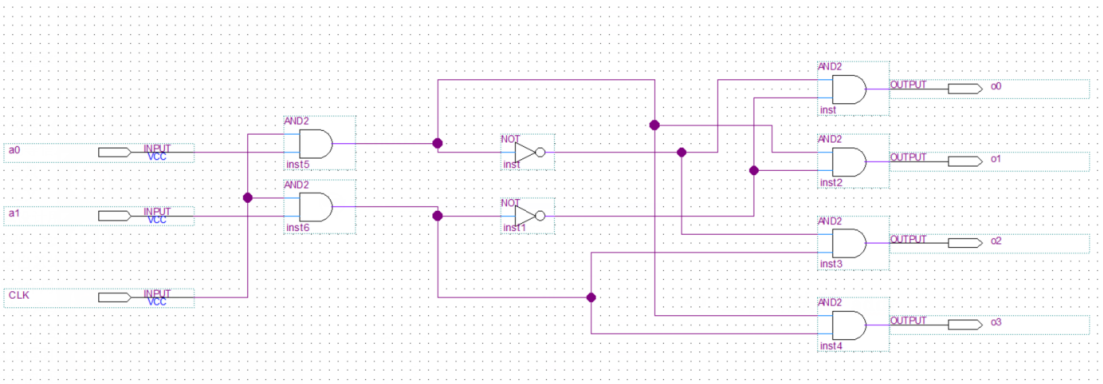


图 2.9 2-4 译码器

2-4 译码器，将 a0 和 a1 输入经过 2-4 译码得到 4 个输出。在实际的电路中，使用了 74138 译码器。当 a_1a_0 输入分别为 00,01,10,11 时，分别有 o0,o1,o2,o3 进行输出，从而实现 2-4 译码。

10.4-16 译码器

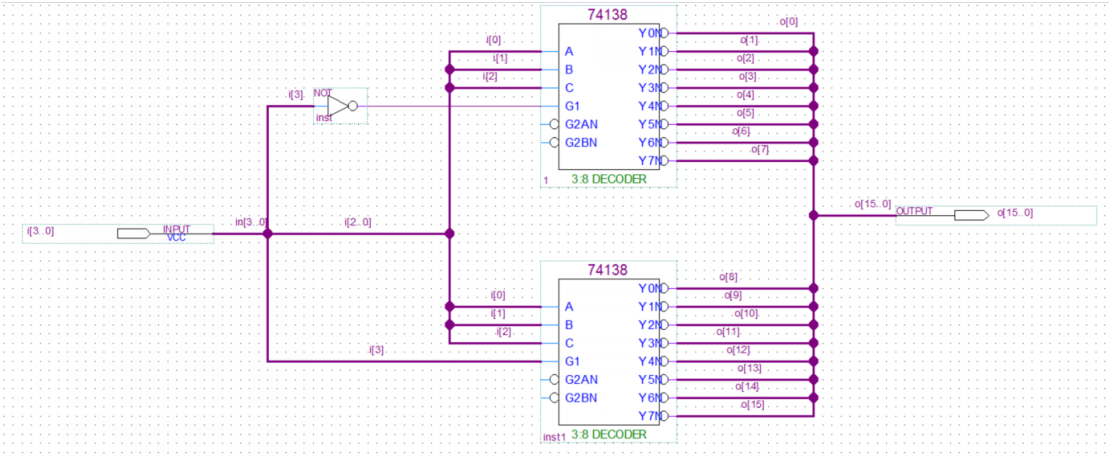


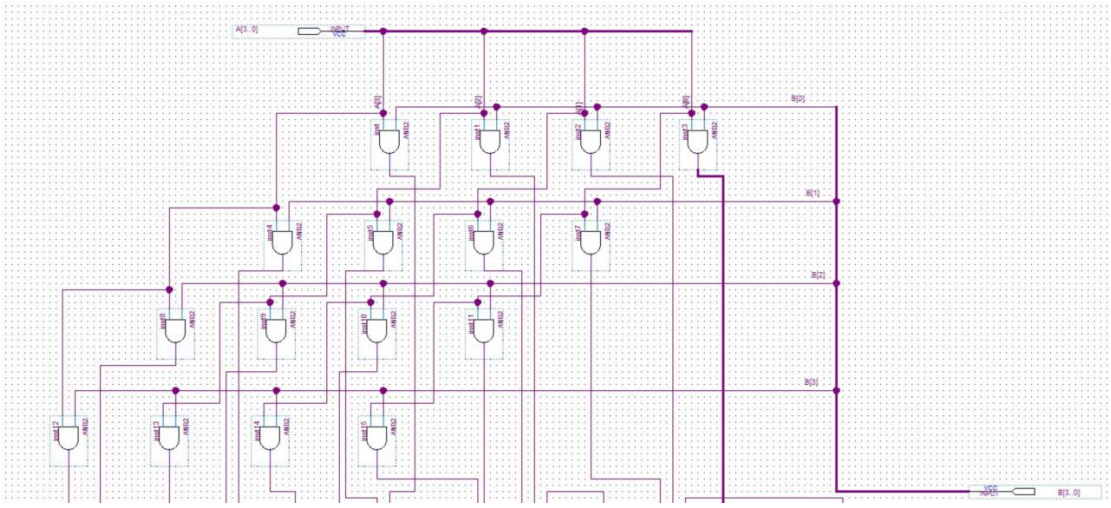
图 2.10 4-16 译码器

4-16 译码器，将 4 位输入进行译码并输出，可以用于指令的拓展。在实验设计的初期，曾设想进行指令长度的变换，从而设计了 4-16 译码器。通过译码后可以将指令集进行扩展，完成单地址、双地址、零地址指令等。

当 $i_{3\text{-}0}$ 为 1111 时，会有 o[15]输出 1 而 o[14..0]输出为 0，将 1111 进行译码，从而将 4 位的输入扩展为 16 位，能够分别进行功能的实现。

4-16 译码器通过两个 74138 译码器实现，输出封装为总线格式。

11.阵列乘法器 Multi



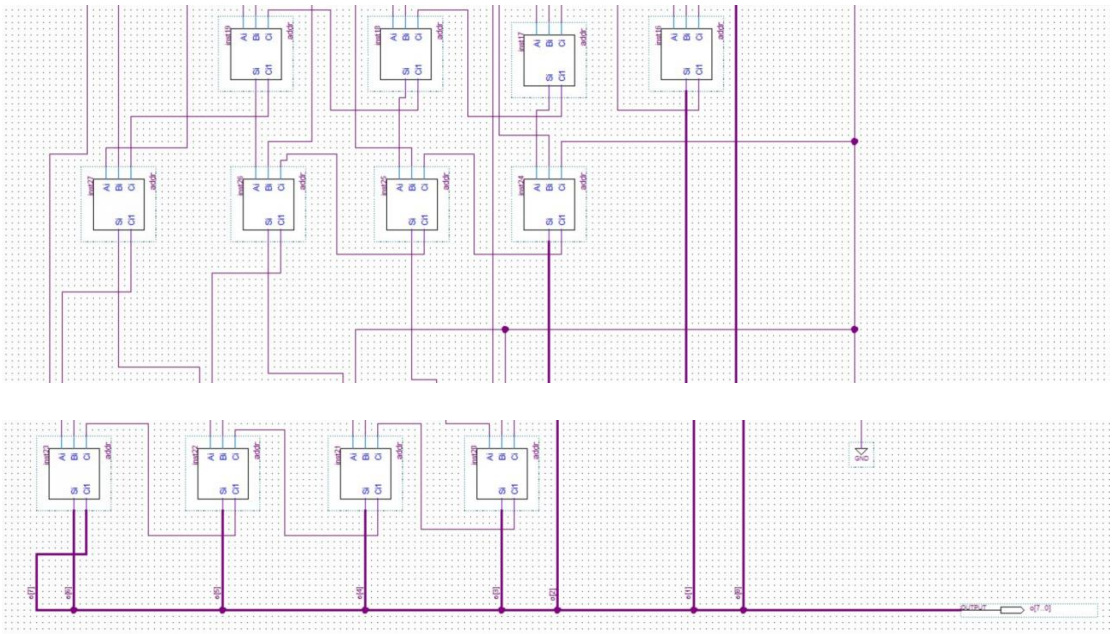


图 2.11 乘法器 Multi

乘法器 Multi，按照阵列乘法实现。考虑到如果 8 位的输入会得到 16 位的输出，数据严重溢出，于是将输入设置为 4 位，输出定为 8 位，限制了数据的输入大小但是使数据更加统一，可以实现一个传入数据（8 位）即可实现数据内的乘法。功能即是完成 4 位的算术乘法运算。

输入输出全部采用总线形式。可以输入一个数据，将高 4 位接入 a[3..0]而低位接入 b[3..0]，从而可以在一个数据的输入即可完成运算，也可以通过输入两个数据分别对高 4 位相乘以及低 4 位相乘。例在 R_0 寄存器存有数据 1010 0101，可以只讲将 R_0 传入 Multi，从而完成 1010 * 0101 = 10 * 5 = 50 = 0011 0010。

12. Addr

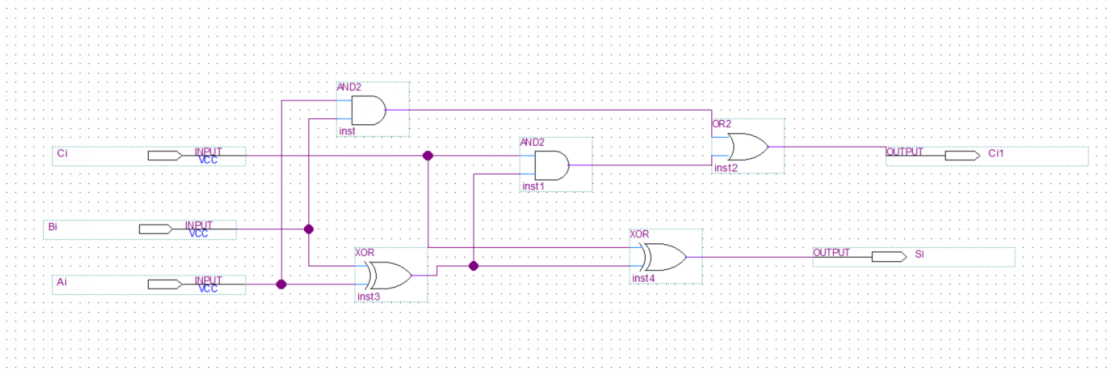


图 2.12 Addr

addr 元件，乘法器内部元件，可以将 A 和 B 相加进行输出，并且支持 C 进位，如果高位有输出则输出到 Ci1，低位 Si 用于判断是否有等位输出。在 Multi 中用于进位和相加，通过移位进行类似相乘的操作。封装后在 Multi 中使用。

13.移位器

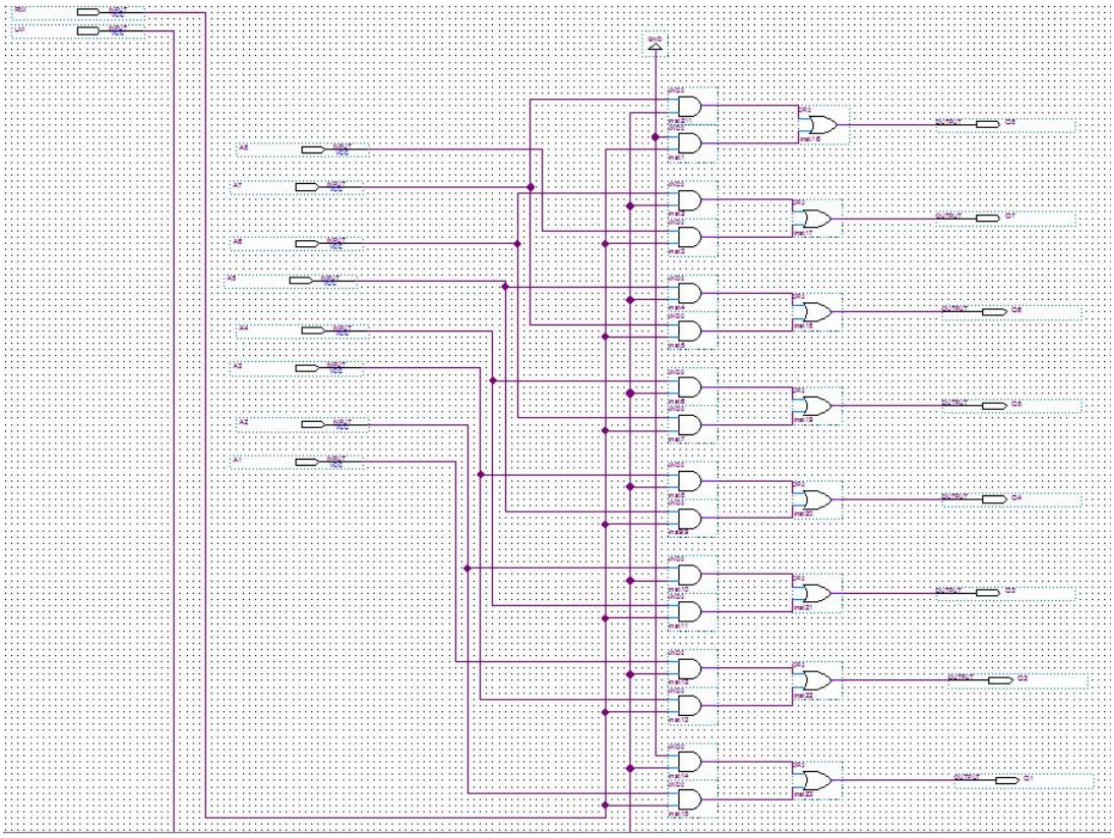


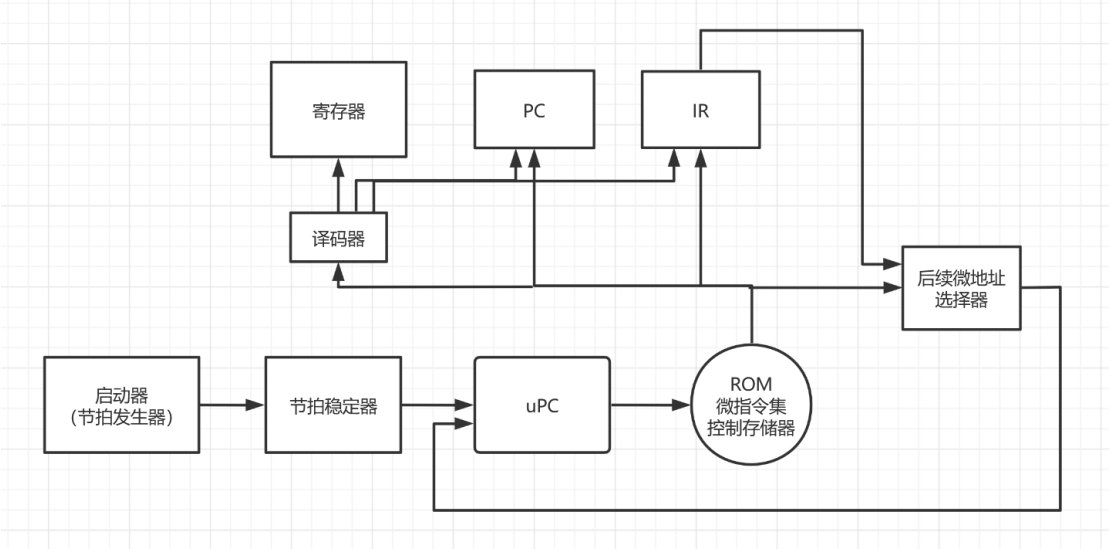
图 2.13 移位器

移位器 shifter，功能是将输入 $A_{8\text{-}0}$ 进行移位输出，根据输入 LM 和 RM 判断左移还是右移。若 LM 为 1，RM 为 0，说明为左移操作，将 $A_{6..0}$ 移位到 $OUT_{7..1}$，OUT_0 为 0 进行输出。同样地，若 RM 为 1，LM 为 0，说明为右移操作，将 $A_{7..1}$ 移位到 $OUT_{6..0}$，OUT_7 为 0 进行输出。

在 CPU 模型中，执行单纯的移位操作，可以更快地实现数据与 2,4,8,16 等数据的相乘（或者相除），不需要经过乘法器。

三. 控制部件 CU

1. CU 框图



（1）后续微地址形成电路

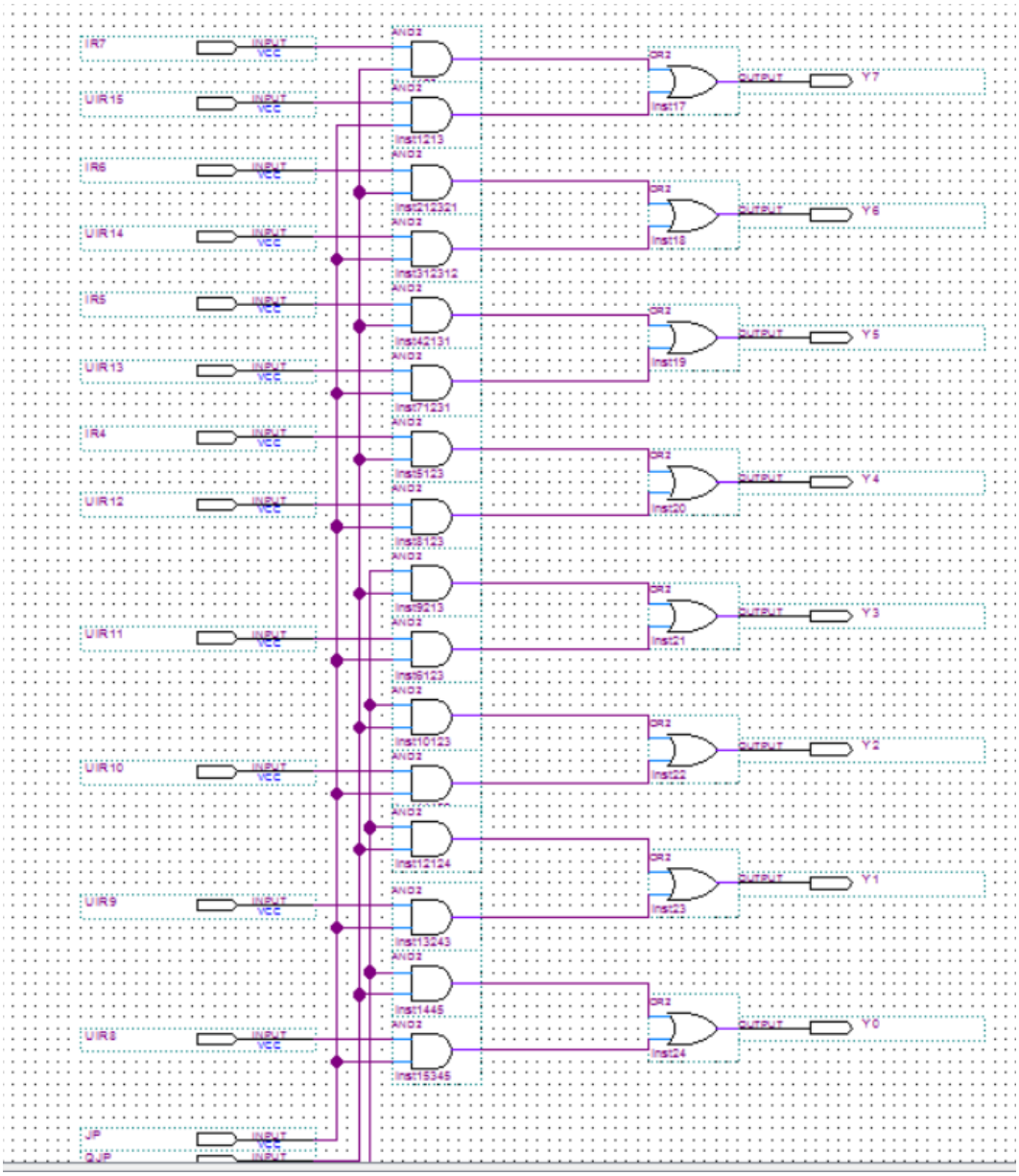


图 3.1-1 后续地址选择器

后续地址选择器的功能是根据 JP 和 QJP 两个输入，选择将 $IR_{7\sim4}$ 还是 $UIR_{15\sim8}$ 进行输出。当 JP=0，QJP=1 时，输出 $IR_{7\sim4}$；当 JP=1，QJP=0 时，输出 $UIR_{15\sim8}$。

其作用是在进行 uPC 的地址跳转时，根据 JP 或者 QJP 来决定跳转到哪个地址。JP 可以直接根据 $UIR_{15\sim8}$ 进行跳转，一般用来跳转到 00 地址进行读指令操作；而 QJP 可以根据 $IR_{7\sim4}$ 高四位进行跳转，一般用来跳跃到指令集的入口，如 10,20,30 地址等，然后顺序执行以完成一系列微指令。

实现原理即为通过 JP 或者 QJP 与相应的 UIR 或者 IR 的数据进行与运算，从而实现通过 JP 或者 QJP 来选择数据，最后将两者统一到 OUTPUT 进行输出。

（2）控制存储器 ROM

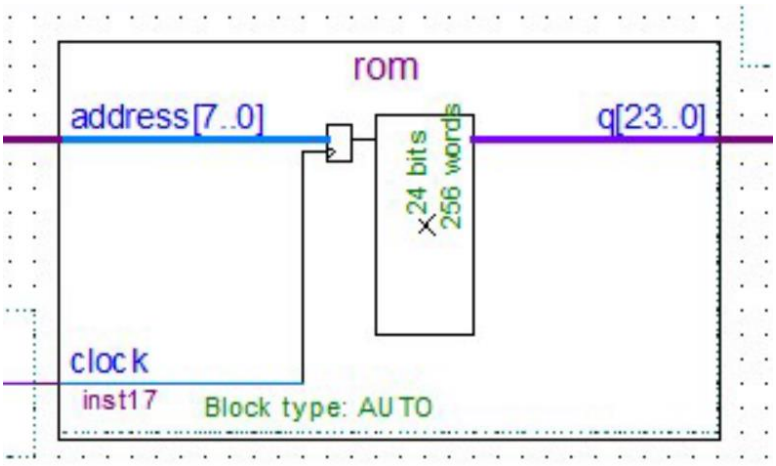


图 3.1-2 控制存储器 ROM

控制存储器 ROM 是只读存储器，储存的是微操作集。功能是根据address[7..0]地址，在每次 clock 输入时，将 ROM 中 address[7..0]地址的数据读出到 q[23..0]进行输出。

在 ROM 不同地址设置不同的微操作，通过改变地址输入，可以选择实现不同的功能。如 00 — 02 地址为读指令的地址，通过将 address 设置为 00，然后依次加 1 可以顺序从控制存储器中读出 00,01,02 地址的操作进行执行，从而实现读指令的功能。

（3）译码器

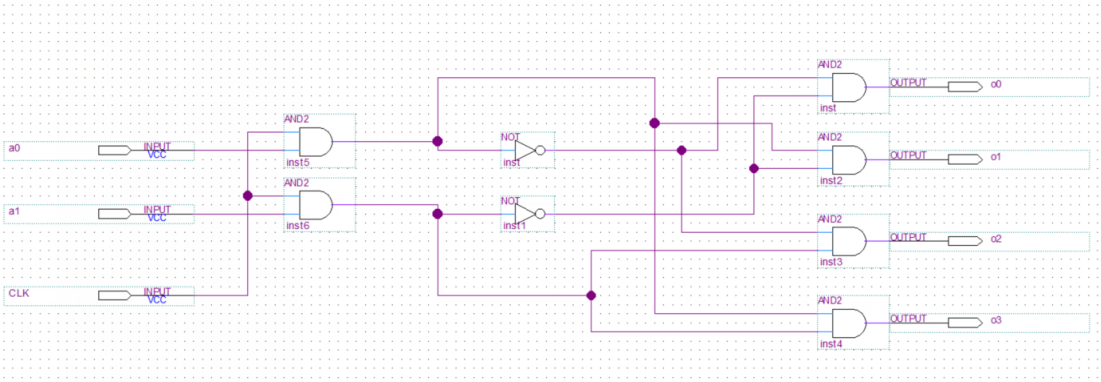


图 3.1.3-1 2-4 译码器

2-4 译码器，将 a0 和 a1 输入经过 2-4 译码得到 4 个输出。在实际的电路中，使用了 74138 译码器。当 a_1a_0 输入分别为 00,01,10,11 时，分别有 o0,o1,o2,o3 进行输出，从而实现 2-4 译码。

在本次实验中，事先实现了 2-4 译码器，但是在使用过程中为了统一性，统一选择了 74138 译码器。

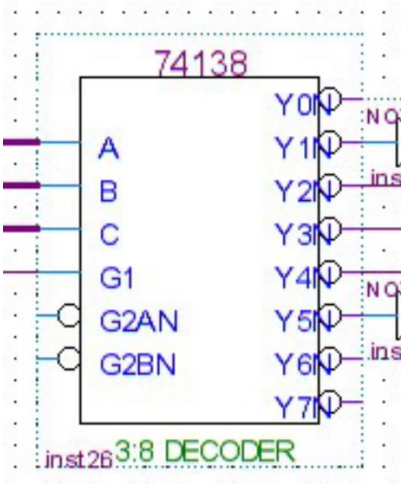


图 3.1.3-2 74138 译码器

74138 译码器，能够实现 3-8 译码。当 G1 为 1 时，$Y_{0\text{-}8}$ 会根据 ABC 的顺序进行译码。

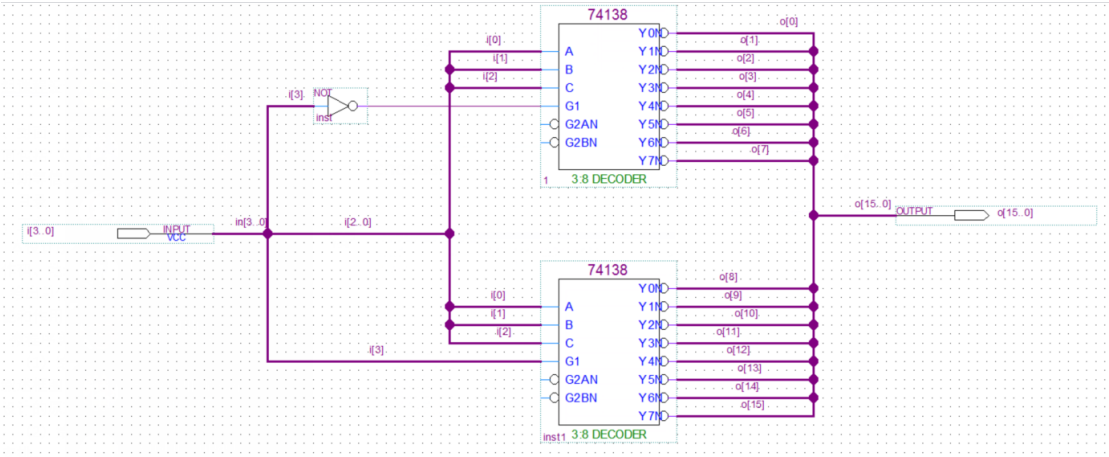


图 3.1.3-3 4-16 译码器

4-16 译码器，将 4 位输入进行译码并输出，可以用于指令的拓展。在实验设计的初期，曾设想进行指令长度的变换，从而设计了 4-16 译码器。但是在后续实验中，因为时间原因没有实现更多的功能，因而没有用到 4-16 译码器。

（4）uPC

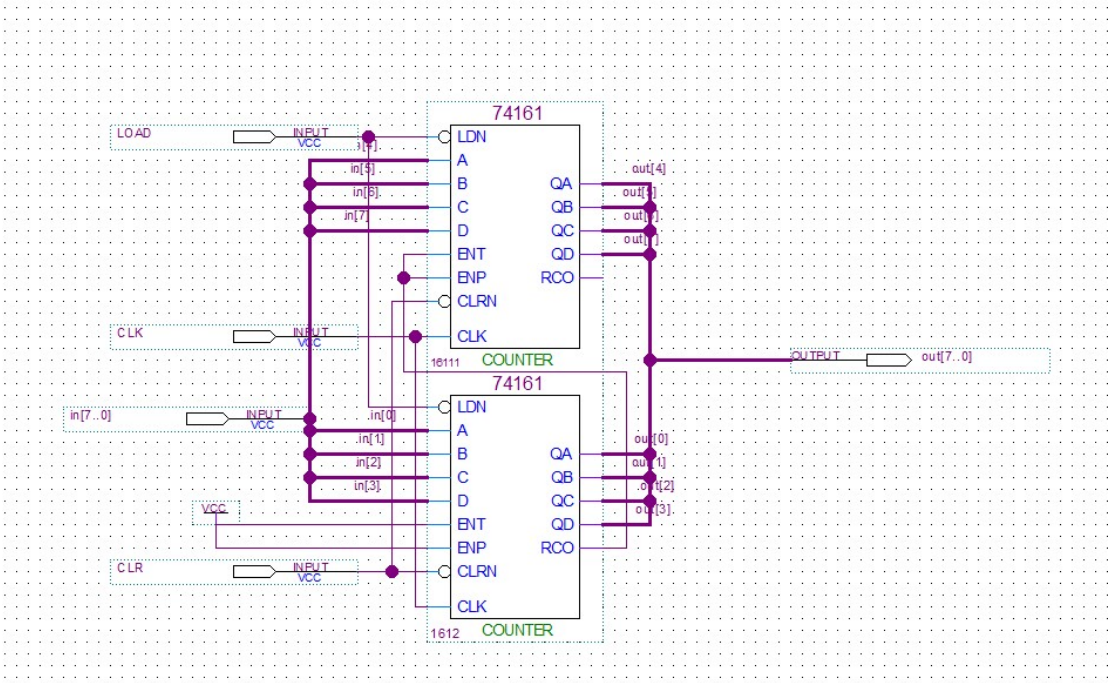


图 3.1-4 指令计数器 uPC

指令计数器 PC 可以在有 CLK 输入时，对 8 位数据进行加 1 处理，或者在 LOAD 有输入时，将数据置换为 in[7..0]的数据并进行输出。作用是可以每次对自身数据加 1 来得到下一个数据或者地址的地址，或者根据 in[7..0]来指定数据，进行地址的跳转。

uPC 会自动将地址加 1 送入 ROM，从而顺序从 ROM 中读取微操作，以完成一段完整的指令。

2. 指令执行流程

首先进行取指周期，即从 00 地址开始顺序执行。在 02 地址处执行 QJP，跳转到 IR 所指示的地址，执行执行周期。

程序	微操作	微地址	
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#RAM 送入 R0	PC->MAR	10	执行周期，PC 送入 MAR；
	PC+1->PC	11	PC 加一；
	RAM->R0	12	RAM 根据 MAR 选择数据送入 R0
	PC->MAR	13	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	14	跳转到取指周期

读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#RAM 送入 R1	PC->MAR	20	执行周期，PC 送入 MAR；
	PC+1->PC	21	PC 加一；
	RAM->R1	22	RAM 根据 MAR 选择数据送入 R1
	PC->MAR	23	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	24	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0+R1 送到 RAM	PC->MAR	30	执行周期，PC 送入 MAR；
	PC+1->PC	31	PC 加一；
	RAM->MAR	32	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（80）；
	R0+R1->(70)	33	R0+R1 得到的数据送入 RAM；
	PC->MAR	34	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	35	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0-R1 送到 RAM	PC->MAR	40	执行周期，PC 送入 MAR；
	PC+1->PC	41	PC 加一；
	RAM->MAR	42	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（81）；
	R0-R1->(71)	43	R0-R1 得到的数据送入 RAM；
	PC->MAR	44	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	45	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0*R1 送到 RAM	PC->MAR	50	执行周期，PC 送入 MAR；
	PC+1->PC	51	PC 加一；
	RAM->MAR	52	RAM 根据 MAR 选择数据送入

	R0*R1->(72)	53	MAR，此时的 MAR 即为数据存放的地址（82）； R0*R1 得到的数据送入 RAM；
	PC->MAR	54	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	55	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0&R1 送到 RAM	PC->MAR	60	执行周期，PC 送入 MAR；
	PC+1->PC	61	PC 加一；
	RAM->MAR	62	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（83）；
	R0&R1->(73)	63	R0&R1（与运算）得到的数据送入 RAM；
	PC->MAR	64	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	65	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0|R1 送到 RAM	PC->MAR	70	执行周期，PC 送入 MAR；
	PC+1->PC	71	PC 加一；
	RAM->MAR	72	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（84）；
	R0|R1->(74)	73	R0|R1（或运算）得到的数据送入 RAM；
	PC->MAR	74	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	75	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#（R0+R1）非送到 RAM	PC->MAR	80	执行周期，PC 送入 MAR； PC 加一；
	PC+1->PC	81	
	RAM->MAR	82	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放

	/(R0+R1)->(75)	83	的地址（85）； /（R0+R1）（与非运算）得到的数据送入 RAM；
	PC->MAR	84	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	85	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0^R1 送到 RAM	PC->MAR	90	执行周期，PC 送入 MAR；
	PC+1->PC	91	PC 加一；
	RAM->MAR	92	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（86）；
	R0^R1->(76)	93	R0^R1（异或运算）得到的数据送入 RAM；
	PC->MAR	94	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	95	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0+1 送到 RAM	PC->MAR	a0	执行周期，PC 送入 MAR；
	PC+1->PC	a1	PC 加一；
	RAM->MAR	a2	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（87）；
	R0+1->(77)	a3	R0+1 得到的数据送入 RAM；
	PC->MAR	a4	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	a5	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0-1 送到 RAM	PC->MAR	b0	执行周期，PC 送入 MAR；
	PC+1->PC	b1	PC 加一；
	RAM->MAR	b2	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（88）；
	R0-1->(78)	b3	R0-1 得到的数据送入 RAM；

	PC->MAR	b4	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	b5	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0 取反送到 RAM	PC->MAR	c0	执行周期，PC 送入 MAR；
	PC+1->PC	c1	PC 加一；
	RAM->MAR	c2	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（89）；
	/R0->(79)	c3	R0 取反得到的数据送入 RAM；
	PC->MAR	c4	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	c5	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0 左移送到 RAM	PC->MAR	d0	执行周期，PC 送入 MAR；
	PC+1->PC	d1	PC 加一；
	RAM->MAR	d2	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（8A）；
	（R0<-1)->(7A)	d3	R0 左移一位得到的数据送入 RAM；
	PC->MAR	d4	PC 送入 MAR，用于确定下一个微操作的地址；
	JP	d5	跳转到取指周期
读取指令	RAM->IR	00	取指周期
	PC+1->PC	01	PC 加 1
	QJP	02	QJP，跳转到 IR 指定的地址
#R0 间接送到 RAM	PC->MAR	e0	执行周期，PC 送入 MAR；
	PC+1->PC	e1	PC 加一；
	RAM->MAR	e2	RAM 根据 MAR 选择数据送入 MAR，此时的 MAR 即为数据存放的地址（8B）；
	RAM->MAR	e3	RAM 再一次根据 MAR（8B）选择数据送入 MAR，此时的 MAR 即为数据存放的地址（8C）；
	R0->(7C)	e4	R0 送入 RAM；
	PC->MAR	e5	PC 送入 MAR，用于确定下一个微操作的地址；

HALT		e6	停机指令

3. 微指令格式

字段序号	功能	下地址字段	解释说明
23	移位判断		1 代表进行移位操作，0 代表不进行移位操作，属于操作码
22	乘法判断		1 代表进行乘法操作，0 代表不进行乘法操作，属于操作码
21	M		ALU 的操作码，根据 74181 的功能表进行实现。 M 决定为逻辑运算还是算术运算； $S_{3\text{-}0}$ 决定操作； C_0 为进位。
20	S_3		
19	S_2		
18	S_1		
17	S_0		
16	C_0		
15 14	数据 A 来源选择	当执行 JP 指令时，该部分为 JP 的目的地址	运算器数据 A 的来源，对 $uIR_{15,14}$ 进行 2-4 译码。00 和 11 无选择，10 输入选择为为 R_0，01 输入选择为 RAM
13 12	数据 B 来源选择		运算器数据 B 的来源，对 $uIR_{13,12}$ 进行 2-4 译码。00 和 11 无选择，10 输入选择为为 PC，01 输入选择为 R_1
11 10 9	输出寄存器选择		运算器结果输出位置选择。对 $uIR_{11\text{-}9}$ 进行 3-8 译码，经过译码后 1 代表输出到 R0，2 代表输出到 R1，3 代表输出到 PC，4 代表输出到 IR，5 代表输出到 MAR。
8	NULL		
7	NULL		
6	移位方向		如果 uIR 为 1，说明进行移位操作，则此位置决定移位方向。1 代表左移 1 位，0 代表右移 1 位。
5	NULL		
4	RAM 读写标识符		对 RAM 进行读或者写的标识符。1 代表写操作，0 代表读操作。
3	NULL		
2 1 0	跳转方式		经过 3-8 译码器后可以为 8 种地址生成方式。其中 1 为 PC+1 顺序执行；2 为 JP 为无跳转转移，根据 $uIR_{15\text{~}8}$ 跳转；3 为 QJP，根据操作码 IR 的高 4 位跳转，低 4 位默认为 0；4 为 YJP，给定高 4 位，低 4 位按照源寻址方式进行转移；5 为 MJP，

			给定高 4 位，低 4 位按照目寻址方式转移。

程序	微操作	微地址	16 进制码
读取指令	RAM->IR	00	3E8801
	PC+1->PC	01	121601
	QJP	02	000003
#RAM 送入 R0	PC->MAR	10	341A01
	PC+1->PC	11	121601
	RAM->R0	12	3E8201
	PC->MAR	13	341A01
	JP	14	000002
#RAM 送入 R1	PC->MAR	20	341A01
	PC+1->PC	21	121601
	RAM->R1	22	3E8401
	PC->MAR	23	341A01
	JP	24	000002
#R0+R1 送到 RAM	PC->MAR	30	341A01
	PC+1->PC	31	121601
	RAM->MAR	32	348A01
	R0+R1->(70)	33	136011
	PC->MAR	34	341A01
	JP	35	000002
#R0-R1 送到 RAM	PC->MAR	40	341A01
	PC+1->PC	41	121601
	RAM->MAR	42	348A01
	R0-R1->(71)	43	0C6011
	PC->MAR	44	341A01
	JP	45	000002
#R0*R1 送到 RAM	PC->MAR	50	341A01
	PC+1->PC	51	121601
	RAM->MAR	52	348A01
	R0*R1->(72)	53	406011
	PC->MAR	54	341A01
	JP	55	000002
#R0&R1 送到 RAM	PC->MAR	60	341A01
	PC+1->PC	61	121601
	RAM->MAR	62	348A01
	R0&R1->(73)	63	366011
	PC->MAR	64	341A01

	JP	65	000002
#R0|R1 送到 RAM	PC->MAR	70	341A01
	PC+1->PC	71	121601
	RAM->MAR	72	348A01
	R0|R1->(74)	73	3C6011
	PC->MAR	74	341A01
	JP	75	000002
#（R0+R1）非送到 RAM	PC->MAR	80	341A01
	PC+1->PC	81	121601
	RAM->MAR	82	348A01
	/(R0+R1)->(75)	83	226011
	PC->MAR	84	341A01
	JP	85	000002
#R0^R1 送到 RAM	PC->MAR	90	341A01
	PC+1->PC	91	121601
	RAM->MAR	92	348A01
	R0^R1->(76)	93	2C6011
	PC->MAR	94	341A01
	JP	95	000002
#R0+1 送到 RAM	PC->MAR	a0	341A01
	PC+1->PC	a1	121601
	RAM->MAR	a2	348A01
	R0+1->(77)	a3	004011
	PC->MAR	a4	341A01
	JP	a5	000002
#R0-1 送到 RAM	PC->MAR	b0	341A01
	PC+1->PC	b1	121601
	RAM->MAR	b2	348A01
	R0-1->(78)	b3	1F4011
	PC->MAR	b4	341A01
	JP	b5	000002
#R0 取反送到 RAM	PC->MAR	c0	341A01
	PC+1->PC	c1	121601
	RAM->MAR	c2	348A01
	/R0->(79)	c3	204011
	PC->MAR	c4	341A01
	JP	c5	000002
#R0 左移送到 RAM	PC->MAR	d0	341A01
	PC+1->PC	d1	121601
	RAM->MAR	d2	348A01

	（R0<-1)->(7A)	d3	804051
	PC->MAR	d4	341A01
	JP	d5	000002
#R0 间接送到 RAM	PC->MAR	e0	341A01
	PC+1->PC	e1	121601
	RAM->MAR	e2	348A01
	RAM->MAR	e3	348A01
	R0->(7C)	e4	3E4011
	PC->MAR	e5	341A01
#HALT		e6	000008

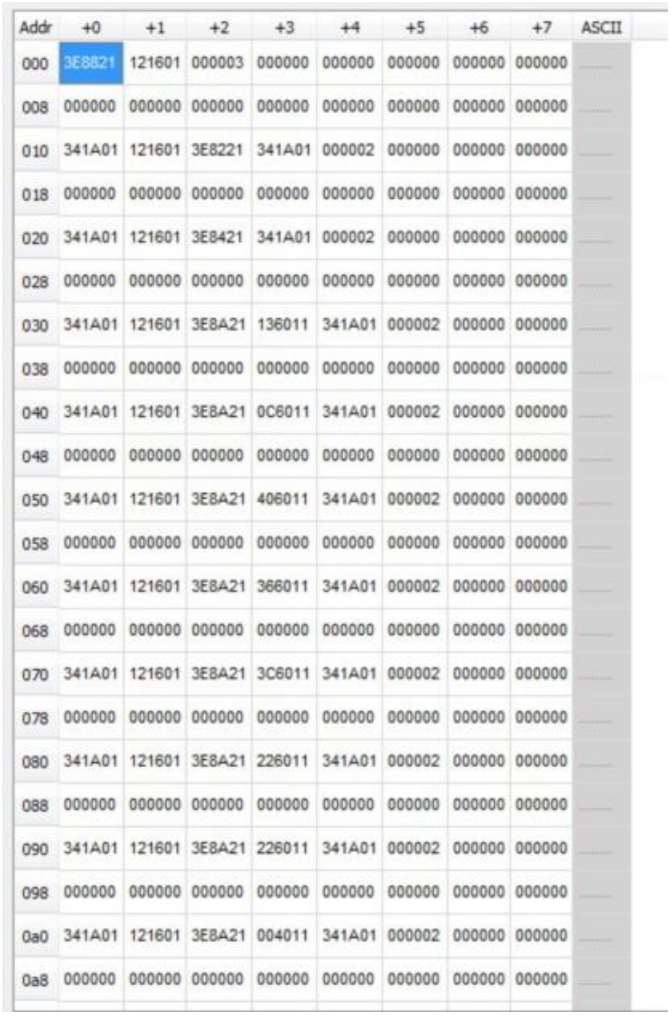
4. RAM 应用程序

汇编语言

#RAM 送入 R0	MOV1 01#，R_0
#RAM 送入 R1	MOV2 03#，R_1
#R0+R1 送到 RAM	MOV1 01#，R_0
	MOV2 03#，R_1
	ADD R_0,R_1,(70#)
#R0-R1 送到 RAM	MOV1 01#，R_0
	MOV2 03#，R_1
	SUB R_0,R_1,(71#)
#R0*R1 送到 RAM	MOV1 01#，R_0
	MOV2 03#，R_1
	MUL R_0,R_1,(72#)
#R0&R1 送到 RAM	MOV1 01#，R_0
	MOV2 03#，R_1
	AND R_0,R_1,(73#)
#R0|R1 送到 RAM	MOV1 01#，R_0
	MOV2 03#，R_1
	OR R_0,R_1,(74#)
#（R0+R1）非送到 RAM	MOV1 01#，R_0
	MOV2 03#，R_1

	NOR R_0,R_1,(75#)
#R0^R1 送到 RAM	MOV1 01#，R_0
	MOV2 03#，R_1
	XOR R_0,R_1,(76#)
#R0+1 送到 RAM	MOV1 01#，R_0
	ADD R_0,1,(77#)
#R0-1 送到 RAM	MOV1 01#，R_0
	SUB R_0,1,(78#)
#R0 取反送到 RAM	MOV1 01#，R_0
	NOT R_0,(79#)
#R0 左移送到 RAM	MOV1 01#，R_0
	LM R_0,1,(7A#)
#R0 间接送到 RAM	MOV4 (7B#),MAR
	MOV3 R0, (MAR#)
#HALT	HALT

机器语言



Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
000	3E8821	121601	000003	000000	000000	000000	000000	000000	.....
008	000000	000000	000000	000000	000000	000000	000000	000000	.....
010	341A01	121601	3E8221	341A01	000002	000000	000000	000000	.....
018	000000	000000	000000	000000	000000	000000	000000	000000	.....
020	341A01	121601	3E8421	341A01	000002	000000	000000	000000	.....
028	000000	000000	000000	000000	000000	000000	000000	000000	.....
030	341A01	121601	3E8A21	136011	341A01	000002	000000	000000	.....
038	000000	000000	000000	000000	000000	000000	000000	000000	.....
040	341A01	121601	3E8A21	0C6011	341A01	000002	000000	000000	.....
048	000000	000000	000000	000000	000000	000000	000000	000000	.....
050	341A01	121601	3E8A21	406011	341A01	000002	000000	000000	.....
058	000000	000000	000000	000000	000000	000000	000000	000000	.....
060	341A01	121601	3E8A21	366011	341A01	000002	000000	000000	.....
068	000000	000000	000000	000000	000000	000000	000000	000000	.....
070	341A01	121601	3E8A21	3C6011	341A01	000002	000000	000000	.....
078	000000	000000	000000	000000	000000	000000	000000	000000	.....
080	341A01	121601	3E8A21	226011	341A01	000002	000000	000000	.....
088	000000	000000	000000	000000	000000	000000	000000	000000	.....
090	341A01	121601	3E8A21	226011	341A01	000002	000000	000000	.....
098	000000	000000	000000	000000	000000	000000	000000	000000	.....
0a0	341A01	121601	3E8A21	004011	341A01	000002	000000	000000	.....
0a8	000000	000000	000000	000000	000000	000000	000000	000000	.....
0b0	341A01	121601	3E8A21	1F4011	341A01	000002	000000	000000	.....
0b8	000000	000000	000000	000000	000000	000000	000000	000000	.....
0c0	341A01	121601	3E8A21	204011	341A01	000002	000000	000000	.....
0c8	000000	000000	000000	000000	000000	000000	000000	000000	.....
0d0	341A01	121601	3E8A21	804051	341A01	000002	000000	000000	.....
0d8	000000	000000	000000	000000	000000	000000	000000	000000	.....
0e0	341A01	121601	3E8A21	3E8A21	3E4011	341A01	000008	000000	.....
0e8	000000	000000	000000	000000	000000	000000	000000	000000	.....
0f0	000000	000000	000000	000000	000000	000000	000000	000000	.....
0f8	000000	000000	000000	000000	000000	000000	000000	000000	.....

	A	B	C	D	E	F	G	H
1	10	08	20	04	30	70	40	71
2	50	72	60	73	70	74	80	75
3	90	76	a0	77	b0	78	c0	79
4	d0	7A	e0	20	7B			
5	FF							

四．课程设计总结

经过本次课程设计实验，我收获非常良多。

首先是对计算机的组成有了更加深刻的认识，通过一步一步设计零件，对系统的认识有了更加深刻的理解。从一开始设计操作功能（见下图）。

指令	几地址	操作码
取数	1地址，操作码6位	1111 00
存数	1地址，操作码6位	1111 01
加法	1地址，操作码6位	1110 00
	2地址	0000
减法	1地址	1110 01
	2地址	0001
乘法	1地址	1110 10
	2地址	0010
PC转移	1地址	1100 00
左移	1地址	1100 01
	0地址（ACC	1011 11 00
右移	1地址	1100 10
	0地址（ACC	1011 11 01
取反—	1地址	1101 00
	0地址（ACC	1011 11 10

与运算&	1地址	1101 01
	2地址	0011
或运算|	1地址	1101 10
	2地址	0100
异或^	1地址	1101 11
	2地址	0101
条件slt小于	0地址（$R_0<R_1$？1:0）	1011 11 11

00——R0

01——R1

10——立即数

11——直接寻址

根据操作功能设计完元件（如 ALU，乘法器，比较器，移位器等），在进行整体框架的链接上却产生了难以逾越的困难。因为对整体框架的条理没有理顺，在操作码的扩展上，难以对不同操作进行统一（如对 1 地址的几种运算没有统一的框架，导致在连线上如果仅采用一对一的连线，会非常之复杂）。此外，对于操作码的扩展也设计了 2-4 译码器以及 4-16 译码器，但是就会导致某些位置既可能为地址码又可能为操作码，对判断上产生了困难。后来经过仔细研读，决定扩展 8 位的指令字长为统一的 24 位，对于不同位置的扩展也有了更大的可能性。

在改为 24 位字长之后，因为在零件设计时都封装成 bus 总线的形式，在整体部件上的连线没有过多的困难，但是因为时间关系，有些曾设想实现的功能最终也没有落实。连线完成后，在指令的设计上又产生了疑问，如何才能实现代码的循环使用，以及**微操作，微指令，微命令，微程序**等概念尽管复习了一遍又一遍，再一次遇见时也会产生模糊，也因此对于整体的运行没有清晰的观念。后来仔细捋清运行过程后，设计了测试指令，但是在测试指令的实现上，依然出现了问题。

此时对时序问题的理解还不足，没有意识到时序错误对程序产生的致命影响。私以为问题是出现在部件的连线或者实现上。花费大量时间进行检查后，依然没有解决。后来在不同的部件 clock 上分步运行，成功模拟实现。不过相同的程序以及相同的操作步骤，有时也会产生错误的运行结果。了解到时序可能会产生问题后，设计了节拍稳定器来稳定时序，保证运行。

每次遇到困难时，都不禁赞叹于先辈设计计算时的辛苦历程，而每次解决困难之后，又惊诧于系统高度的统一性和系统性，每一步运算都是那样的精妙，每一个操作都是如此巧妙，更加提高了我对先辈工作的敬仰之情。

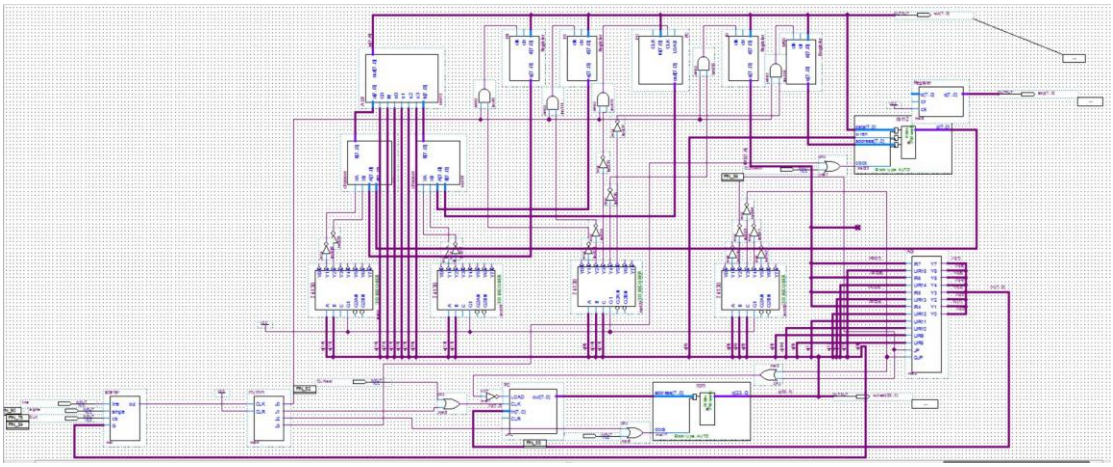
完成实验之后，自身能力也得到了极大的提高。首先是对知识的掌握程度有了极大的提高，对曾经模糊的知识有了更深的认识（不过对于**微操作，微指令，微命令，微程序**等的分别还是难以完全准确地区分）；再者是对 debug 的能力增强了，比如通过仔细阅读运行过程，寻找每一步可能出现错误的地方，从而针对性下药，（不过时至今日，我也不知道为什么在 debug 时对某些线路仅仅加了一个输出就导致运行结果的完全错误，而仅仅删除掉输出与其中间的连线便可以成功运行）；对于自身细心和认真的态度也有所帮助，只有认真做好每一

个环节，没有一点差错，才能保证整个程序的顺利运行，这就极大地锻炼了细心认真的态度。

最后，需要对老师、助教以及同学的帮助给予极大的感谢。没有他们的帮助，在一些问题上可能要花费更多的时间甚至到实验结束时间也难以解决。

五． 附录

附录 1：数据通路图



（或见 **附件 1：数据通路图.png**）

附录 2：微程序流程图

（详见 **附录 2：指令执行流程图.pdf** 或 **附录 2：指令执行流程图.xlsx**）

附录 3：微程序

（详见 **附录 3：rom 微操作.xlsx**）