第3章 控制器

人们在使用计算机时，需要通过执行程序来实现相应的功能。这种程序，可以是别人写好后，作为产品出售，用户买回来使用，也可以由计算机用户自己编写。无论是哪种程序，都是由计算机语言编写而成的。汇编语言就是计算机语言的一种。另外一种更常见的计算机语言叫做高级语言，C、C++、C＃、Java 等都属于这一种。无论是汇编语言，还是高级语言，计算机都无法直接识别。计算机只能直接识别另一种计算机语言——机器语言。需要把汇编语言或高级语言编写的程序翻译成机器语言形式，计算机才能识别。

3.1 机器语言

下面看一下机器语言的形式。鼠标单击“开始”→“运行”按钮，用键盘输入“DEBUG”、“回车”，启动 DEBUG。输入“A 100”、“回车”，进入输入汇编指令状态。输入汇编指令“INC AX”、“回车”。再输入“回车”，退出输入汇编指令状态，回到输入 DEBUG 命令状态。输入“U 100，100”、“回车”，显示刚才输入的那句汇编指令。屏幕上出现如图 3.1 所示的结果。

图 3.1 DEBUG 的 U 命令

左侧 1383：0100 是地址，是这句指令在内存中的保存位置。右侧可以看到刚输入的那句汇编指令 INC AX。中间的 40 就是机器语言形式的指令——机器指令。当然，这个 40 是十六进制的。在计算机内部，机器指令是按照二进制形式保存的。这句机器指令在计算机内部是 01000000。机器指令 40 与汇编指令 INC AX 的含义是一样的，都是把寄存器 AX 加 1。

先看一下汇编指令 INC AX 在计算机内部的形式。在 DEBUG 中输入“E 200”、“回车”、“49”、“空格”、“4E”、“空格”、“43”、“空格”、“20”、“空格”、“41”、“空格”、“58”、“回车”、“D 200，205”、“回车”，屏幕上出现如图 3.2 所示的结果。

在屏幕的右侧会看到汇编指令 INC AX，左侧会看到刚才输入的 6 个十六进制数 49、4E、43、20、41、58。这 6 个数就是 INC AX 在计算机内部的存在形式，依次表示 I、N、C、空

```
-E 200
1383:0200  00.49   00.4E   00.43   00.20   00.41   00.58
-D 200,205
1383:0200  49 4E 43 20 41 58                              INC AX
-
```

图 3.2 汇编指令的机内形式

格、A、X。

其实，这是一个计算机如何表示符号的问题。计算机是作为计算工具被发明出来的，它的第一个用途就是数值计算。后来，人们试图用它处理符号。首先要解决的是如何在计算机内部表示符号。计算机内部只有二进制数，只能用不同的二进制数代表不同的符号。本来，哪个二进制数代表哪个符号都是可以的。但是，定义不一致的计算机是无法交流的。例如，甲计算机中规定 1 代表字母 A，2 代表字母 B，按照字母表的顺序以此类推。计算机内部用 144 表示 ADD。乙计算机中规定 0 代表字母 A，1 代表字母 B，以此类推。甲计算机中的 144 传送到乙计算机中，就被误解为 BEE 了。所以，为了方便交流，人们都使用相同的规定。习惯上，使用比较多的是 ASCII 码（American Standard Code for Information Interchange，美国标准信息交换码）。

ASCII 码规定，用 7 位二进制数表示 128 个符号。其中，1000001 表示大写字母 A，1000010 表示大写字母 B，以此类推。0100000 表示“空格”。计算机中多以字节为最小访问单位，每个字节是一个 8 位二进制数。为了方便使用，每个 ASCII 码在计算机中占用一个字节，最高位不用或用作其他用途。

前面的汇编指令 INC AX 中，字母 I 用二进制数 01001001 表示，DEBUG 中用十六进制表示为 49。同理，N、C、空格、A、X 也分别用一个 8 位二进制数表示。

汇编指令 INC AX 翻译成机器指令 01000000 的过程类似于一个查表过程。把汇编指令 INC AX 的 ASCII 码 49、4E、43、20、41、58 送入表中，在表中找到相应的行，就找到了相应的机器指令 01000000。实际的过程要复杂一些，大致的流程是这样的。

与汇编指令 INC AX 相对应的机器指令 01000000，其中包括两部分信息：一个是说明指令的功能是加 1，另一个是说明被加 1 的存储部件是寄存器 AX。说明指令功能的部分称作操作码，说明操作对象的部分称作地址码。按照地址码找到的操作对象称作操作数。机器指令 01000000，前 5 位 01000 是操作码，后 3 位 000 是地址码，000 是寄存器 AX 的编号，操作数就是寄存器 AX 中的数据。如果把 INC AX 中的 AX 换成其他的寄存器（BX、CX、DX、BP、SP、SI、DI），就会发现，只是地址码在变，操作码是不变的。

操作码实际上是一个编号。计算机所能进行的各种操作，统一编上号，操作码是哪个编号，计算机就进行对应的操作。

指令的操作对象称作操作数，保存在可以直接访问的存储部件，主要是寄存器和内存。指令确定操作数的方式称作寻址方式。例如，INC AX 这句指令，指令中直接给出了存有操作数的寄存器 AX 的编号，这种寻址方式叫做寄存器直接寻址。对应的机器指令 01000000 中，操作码 01000 不仅指定了指令的处理动作是加 1，而且指定了寻址方式是寄存器直接寻址。

再看一个例子。输入“A 101”，“回车”，进入输入汇编指令状态。输入“MOV CX，AX”、“回车”、“回车”，输入汇编指令并退回输入 DEBUG 命令状态。输入“RAX”、“回车”、“9”、“回车”，修改寄存器 AX 的内容。输入“R”、“回车”，显示寄存器的变化。输入“T＝

101”、“回车”，执行 MOV CX,AX 指令。屏幕上显示如图 3.3 所示的结果。

```
-A 101
1383:0101 MOV CX,AX
1383:0103
-R AX
AX 0000
:9
-R
AX=0009  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1383  ES=1383  SS=1383  CS=1383  IP=0100   NV UP EI PL NZ NA PO NC
1383:0100 0089C100      ADD     [BX+DI+00C1],CL                 DS:00C1=6E
-T=101

AX=0009  BX=0000  CX=0009  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1383  ES=1383  SS=1383  CS=1383  IP=0103   NV UP EI PL NZ NA PO NC
1383:0103 0000          ADD     [BX+SI],AL                      DS:0000=CD
-
```

图 3.3 执行 MOV CX,AX 指令

可以看到，寄存器 CX 的内容变得和 AX 一样了。这是一句传送指令，把寄存器 AX 中的数据传送到 CX 中。输入 U 101,101、“回车”，屏幕上显示如图 3.4 所示的结果。

```
-U 101,101
1383:0101 89C1          MOV     CX,AX
-
```

图 3.4 MOV CX,AX 的机器指令

可以看到，MOV CX,AX 对应的机器指令是 89C1，二进制形式是 1000100111000001。其中 1000100111 是操作码，指出这是一句在寄存器之间传输的指令。000 和 001 是地址码，是寄存器 AX 和 CX 的编号。

3.2 微程序控制器

寄存器之间都有数据线相连。寄存器的入口和出口都有开关，这些开关通常都是断开的。通过控制这些开关，就可以实现寄存器之间的数据传送功能。以指令 MOV CX,AX 为例：控制器把 AX 的出口开关接通，AX 中的数据就出现在数据线上了。再把寄存器 CX 的入口开关接通，数据线上的数据就进入 CX 了。再依次断开 CX 的入口开关和 AX 的出口开关，传送功能就完成了。

寄存器入口和出口的开关上都有控制端，通过在控制端上输入不同的电位，就可以控制开关的状态。例如，在控制端上输入高电位表示有效，开关就接通，输入零电位表示无效，开关就断开。

这些开关的控制端上的信号，来自一个叫做控制信号发生器的部件。控制信号发生器有多种实现方案，下面以微程序方案为例。在微程序方案中，开关控制端上的信号是从一个寄存器送来的，这个寄存器叫做微指令寄存器。微指令寄存器中存有一个二进制数，称作微指令。微指令的每一个数位对应一根寄存器的输出线。其中的两根输出线就接到了寄存器 AX 输出开关的控制端和 CX 输入开关的控制端，其余的输出线接到其他的控制端。在执行 MOV CX,AX 指令时，微指令对应 AX 的出口和 CX 的入口的两个数位为 1，其余的数位为 0。那么，微指令寄存器的输出线中，只有通向寄存器 AX 的出口和 CX 的入口的两根是有效的，其余的都是无效的。那么，寄存器 AX 的出口和 CX 的入口就接通了，计算机中其

余的开关都是断开的。这样，寄存器 AX 里的数据就传送到了 CX。

那么，微指令寄存器中的这个微指令又是从哪里来的呢？

这个微指令是从一个只读存储器 ROM 中读出来的。这个 ROM 叫做控制存储器。控制存储器中存有全部的微指令，计算机的每一种机器指令，这里都能找到与之对应的微指令。那么，控制存储器里有很多微指令，如何确定应该把哪一个读出来，送给微指令寄存器呢？

很简单，还是以这个 MOV CX，AX 指令为例。所有的机器指令的操作码都是各不相同的，MOV CX，AX 的操作码是 1000100111，其他指令的操作码都不是 1000100111。只要把 1000100111 作为地址，到控制存储器中查找这个位置，读出这个位置的微指令，写入微指令寄存器就可以了。

操作码是从指令寄存器里读出来的。指令寄存器中保存着正在执行的机器指令。这个机器指令是从内存读过来的。还有一个寄存器叫做指令指针寄存器，保存着机器指令在内存中的存放地址。指令寄存器里的机器指令是按照这个地址，从内存读出来的。在 DEBUG 的 R 命令中看到的寄存器 IP(Instruction Pointer)就是这个寄存器。

控制器就这样实现了“控制”。图 3.5 演示了控制器在执行指令 MOV CX，AX 时发挥

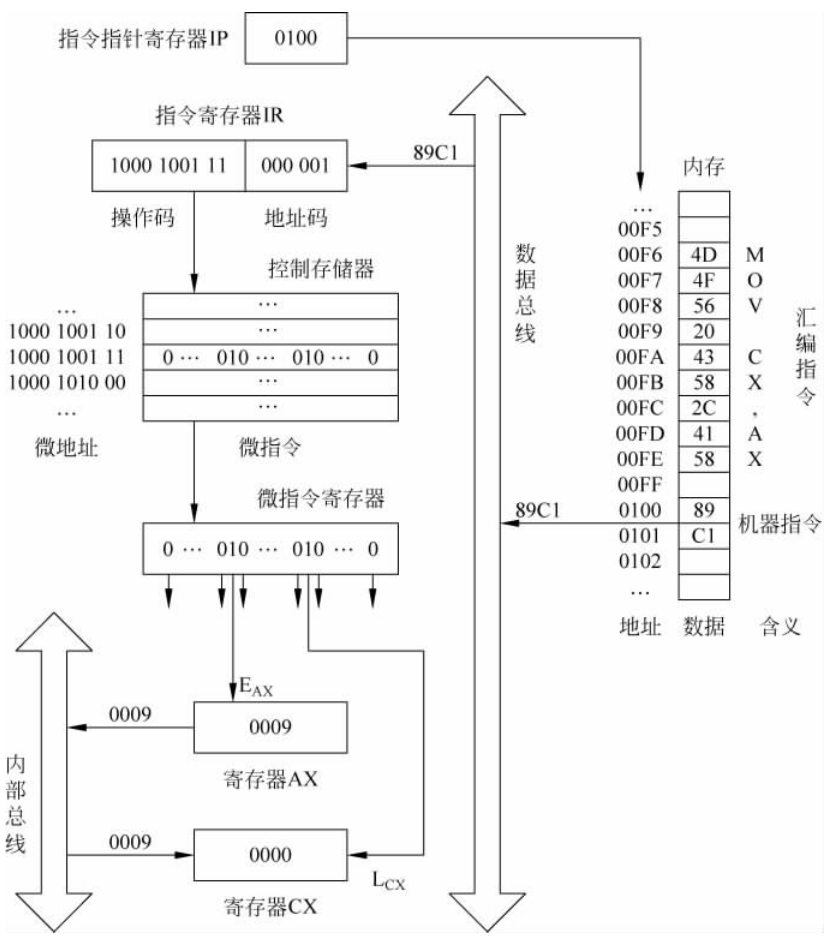

图 3.5 微程序控制器的工作流程

作用的流程。编写汇编指令 MOV CX,AX,以 ASCII 码的形式保存在内存中 00F6H～00FEH 存储单元中。然后翻译成机器指令 89C1H,保存在内存中 0100H 和 0101H 存储单元中。执行这条指令之前,要事先把它在内存中的存放位置 0100H 写到指令指针寄存器 IP 中。控制器按照 IP 中的指令地址 0100H,从内存中读出机器指令 89C1H,送入指令寄存器 IR。89C1H 的高 10 位 1000100111B 是操作码,控制器把它送到控制存储器,从控制存储器中 1000100111B 这个位置读出微指令 0…010…010…0,送入微指令寄存器。微指令寄存器中值为 1 的两个数位对应的输出线变成高电位,其余输出线为低电位。两条高电位的输出线分别接到了寄存器 AX 的三态输出门的使能端 E_{AX}和寄存器 CX 的装入门的装入端 L_{CX}。E_{AX}有效,寄存器 AX 内的数据 0009H 被送到了内部总线。L_{CX}有效,内部总线上的数据 0009H 被送入寄存器 CX。至此,指令 MOV CX,AX 的功能就完成了。当然,这是简化了的过程,实际的计算机还有很多细节。

一条指令的执行过程分为三个阶段。第一阶段把指令从内存中取出,送入指令寄存器 IR,称为取指令;第二阶段根据 IR 中的操作码形成控制信号,称为分析指令;第三阶段让控制信号发挥作用,完成指令的功能,称为执行指令。

指令指针寄存器 IP 保存的是下一条要执行的指令在内存中的保存位置。如图 3.6 所示,内存中存有三条机器指令。0100H 单元存有机器指令 40H,功能是把寄存器 AX 加 1。0101H 和 0102H 单元存有机器指令 01DAH,功能是把寄存器 BX 中的数据加到寄存器 DX 中。0103H～0105H 单元存有机器指令 B93412H,功能是把数据 1234H 存入寄存器 CX 中。执行这个程序段时,首先把程序段的首地址 0100H 存入指令指针寄存器 IP。控制器按照 IP 内的 0100H,到内存中找到 0100H 单元的指令 40H,把 40H 从内存中读出,送入指令寄存器 IR。此时,IP 的内容就变成了 0101H,指向了下一条指令 01DAH 的内存地址。至此,完成取指令。然后,控制器根据 IR 中的 40H,形成相应的控制信号,完成分析指令。再后,在控制信号的作用下,实现寄存器 AX 加 1 的功能,完成执行指令。第一条指令的执行到此完成。随后,控制器开始下一条指令的执行过程,按照 IP 内的 0101H,到内存中找到下一条指令 01DAH。当第二条指令 01DAH 取到指令寄存器 IR 之后,IP 的内容就变成了第三条指令 B93412H 的内存地址 0103H,然后进行第二条指令的分析指令和执行指令。

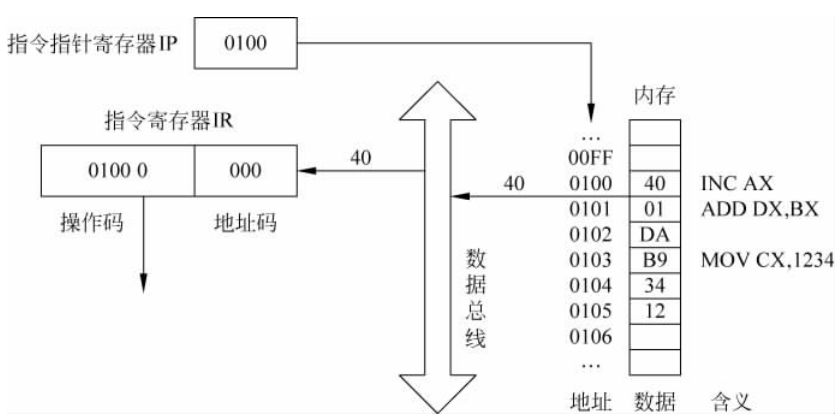

图 3.6 指令指针的作用

取指令也是在控制信号作用下完成的，也需要微指令寄存器提供控制信号。所以，在取指令之前，要在微指令寄存器中存入“取指微指令”。

操作码的作用已经充分展现了出来。下面看一下地址码是如何起作用的。在 DEBUG 中输入“A 103”、“回车”，进入输入汇编指令状态。输入“MOV DX,BX”、“回车”，输入汇编指令并退回输入 DEBUG 命令状态。输入“U 103,103”、“回车”。屏幕上出现如图 3.7 所示的结果。

```
-A 103
1383:0103 MOV DX,BX
1383:0105
-U 103,103
1383:0103 89DA          MOV     DX,BX
-
```

图 3.7　MOV DX,BX 指令

可以看到，MOV DX,BX 对应的机器指令是 89DA，二进制形式为 1000100111011010。其中，操作码是 1000100111，与 MOV CX,AX 的操作码是一样的。地址码不一样，011 是寄存器 BX 的编号，010 是寄存器 DX 的编号。

为了让地址码发挥作用，要对刚才的控制器工作流程作一下微调，如图 3.8 所示。微指

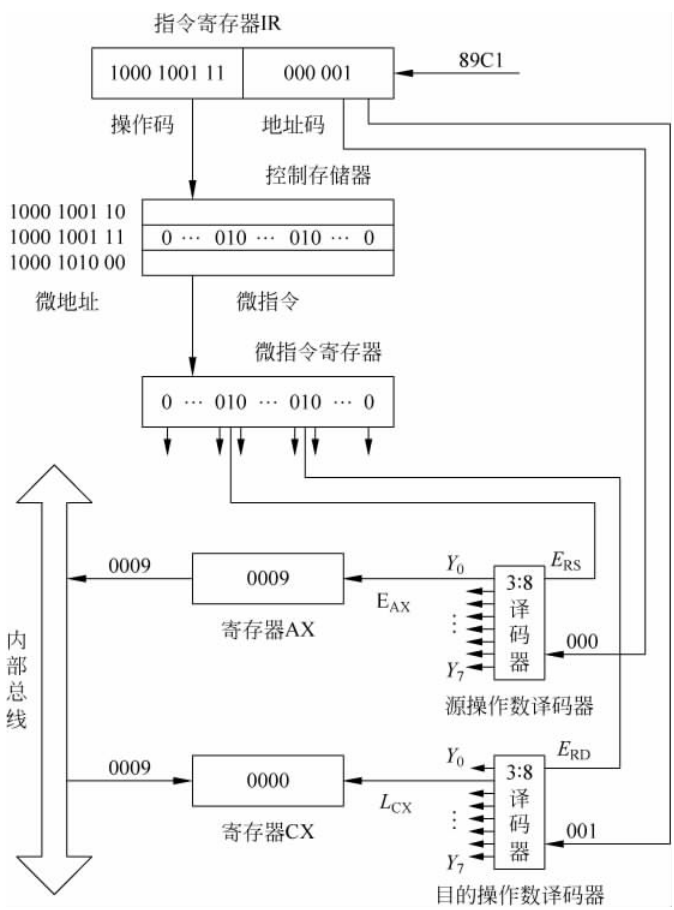

图 3.8　微指令控制器中的地址码

令寄存器中刚才用到的那两条输出线，不再直接接到寄存器 AX 的出口和 CX 的入口，而是接到两个 3∶8 译码器的使能端 E_{RS}和 E_{RD}。使能端收到有效信号时，译码器处于工作状态。其中一个译码器接收源操作数的地址码，例如代表 AX 的 000。这 8 个译码线分别接到了寄存器 AX、BX、CX、DX、BP、SP、SI 和 DI 的出口，控制这 8 个出口上的开关是否接通。这个译码器现在输入的是 000，对应的 0 号译码线有效，其余 7 个无效。0 号译码线接到了寄存器 AX 的出口控制端 E_{AX}，接通了开关，AX 里的数据 0009 就流出到数据线上了。另一个 3∶8 译码器接收目的操作数的地址码，例如，代表寄存器 CX 的 001。这 8 个译码线分别接到了寄存器 AX、BX、CX、DX、BP、SP、SI 和 DI 的入口，控制这 8 个入口上的开关是否接通。这个译码器现在输入的是 001，对应的 1 号译码线有效，其余 7 个无效。1 号译码线接到了寄存器 CX 的入口控制端 L_{CX}，接通了开关，数据线上的数据 0009 就流进了 CX 里。

指令中的地址码不仅可以是寄存器编号，也可以是内存地址、外部设备的端口地址或者操作数。例如：指令“MOV [1234]，AL”的功能是把寄存器 AL 中的数据送入内存 1234 单元中，对应的机器指令是 88263412，其中目的操作数地址码 3412 就是内存地址（机器指令中高字节写在右边）。指令“OUT 56，AL”的功能是把寄存器 AL 中的数据送给地址为 56 的外部设备端口，对应的机器指令是 E656，其中目的操作数地址码 56 是外部设备的端口地址。指令“MOV BL，78”的功能是把数据 78 送入寄存器 BL，对应的机器指令是 B078，其中源操作数地址码 78 是操作数。

3.3 控制信号的次序

现在分析一下指令“ADD DX，BX”的执行过程。这句指令的功能是把寄存器 BX 中的内容加到寄存器 DX 中。执行过程中，要先从寄存器 DX 中读出一个加数，送给运算器。加法的结果最后要写入寄存器 DX 中。很显然，发生在寄存器 DX 上的读出与写入两个动作不可能同时进行，要有一个先后次序。

为了控制动作的先后次序，计算机中需要图 3.9 中的这样一组信号。图 3.10 是发出这组信号的节拍发生器。

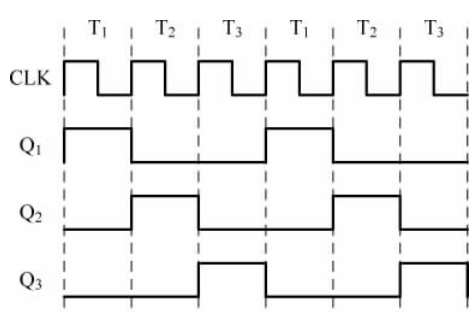

图 3.9 节拍的时序

如图 3.11 所示，在 Q_2节拍寄存器 DX 的输出使能端 E_{DX}有效，实现寄存器的读出数据操作。在 Q_3节拍寄存器 DX 的输入加载端 L_{DX}有效，实现寄存器的写入数据操作。在节拍信号 Q_2、Q_3作用下，实现了同一个寄存器先读后写的操作。

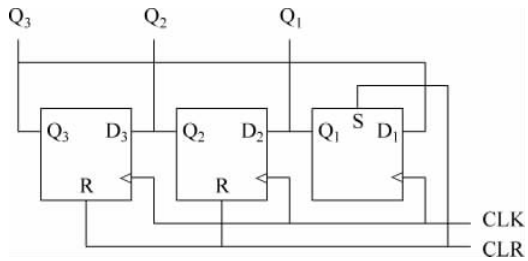

图 3.10 节拍发生器

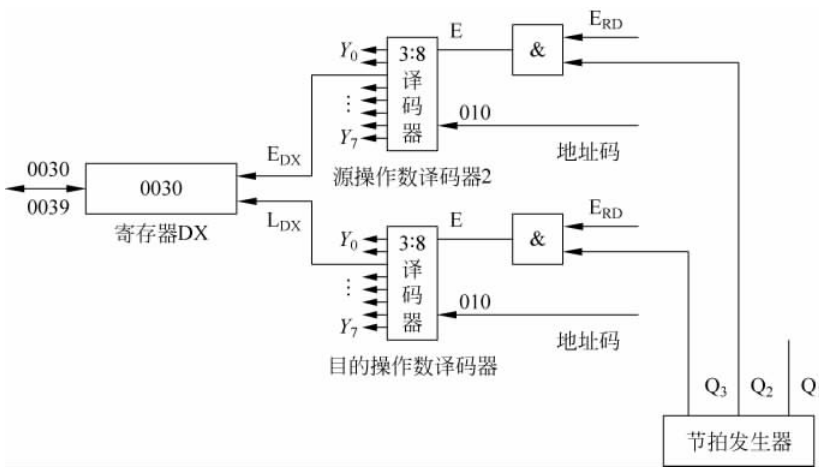

图 3.11 节拍控制下的寄存器读写次序

图 3.12 描述了指令“ADD DX,BX”的执行过程。这条汇编指令的功能是把寄存器 BX 中的数据加到寄存器 DX 里。ADD DX,BX 翻译后,相对应的机器指令为 01DA。其中,操作码为 0000000111。源操作数地址码为 011,这是寄存器 BX 的编号。目的操作数地址码为 010,这是寄存器 DX 的编号。根据操作码 0000000111,在控制存储器中找到微指令 0…010…01011110,并送到微指令寄存器。这条微指令中有 6 个 1,对应的 6 条控制线有效。在 Q_1 节拍,控制线 E_{RS} 和 L_L 有效。控制线 E_{RS} 使源操作数译码器有效,该译码器根据源操作数地址码 011,找到寄存器 BX 的输出使能端 E_{BX},在 Q_1 节拍的配合下打开寄存器 BX 的出口,使其中的数据 0009 送到内部总线。同时,另一条控制线 L_L 打开运算器的左暂存器载入控制端,使总线上的数据 0009 进入左暂存器。到了 Q_2 节拍,控制线 E_{RS} 和 L_L 变无效,E_{RD} 和 L_R 变有效。控制线 E_{RD} 使目的操作数译码器有效,该译码器根据目的操作数地址码 010,找到寄存器 DX 的输出使能端 E_{DX},在 Q_2 节拍的配合下打开寄存器 DX 的出口,使得寄存器 DX 中的数据 0030 送到内部总线。同时,控制线 L_R 打开右暂存器的载入控制端,使总线上的数据 0030 进入右暂存器。暂存器与运算器 ALU 之间是没有控制端的,运算器在控制线 C_{ADD} 的作用下,加法的结果很快就会出现在运算器的出口。到了 Q_3 节拍,控制线 L_R 变无效,E_{ALU} 变有效,E_{RD} 仍然有效。控制线 E_{ALU} 在 Q_3 节拍的配合下,打开运算器的输出控制端,把加法的结果 0039 送到内部总线。同时,控制线 E_{RD} 在节拍 Q_3 的配合下,使得寄存器 DX 的载入控制端 L_{DX} 有效,把总线上的加法结果 0039 送进寄存器 DX。

功能复杂的机器指令无法用一句微指令提供全部的控制信号,需要多条微指令依次提供控制信号。这些为同一个机器指令提供控制信号的微指令序列,被称作微程序。

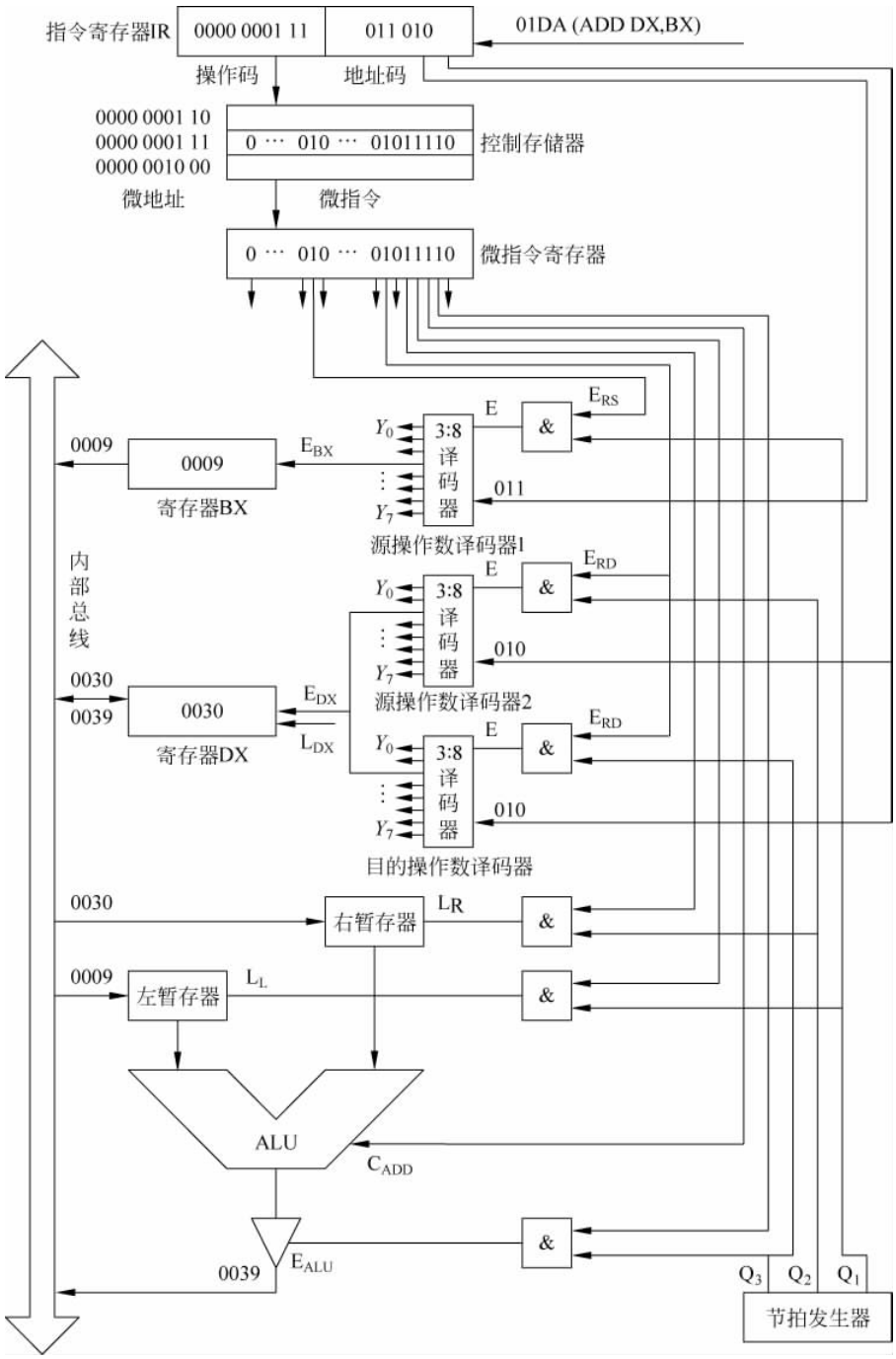

图 3.12 控制信号的次序

3.4 组合逻辑电路控制器

除了微程序方式,控制器也可以采用组合逻辑电路的方式实现。下面以图 3.5 中用到的指令 89C1H(MOV CX,AX)和图 3.12 中用到的指令 01DAH(ADD DX,BX)为例。

首先，需要一个译码器。这个译码器输入的是指令寄存器 IR 输出的操作码，所以这个译码器被称为指令译码器(Instruction Decoder，ID)，如图 3.13 所示。

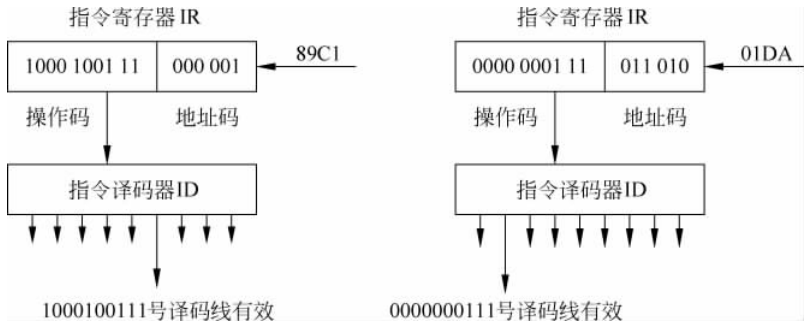

图 3.13 指令译码器 ID

指令译码器的输入端输入不同的操作码，与这个操作码对应的译码线就会输出有效信号，其余的译码线输出无效信号。

如图 3.14 所示，当指令寄存器中的指令是 89C1H(MOV CX，AX)时，这个指令的控制信号是 E_{RS}和 E_{RD}。指令 89C1H 的操作码 1000100111 送入指令译码器 ID，指令译码器 ID 的 1000100111 号译码线有效。把这条译码线作为控制信号 E_{RS}和 E_{RD}，分别接到源操作数

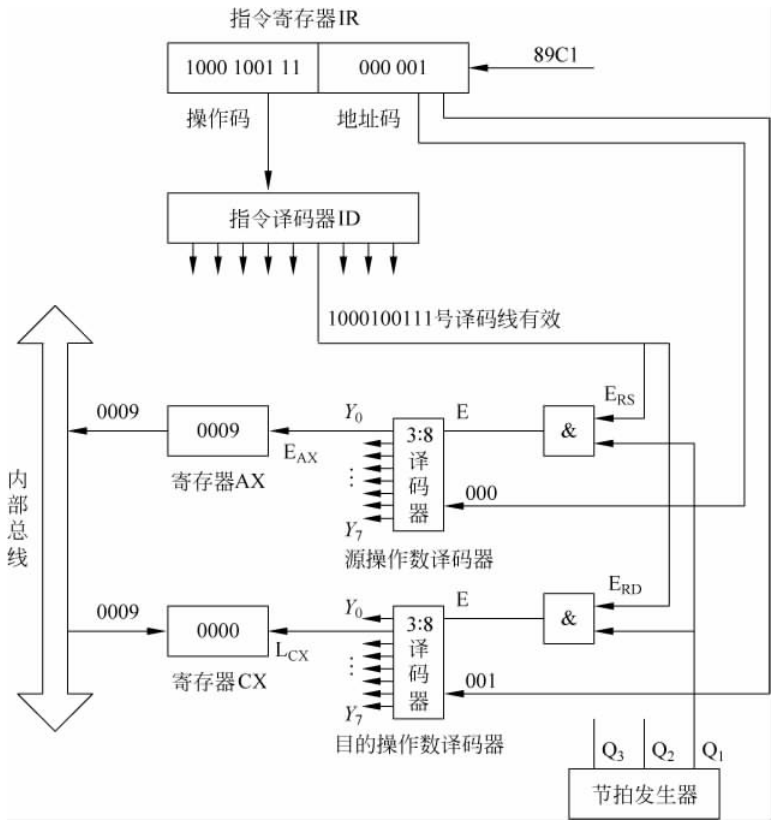

图 3.14 组合逻辑控制 MOV 指令

译码器的使能端 E_{RS}和目的操作数译码器的使能端 E_{RD}，再由地址码向源操作数译码器提供寄存器 AX 的编号 000，向目的操作数译码器提供寄存器 CX 的编号 001，这句传数指令的功能就可以实现了。

如图 3.15 所示，当指令寄存器中的指令是 01DAH(ADD DX,BX)时，这个指令的控制信号是 E_{RS}、L_L、E_{RD}、L_R、C_{ADD}和 E_{ALU}。指令 01DAH 的操作码 0000000111 送入指令译码器 ID，

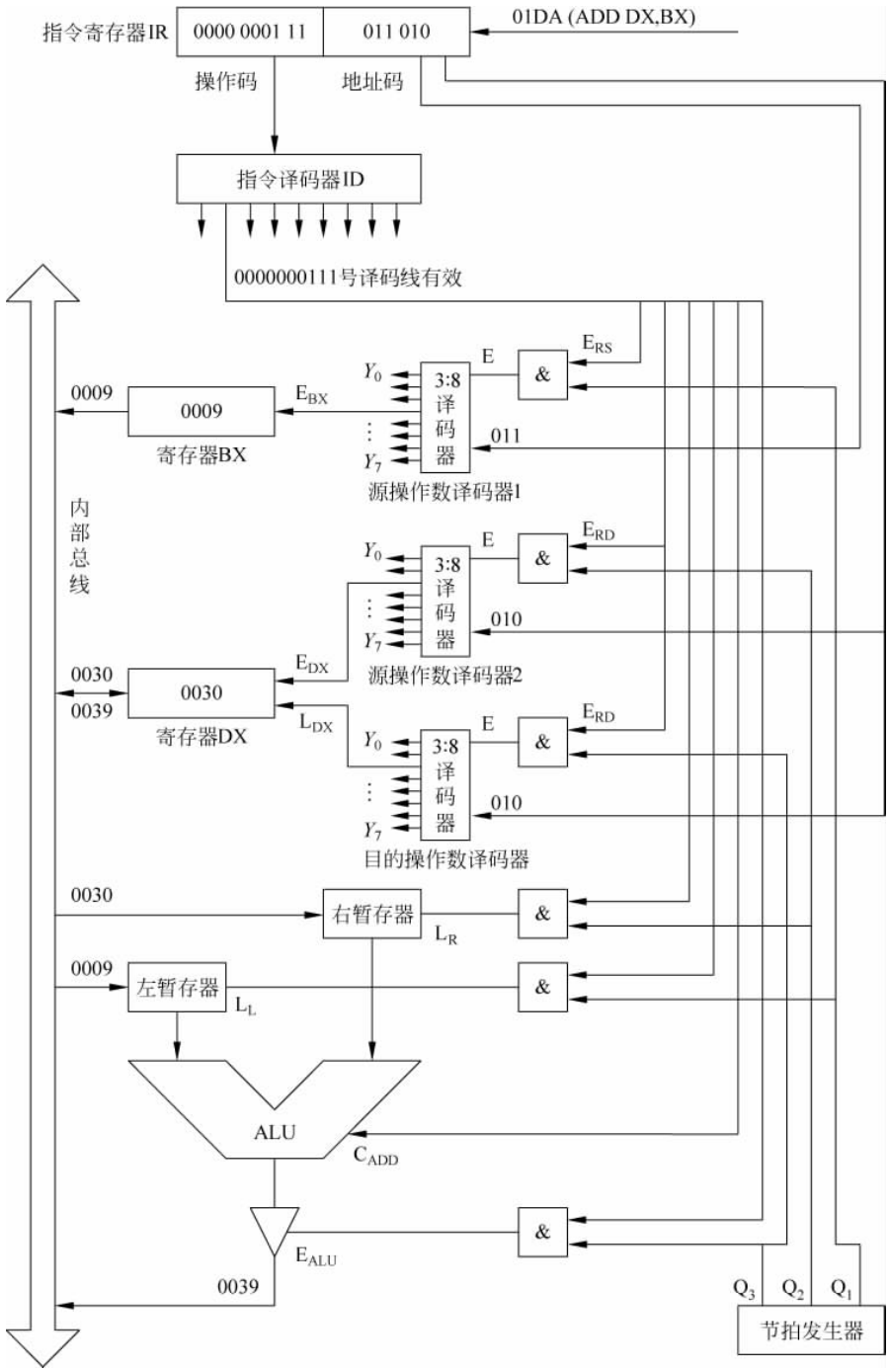

图 3.15 组合逻辑控制 ADD 指令

指令译码器 ID 的 0000000111 号译码线有效。把这条译码线作为控制信号 E_{RS}、L_L、E_{RD}、L_R、C_{ADD}和 E_{ALU}，分别接到相应的被控制部件，在节拍信号 Q_1、Q_2、Q_3的配合下，完成加法功能。

在图 3.16 中，组合逻辑电路可以同时控制上述两种指令。其中的虚线部分称作控制信号发生器。它的输入信号有指令译码器的全部译码线和节拍发生器的节拍信号，输出信号有各种控制信号。

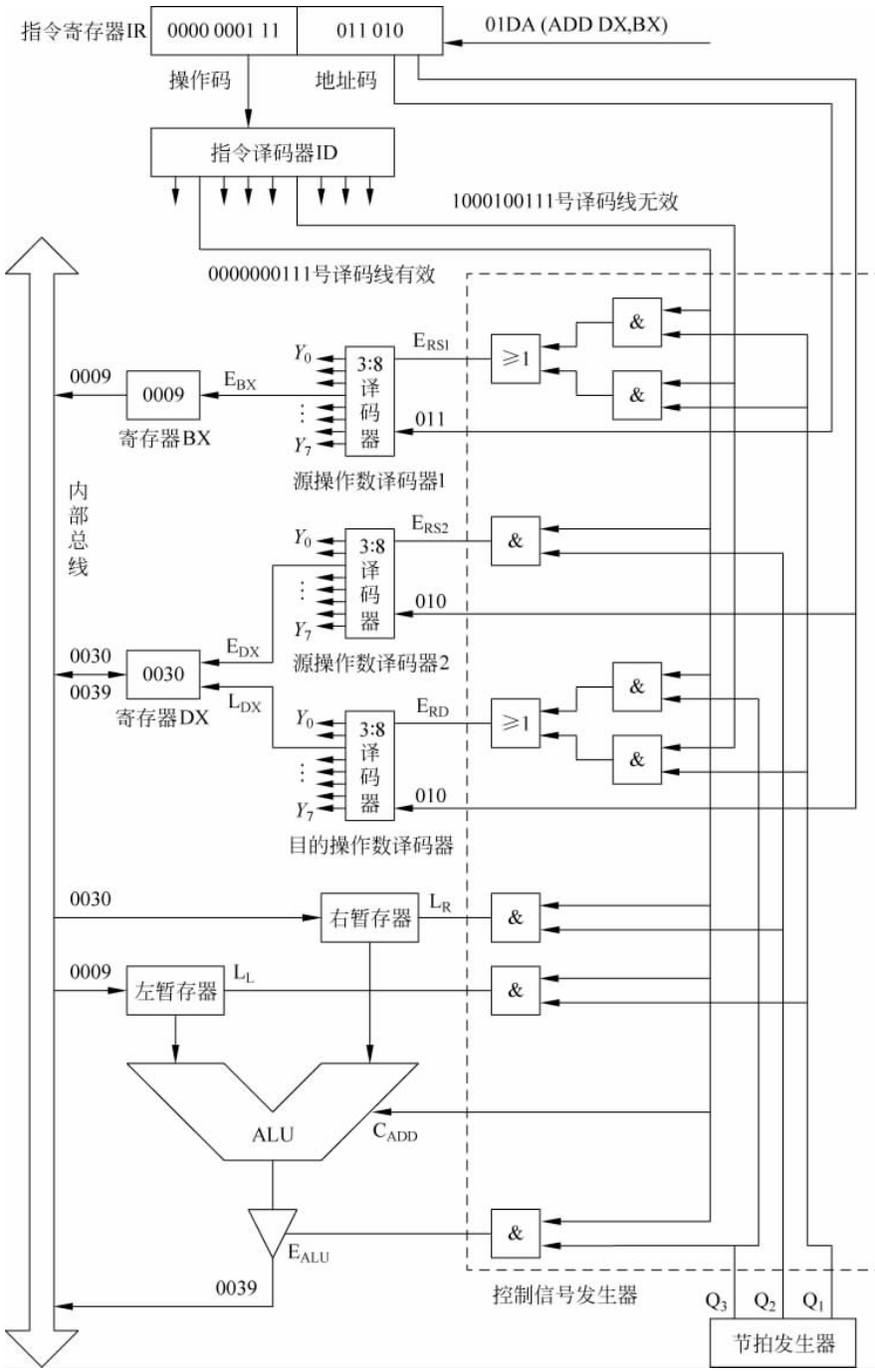

图 3.16　控制 MOV 和 ADD 两种指令的逻辑电路

3.5 逻辑表达式的化简

图 3.16 中的组合逻辑电路是可以化简的。逻辑电路与逻辑表达式有对应关系。写出需要化简的逻辑电路所对应的逻辑表达式，化简这个表达式。然后再画出与简化后的逻辑表达式对应的逻辑电路，就把逻辑电路化简了。

布尔代数是常用的逻辑表达式工具。这里只讨论与、或、非运算和常用运算规律。

3.5.1 与运算

与运算的规则是这样的：

$$0 \cdot 0 = 0 \qquad 0 \cdot 1 = 0 \qquad 1 \cdot 0 = 0 \qquad 1 \cdot 1 = 1$$

其中，符号“·”表示与运算。计算机中的数据都是多位二进制数。两个多位二进制数进行与运算时，对应位的两个数进行运算，不同数位之间没有影响。

【例 3.1】 $X=00111100$，$Y=01100101$。求 X 和 Y 的与运算结果。

$$\begin{array}{r} 0\,0\,1\,1\,1\,1\,0\,0 \\ \cdot\quad 0\,1\,1\,0\,0\,1\,0\,1 \\ \hline 0\,0\,1\,0\,0\,1\,0\,0 \end{array}$$

答：X 和 Y 的与运算结果为 00100100。

ASM86 汇编语言中，AND 指令的功能是对两个数进行与运算。

图 3.17 中的寄存器 AL、BL 指的是寄存器 AX、BX 的低位部分。AX、BX、CX、DX 都是 16 位寄存器，可以分开当作 8 位寄存器用，高位部分分别称作 AH、BH、CH、DH，低位部分分别称作 AL、BL、CL、DL。送入 AL 中的 3C，就是例题 3.1 中的 X，送入 BL 中的 65，就是 Y，用 G 命令执行这三句指令之后，AL 中的 24，就是 X 和 Y 与操作的结果。

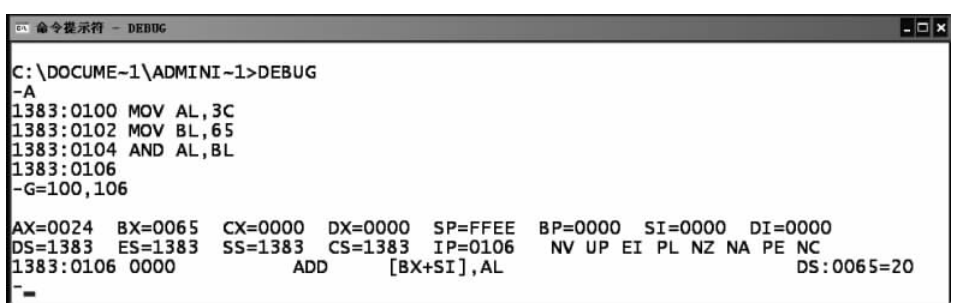

图 3.17 与指令 AND 的执行效果

3.5.2 或运算

或运算的规则是这样的：

$$0 + 0 = 0 \qquad 0 + 1 = 0 \qquad 1 + 0 = 0 \qquad 1 + 1 = 1$$

其中，符号“+”表示或运算。两个多位二进制数进行或运算时，对应位的两个数进行运算，不同数位之间没有影响。

【例 3.2】 $X=00111100$，$Y=01100101$。求 X 和 Y 的或运算结果。

$$\begin{array}{r} 0\,0\,1\,1\,1\,1\,0\,0 \\ +\quad 0\,1\,1\,0\,0\,1\,0\,1 \\ \hline 0\,1\,1\,1\,1\,1\,0\,1 \end{array}$$

答：X 和 Y 的或运算结果为 01111101。

ASM86 汇编语言中，OR 指令的功能是对两个数进行或运算，如图 3.18 所示。

```
-A
1381:0106 MOV AL,3C
1381:0108 MOV BL,65
1381:010A OR AL,BL
1381:010C
-G=106,10C

AX=007D  BX=0065  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1381  ES=1381  SS=1381  CS=1381  IP=010C   NV UP EI PL NZ NA PE NC
1381:010C 0000          ADD     [BX+SI],AL                      DS:0065=20
-
```

图 3.18　或指令 OR 的执行效果

3.5.3　非运算

非运算的规则是这样的：

$$\overline{0}=1 \qquad \overline{1}=0$$

上画线表示非运算。多位二进制数进行非运算时，不同数位之间没有影响。

【例 3.3】 $Y=01100101$。求 Y 的非运算结果。

$$\overline{Y}=1\,0\,0\,1\,1\,0\,1\,0$$

答：Y 的非运算结果为 10011010。

ASM86 汇编语言中，NOT 指令的功能是对数据进行非运算，如图 3.19 所示。

```
-A10C
1381:010C MOV BL,65
1381:010E NOT BL
1381:0110
-G=10C,110

AX=0000  BX=009A  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1381  ES=1381  SS=1381  CS=1381  IP=0110   NV UP EI PL NZ NA PO NC
1381:0110 0000          ADD     [BX+SI],AL                      DS:009A=2E
-
```

图 3.19　非指令 NOT 的执行效果

3.5.4　常用运算规律

1. 恒等式

$$A\cdot 0=0 \qquad A\cdot 1=A \qquad A\cdot A=A$$
$$A+0=A \qquad A+1=1 \qquad A+A=A$$

2. 交换律

$$A\cdot B=B\cdot A \qquad A+B=B+A$$

3. 结合律

$$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C$$
$$(A + B) + C = A + (B + C) = A + B + C$$

4. 分配律

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$
$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

5. 摩根定理

$$\overline{A+B} = \overline{A} \cdot \overline{B} \qquad \overline{A \cdot B} = \overline{A} + \overline{B}$$

图 3.20 是源寄存器译码器的使能端 E_{RS1} 的形成电路。

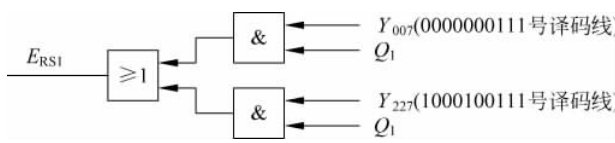

图 3.20 控制信号 E_{RS1} 的形成电路

这个电路对应的逻辑表达式为：$E_{RS1} = Y_{007} \cdot Q_1 + Y_{227} \cdot Q_1$。化简后的逻辑表达式为：$E_{RS1} = (Y_{007} + Y_{227}) \cdot Q_1$。化简后的逻辑表达式对应的电路如图 3.21 所示。

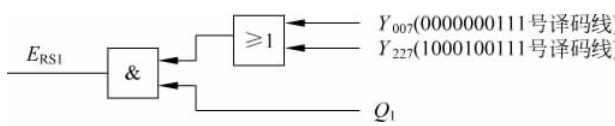

图 3.21 化简后的 E_{RS1} 的形成电路

经过化简之后，电路的逻辑门从三个减少到两个。

习题 3

3.1 什么是机器语言？

3.2 什么是操作码？什么是操作数？

3.3 控制器是如何控制数据进入寄存器的？

3.4 什么是微指令？什么是微程序？

3.5 微程序与机器指令是什么关系？

3.6 一条机器指令执行过程中，分哪几个阶段？

3.7 控制器是如何实现控制信号的次序的？

3.8 指令指针 IP 有什么作用？

3.9 组合逻辑电路控制器的控制信号发生器有哪些输入信号和输出信号？

3.10 化简下面的逻辑表达式。

$$B \cdot (\overline{A} + C) + A \cdot (B + \overline{C})$$