$T_2$　$R_2 \rightarrow Bus \rightarrow E$

(3) 完成“STA @ mem”指令所需的微操作命令及节拍安排如下：

取指周期

$T_0$　$PC \rightarrow Bus \rightarrow MAR, 1 \rightarrow R$

$T_1$　$M(MAR) \rightarrow MDR, (PC) + 1 \rightarrow PC$

$T_2$　$MDR \rightarrow Bus \rightarrow IR, OP(IR) \rightarrow$微操作命令形成部件

间址周期

$T_0$　$Ad(IR) \rightarrow Bus \rightarrow MAR, 1 \rightarrow R$

$T_1$　$M(MAR) \rightarrow MDR$

执行周期

$T_0$　$MDR \rightarrow Bus \rightarrow MAR, 1 \rightarrow W$

$T_1$　$ACC \rightarrow Bus \rightarrow MDR$

$T_2$　$MDR \rightarrow M(MAR)$

**例 10.3**　设寄存器均为16位，实现补码 Booth 算法的运算器框图如图 6.9 所示。其中寄存器 A、X 最高2位 $A_0$、$A_1$ 和 $X_0$、$X_1$ 为符号位，寄存器 Q 最高位 $Q_0$ 为符号位，最末位 $Q_{15}$ 为附加位。假设上条指令的运行结果存于 A(即被乘数)中。

(1) 若 CU 为组合逻辑控制，且采用中央和局部控制相结合的方法，写出完成“MUL α”(α为主存地址)指令的全部微操作命令及节拍安排。

(2) 指出哪些节拍属于中央控制节拍，哪些节拍属于局部控制节拍，局部控制最多需要几拍？

**解**：(1) 取指阶段

$T_0$　$PC \rightarrow MAR, 1 \rightarrow R$

$T_1$　$M(MAR) \rightarrow MDR, (PC) + 1 \rightarrow PC$

$T_2$　$MDR \rightarrow IR, OP(IR) \rightarrow ID$

执行阶段

乘法开始前要将被乘数由 $A \rightarrow X$，并将乘数从主存 α 单元取出送至 Q 寄存器。因 $Q_{15}$(最末位)为附加位，还必须 $0 \rightarrow Q_{15}$，并将 A 清零。上述这些操作可安排在中央控制节拍内完成。乘法过程的重复加操作受 Q 寄存器末两位 $Q_{14}$、$Q_{15}$ 控制，重复移位操作在两个串接的寄存器 A//Q 中完成，这两种操作可安排在局部控制节拍内完成。具体安排如下：

$T_0$　$Ad(IR) \rightarrow MAR, 1 \rightarrow R, A \rightarrow X$

$T_1$　$M(MAR) \rightarrow MDR, 0 \rightarrow Q_{15}, 0 \rightarrow A$

$T_2$　$MDR \rightarrow Q_{0\sim14}$　　(Q 寄存器仅取1位符号位)

$T_0^*$　$\overline{Q}_{14}Q_{15} \cdot (A + X) + Q_{14}\overline{Q}_{15} \cdot (A+\overline{X}+1) + \overline{Q}_{14}\,\overline{Q}_{15} \cdot A + Q_{14}Q_{15} \cdot A \rightarrow A$

$T_1^*$　$L(A//Q) \rightarrow R(A//Q)$　(A//Q 算术右移一位)

⋮

（2）中央控制节拍包括取指阶段所有节拍和执行阶段的 $T_0$、$T_1$、$T_2$ 3 个节拍，完成取指令和取操作数及乘法运算前的准备工作。局部控制节拍是执行阶段的 $T_0^*$ 和 $T_1^*$ 节拍，其中 $T_0^*$ 为重复加操作，受 Q 寄存器末两位 $Q_{14}$、$Q_{15}$控制，最多执行 15 次；$T_1^*$ 为移位操作，共执行 14 次。

### 10.1.3　组合逻辑设计步骤

采用组合逻辑设计控制单元时，首先根据上述 10 条指令微操作的节拍安排，列出微操作命令的操作时间表，然后写出每一个微操作命令（控制信号）的逻辑表达式，最后根据逻辑表达式画出相应的组合逻辑电路图。

1. 列出微操作命令的操作时间表

表 10.1 列出了上述 10 条机器指令微操作命令的操作时间表。表中 FE、IND 和 EX 为 CPU 工作周期标志（参见图 8.9），$T_0 \sim T_2$ 为节拍，I 为间址标志，在取指周期的 $T_2$ 时刻，若测得 I=1，则 IND 触发器置“1”，标志进入间址周期；若 I=0，则 EX 触发器置“1”，标志进入执行周期。同理，在间址周期的 $T_2$ 时刻，若测得 IND=0（表示一次间接寻址），则 EX 触发器置“1”，进入执行周期；若测得 IND=1（表示多次间接寻址），则继续间接寻址。在执行周期的 $T_2$ 时刻，CPU 要向所有中断源发中断查询信号，若检测到有中断请求并且满足响应条件，则 INT 触发器置“1”，标志进入中断周期。表中未列出 INT 触发器置“1”的操作和中断周期的微操作。表中第一行对应 10 条指令的操作码，代表不同的指令。若某指令有表中所列的微操作命令，其对应的空格内为 1。

2. 写出微操作命令的最简逻辑表达式

纵览表 10.1 便可列出每一个微操作命令的初始逻辑表达式，经化简、整理便可获得能用现成电路实现的微操作命令逻辑表达式。

例如，根据表可写出 M(MAR)→MDR 微操作命令的逻辑表达式：

$$\begin{aligned}&\text{M(MAR)}\rightarrow\text{MDR}\\&=\text{FE}\cdot T_1+\text{IND}\cdot T_1(\text{ADD}+\text{STA}+\text{LDA}+\text{JMP}+\text{BAN})+\text{EX}\cdot T_1(\text{ADD}+\text{LDA})\\&=T_1\{\text{FE}+\text{IND}(\text{ADD}+\text{STA}+\text{LDA}+\text{JMP}+\text{BAN})+\text{EX}(\text{ADD}+\text{LDA})\}\end{aligned}$$

式中，ADD、STA、LDA、JMP、BAN 均来自操作码译码器的输出。

3. 画出微操作命令的逻辑图

对应每一个微操作命令的逻辑表达式都可画出一个逻辑图。例如，M(MAR)→MDR 的逻辑表达式所对应的逻辑图如图 10.3 所示，图中未考虑门的扇入系数。

表 10.1 操作时间表

| 工作周期标记 | 节拍 | 状态条件 | 微操作命令信号 | CLA | COM | SHR | CSL | STP | ADD | STA | LDA | JMP | BAN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FE（取指） | $T_0$ | | PC→MAR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | 1→R | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $T_1$ | | M(MAR)→MDR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | (PC)+1→PC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $T_2$ | | MDR→IR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | OP(IR)→ID | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | I | I→IND | | | | | | 1 | 1 | 1 | 1 | 1 |
| | | $\bar{I}$ | I→EX | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| IND（间接寻址） | $T_0$ | | Ad(IR)→MAR | | | | | | 1 | 1 | 1 | 1 | 1 |
| | | | I→R | | | | | | 1 | 1 | 1 | 1 | 1 |
| | $T_1$ | | M(MAR)→MDR | | | | | | 1 | 1 | 1 | 1 | 1 |
| | $T_2$ | | MDR→Ad(IR) | | | | | | 1 | 1 | 1 | 1 | 1 |
| | | $\overline{IND}$ | I→EX | | | | | | 1 | 1 | 1 | 1 | 1 |
| EX（执行） | $T_0$ | | Ad(IR)→MAR | | | | | | 1 | 1 | 1 | | |
| | | | 1→R | | | | | | 1 | | 1 | | |
| | | | 1→W | | | | | | | 1 | | | |
| | $T_1$ | | M(MAR)→MDR | | | | | | 1 | | 1 | | |
| | | | AC→MDR | | | | | | | 1 | | | |
| | $T_2$ | | (AC)+(MDR)→AC | | | | | | 1 | | | | |
| | | | MDR→M(MAR) | | | | | | | 1 | | | |
| | | | MDR→AC | | | | | | | | 1 | | |
| | | | 0→AC | 1 | | | | | | | | | |
| | | | $\overline{AC}$→AC | | 1 | | | | | | | | |
| | | | L(AC)→R(AC)，$AC_0$ 不变 | | | 1 | | | | | | | |
| | | | $\rho^{-1}$(AC) | | | | 1 | | | | | | |
| | | | Ad(IR)→PC | | | | | | | | | 1 | |
| | | $A_0$ | Ad(IR)→PC | | | | | | | | | | 1 |
| | | | 0→G | | | | | 1 | | | | | |

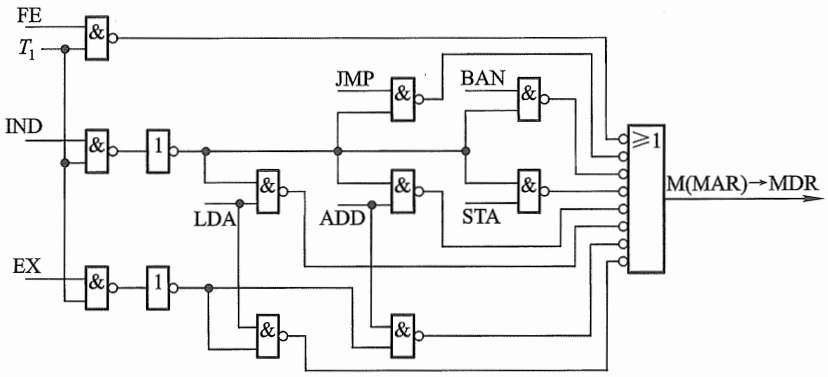



图 10.3　产生 M(MAR)→MDR 命令的逻辑图

当然,在设计逻辑图时要考虑门的扇入系数和逻辑级数。如果采用现成芯片,还需选择芯片型号。

采用组合逻辑设计方法设计控制单元,思路清晰,简单明了,但因为每一个微操作命令都对应一个逻辑电路,因此一旦设计完毕便会发现,这种控制单元的线路结构十分庞杂,也不规范,犹如一棵大树,到处都是不规整的枝杈。而且指令系统功能越全,微操作命令就越多,线路也越复杂,调试就更困难。为了克服这些缺点,可采用微程序设计方案。但是,正如 7.5 节所述,随着 RISC 的出现,组合逻辑设计仍然是设计计算机的一种重要方法。

## 10.2　微程序设计

### 10.2.1　微程序设计思想的产生

微程序设计思想是英国剑桥大学教授 M.V.Wilkes 在 1951 年首先提出的。为了克服组合逻辑控制单元线路庞杂的缺点,他大胆设想采用与存储程序相类似的方法,来解决微操作命令序列的形成。Wilkes 提出,将一条机器指令编写成一个微程序,每一个微程序包含若干条微指令,每一条微指令对应一个或几个微操作命令。然后把这些微程序存到一个控制存储器中,用寻找用户程序机器指令的方法来寻找每个微程序中的微指令。由于这些微指令是以二进制代码形式表示的,每位代表一个控制信号(若该位为 1,表示该控制信号有效;若该位为 0,表示此控制信号无效),因此,逐条执行每一条微指令,也就相应地完成了一条机器指令的全部操作。可见,微程序控制单元的核心部件是一个控制存储器。由于执行一条机器指令必须多次访问控制存储器,以取出多条微指令来控制执行各个微操作,因此要求控制存储器的速度较高。可惜在 Wilkes 那个

年代电子器件生产水平有限，因此微程序设计思想并未实现。直到20世纪60年代出现了半导体存储器，才使这个设计思想成为现实。1964年4月，世界上第一台微程序设计的机器IBM 360研制成功。

微程序设计省去了组合逻辑设计过程中对逻辑表达式的化简步骤，无须考虑逻辑门级数和门的扇入系数，使设计更简便，而且由于控制信号是以二进制代码的形式出现的，因此只要修改微指令的代码，就可改变操作内容，便于调试、修改，甚至增删机器指令，有利于计算机仿真。

## 10.2.2　微程序控制单元框图及工作原理

1. 机器指令对应的微程序

采用微程序设计方法设计控制单元的过程就是编写每一条机器指令的微程序，它是按执行每条机器指令所需的微操作命令的先后顺序而编写的，因此，一条机器指令对应一个微程序，如图10.4所示。图中每一条机器指令都与一个以操作性质命名的微程序对应。

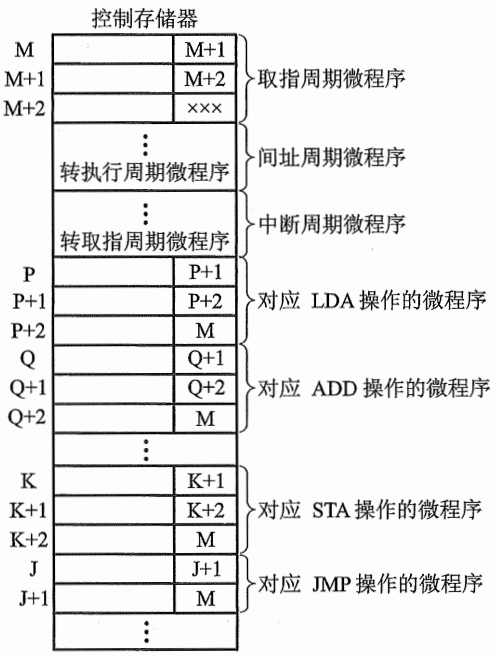



图10.4　不同机器指令所对应的微程序

由于任何一条机器指令的取指令操作是相同的，因此将取指令操作的命令统一编成一个微程序，这个微程序只负责将指令从主存单元中取出送至指令寄存器中，如图10.4所示的取指周期微程序。此外，如果指令是间接寻址，其操作也是可以预测的，也可先编出对应间址周期的微程序。当出现中断时，中断隐指令所需完成的操作可由一个对应中断周期的微程序控制完成。这样，控制存储器中的微程序个数应为机器指令数再加上对应取指、间接寻址和中断周期的3个微程序。

2. 微程序控制单元的基本框图

图10.5示意了微程序控制单元的基本组成。

图中点画线框内为微程序控制单元，与图9.2相比，它们都有相同的输入，如指令寄存器、各种标志和时钟，输出也是输至CPU内部或系统总线的控制信号。

点画线框内的控制存储器（简称控存）是微程序控制单元的核心部件，用来存放全部微程序；CMAR是控存地址寄存器，用来存放欲读出的微指令地址；CMDR是控存数据寄存器，用来存放从控存读出的微指令；顺序逻辑是用来控制微指令序列的，具体就是控制形成下一条微指令（即后续微指令）的地址，其输入与微地址形成部件（与指令寄存器相连）、微指令的下地址字段以及外来

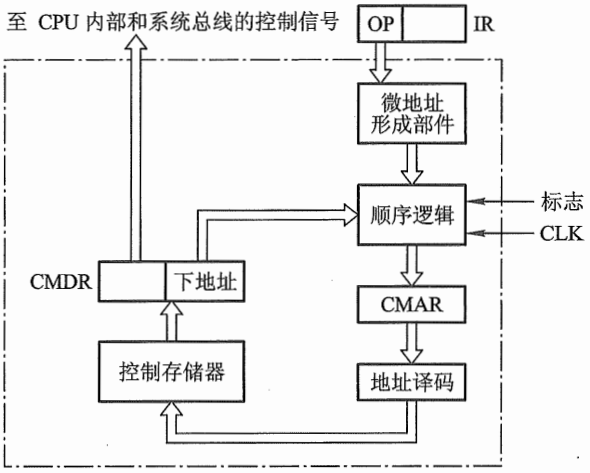



图 10.5　微程序控制单元的基本组成

的标志有关。有关微指令序列地址的形成将在 10.2.4 节中介绍。

微指令的基本格式如图 10.6 所示，共分两个字段，一个为操作控制字段，该字段发出各种控制信号；另一个为顺序控制字段，它可指出下条微指令的地址（简称下地址），以控制微指令序列的执行顺序。

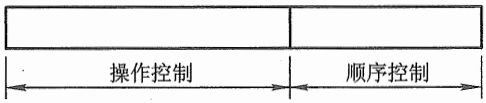



图 10.6　微指令的基本格式

3. 工作原理

假设有一个用户程序如下所示，它存于以 2000H 为首地址的主存空间内。

```
LDA X
ADD Y
STA Z
STP
```

下面结合图 10.4 和图 10.5，分析运行上述程序时微程序控制单元的工作原理。

首先将用户程序的首地址送至 PC，然后进入取指阶段。

（1）取指阶段

① 将取指周期微程序首地址 M→CMAR。

② 取微指令。

将对应控存 M 地址单元中的第一条微指令读到控存数据寄存器中，记作 CM(CMAR)→CMDR。

③ 产生微操作命令。

第一条微指令的操作控制字段中为“1”的各位发出控制信号，如 PC→MAR，1→R，命令主存

接收程序首地址并进行读操作。

④ 形成下一条微指令的地址。

此微指令的顺序控制字段指出了下一条微指令的地址为 M+1,将 M+1 送至 CMAR,即 Ad(CMDR)→CMAR。

⑤ 取下一条微指令。

将对应控存 M+1 地址单元中的第二条微指令读到 CMDR 中,即 CM(CMAR)→CMDR。

⑥ 产生微操作命令。

由第二条微指令的操作控制字段中对应“1”的各位发出控制信号,如 M(MAR)→MDR 使对应主存 2000H 地址单元中的第一条机器指令从主存中读出送至 MDR 中。

⑦ 形成下一条微指令的地址。

将第二条微指令下地址字段指出的地址 M+2 送至 CMAR,即

Ad(CMDR) →CMAR

⋮

以此类推,直到取出取指周期最后一条微指令,并发出微操作命令为止。此时第一条机器指令“LDA X”已存至指令寄存器 IR 中。

(2) 执行阶段

① 取数指令微程序首地址的形成。

当取数指令存入 IR 后,其操作码 OP(IR)直接送到微地址形成部件,该部件的输出即为取数指令微程序的首地址 P,且将 P 送至 CMAR,记作 OP(IR)→微地址形成部件→CMAR。

② 取微指令。

将对应控存 P 地址单元中的微指令读到 CMDR 中,即 CM(CMAR)→CMDR。

③ 产生微操作命令。

由微指令操作控制字段中对应“1”的各位发出控制信号,如 Ad(IR)→MAR,1→R,命令主存读操作数。

④ 形成下一条微指令的地址。

将此条微指令下地址字段指出的 P+1 送至 CMAR,即 Ad(CMDR) →CMAR。

⑤ 取微指令,即 CM(CMAR) →CMDR。

⑥ 产生微操作命令。

⋮

以此类推,直到取出取数指令微程序的最后一条微指令 P+2,并发出微操作命令。至此即完成了将主存 X 地址单元中的操作数取至累加器 AC 的操作。这条微指令的顺序控制字段为 M,即表明 CPU 又开始进入下一条机器指令的取指周期,控存又要依次读出取指周期微程序的逐条微指令,发出微操作命令,完成将第二条机器指令“ADD Y”从主存取至指令寄存器 IR 中……微程序控制单元就是这样,通过逐条取出微指令,发出各种微操作命令,从而实现从主存逐条取出、分析并执行机器指令,以达到运行程序的目的。

由此可见，对微程序控制单元的控存而言，内部信息一旦按所设计的微程序被灌注后，在机器运行过程中，只需具有读出的性能即可，故可采用 ROM。此外，在微程序的执行过程中，关键问题是如何由微指令的操作控制字段形成微操作命令，以及如何形成下一条微指令的地址。这是微程序设计必须解决的问题，它们与微指令的编码方式和微地址的形成方式有关。

### 10.2.3　微指令的编码方式

微指令的编码方式又称微指令的控制方式，它是指如何对微指令的控制字段进行编码，以形成控制信号，主要有以下几种。

1. 直接编码（直接控制）方式

在微指令的操作控制字段中，每一位代表一个微操作命令，这种编码方式即为直接编码方式。上面所述的用控制字段中的某位为“1”表示控制信号有效（如打开某个控制门），以及某位为“0”表示控制信号无效（如不打开某个控制门）就是直接控制方式，如图 10.7 所示。这种方式含义清晰，而且只要微指令从控存读出，即刻可由控制字段发出命令，速度快。但由于机器中微操作命令甚多，可能使微指令操作控制字段达几百位，造成控存容量极大。

2. 字段直接编码方式

这种方式就是将微指令的操作控制字段分成若干段，将一组互斥的微操作命令放在一个字段内，通过对这个字段译码，便可对应每一个微命令，如图 10.8 所示。这种方式因靠字段直接译码发出微命令，故又有显式编码之称。

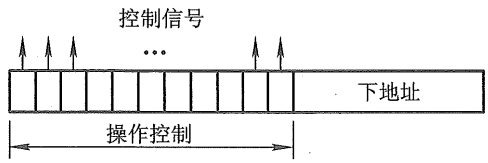



图 10.7　直接编码方式

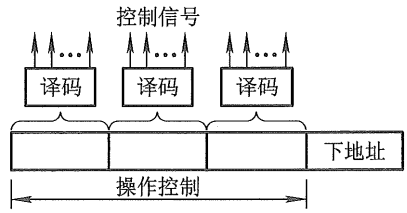



图 10.8　字段直接编码方式

采用字段直接编码方法可用较少的二进制信息表示较多的微操作命令信号。例如，3 位二进制代码译码后可表示 7 个互斥的微命令，留出一种状态表示不发微命令，与直接编码用 7 位表示 7 个微命令相比，减少了 4 位，缩短了微指令的长度。但由于增加了译码电路，使微程序的执行速度稍微减慢。

至于操作控制字段应分几段，与需要并行发出的微命令个数有关，若需要并行发出 8 个微命令，就可分 8 段。每段的长度可以不等，与具体要求互斥的微命令个数有关，若某类操作要求互斥的微命令仅有 6 个，则字段只需安排 3 位即可。

3. 字段间接编码方式

这种方式一个字段的某些微命令还需由另一个字段中的某些微命令来解释，如图 10.9 所示。图中字段 1 译码的某些输出受字段 2 译码输出的控制，由于不是靠字段直接译码发出微命令，故称为字段间接编码，又称隐式编码。

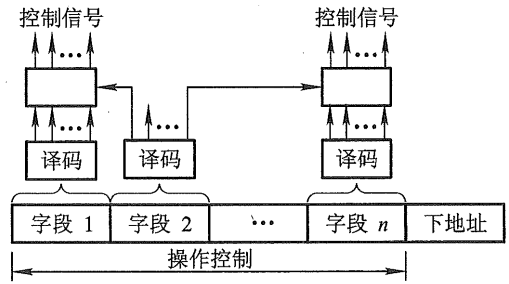



图 10.9　字段间接编码方式

这种方法虽然可以进一步缩短微指令字长，但因削弱了微指令的并行控制能力，因此通常用作字段直接编码法的一种辅助手段。

4. 混合编码

这种方法是把直接编码和字段编码（直接或间接）混合使用，以便能综合考虑微指令的字长、灵活性和执行微程序的速度等方面的要求。

5. 其他

微指令中还可设置常数字段，用来提供常数、计数器初值等。常数字段还可以和某些解释位配合，如解释位为 0，表示该字段提供常数；解释位为 1，表示该字段提供某种命令，使微指令更灵活。

此外，微指令还可用类似机器指令操作码的方式编码，有关内容参见 10.2.5 节微指令格式。

**例 10.4**　某机的微指令格式中，共有 8 个控制字段，每个字段可分别激活 5、8、3、16、1、7、25、4 种控制信号。分别采用直接编码和字段直接编码方式设计微指令的操作控制字段，并说明两种方式的操作控制字段各取几位。

**解**：（1）采用直接编码方式，微指令的操作控制字段的总位数等于控制信号数，即

$$5+8+3+16+1+7+25+4=69$$

（2）采用字段直接编码方式，需要的控制位少。根据题目给出的 10 个控制字段及各段可激活的控制信号数，再加上每个控制字段至少要留一个码字表示不激活任何一条控制线，即微指令的 8 个控制字段分别需给出 6、9、4、17、2、8、26、5 种状态，对应 3、4、2、5、1、3、5、3 位，故微指令的操作控制字段的总位数为

$$3+4+2+5+1+3+5+3=26$$

## 10.2.4　微指令序列地址的形成

由图 10.5 可见，后续微指令的地址大致由两种方式形成。

1. 直接由微指令的下地址字段指出

图 10.4 中大部分微指令的下地址字段直接指出了后续微指令的地址。这种方式又称为断定方式。

2. 根据机器指令的操作码形成

当机器指令取至指令寄存器后，微指令的地址由操作码经微地址形成部件形成。微地址形成部件实际是一个编码器，其输入为指令操作码，输出就是对应该机器指令微程序的首地址。它可采用 PROM 实现，以指令的操作码作为 PROM 的地址，而相应的存储单元内容就是对应该指令微程序的首地址。

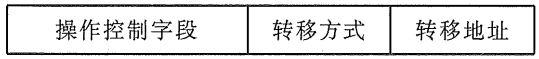
实际上微指令序列地址的形成方式还有以下几种。

3. 增量计数器法

仔细分析发现，在很多情况下，后续微指令的地址是连续的，因此对于顺序地址，微指令可采用增量计数法，即(CMAR)+1→CMAR 来形成后续微指令的地址。

4. 分支转移

当遇到条件转移指令时，微指令出现了分支，必须根据各种标志来决定下一条微指令的地址。微指令的格式如下：




其中，转移方式指明判别条件，转移地址指明转移成功后的去向，若不成功则顺序执行。也有的转移微指令中设两个转移地址，条件满足时选择其中一个转移地址；条件不满足时选择另一个转移地址。

5. 通过测试网络形成

微指令的地址还可通过测试网络形成，如图 10.10 所示。图中微指令的地址分两部分，高段 h 为非测试地址，由微指令的 H 段地址码直接形成；低段 l 为测试地址，由微指令的 L 段地址码通过测试网络形成。

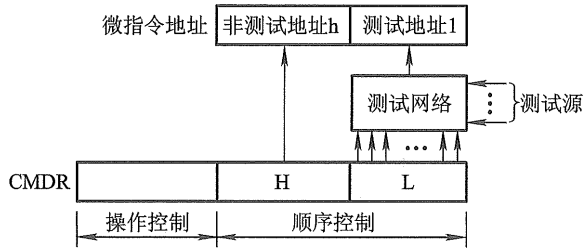



图 10.10　通过测试网络形成微指令地址

6. 由硬件产生微程序入口地址

当电源加电后，第一条微指令的地址可由专门的硬件电路产生，也可由外部直接向 CMAR 输入微指令的地址，这个地址即为取指周期微程序的入口地址。

当有中断请求时，若条件满足，CPU 响应中断进入中断周期，此时需中断现行程序，转至对

应中断周期的微程序。由于设计控制单元时已安排好中断周期微程序的入口地址(参见图10.4),故响应中断时,可由硬件产生中断周期微程序的入口地址。

同理,当出现间接寻址时,也可由硬件产生间址周期微程序的入口地址。

综合上述各种方法,可得出形成后续微指令地址的原理图,如图10.11所示。图中多路选择器可选择以下4路地址。

① (CMAR)+1→CMAR。

② 微指令的下地址字段。

③ 指令寄存器(通过微地址形成部件)。

④ 微程序入口地址。

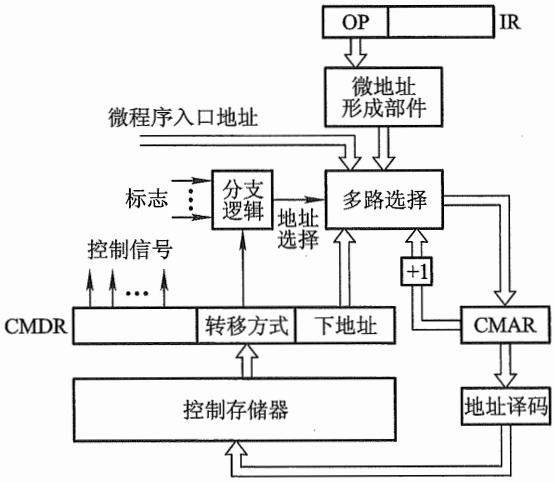



图10.11　后续微指令地址形成方式的原理图

## 10.2.5　微指令格式

微指令格式与微指令的编码方式有关,通常分为水平型微指令和垂直型微指令两种。

1. 水平型微指令

水平型微指令的特点是一次能定义并执行多个并行操作的微命令。图10.7就是典型的水平型微指令。从编码方式看,直接编码、字段直接编码、字段间接编码以及直接和字段混合编码都属于水平型微指令。其中,直接编码速度最快,字段编码要经过译码,故速度受影响。

2. 垂直型微指令

垂直型微指令的特点是采用类似机器指令操作码的方式,在微指令字中,设置微操作码字段,由微操作码规定微指令的功能。通常一条微指令有1~2个微命令,控制1~2种操作。这种

微指令不强调其并行控制功能。

表 10.2 列出了一种垂直型微指令的格式，其中微操作码 3 位，共分六类操作；地址码字段共 10 位，对不同的操作有不同的含义；其他字段 3 位，可协助本条微指令完成其他控制功能。

**表 10.2　垂直型微指令示例**

| 微操作码 | 地址码 | | 其他 | | 微指令类别及功能 |
|---|---|---|---|---|---|
| 0 1 2 | 3~7 | 8~12 | 13　15 | | |
| 0 0 0 | 源寄存器 | 目的寄存器 | 其他控制 | | 传送型微指令 |
| 0 0 1 | ALU<br>左输入 | ALU<br>右输入 | ALU | | 运算控制型微指令<br>按 ALU 字段所规定的功能执行，其结果送暂存器 |
| 0 1 0 | 寄存器 | 移位次数 | 移位方式 | | 移位控制型微指令<br>按移位方式对寄存器中的数据移位 |
| 0 1 1 | 寄存器 | 存储器 | 读写 | 其他 | 访存微指令<br>完成存储器和寄存器之间的传送 |
| 1 0 0 | D | | | S | 无条件转移微指令<br>D 为微指令的目的地址 |
| 1 0 1 | D | | 测试条件 | | 条件转移微指令<br>最低 4 位为测试条件 |
| 1 1 0<br>1 1 1 | | | | | 可定义 I/O 或其他操作<br>第 3~15 位可根据需要定义各种微命令 |

3. 两种微指令格式的比较

① 水平型微指令比垂直型微指令并行操作能力强、效率高、灵活性强。

② 水平型微指令执行一条机器指令所需的微指令数目少，因此速度比垂直型微指令的速度快。

③ 水平型微指令用较短的微程序结构换取较长的微指令结构，垂直型微指令正相反，它以较长的微程序结构换取较短的微指令结构。

④ 水平型微指令与机器指令差别较大，垂直型微指令与机器指令相似。

**例 10.5**　某微程序控制器中，采用水平型直接控制（编码）方式的微指令格式，后续微指令地址由微指令的下地址字段给出。已知机器共有 28 个微命令、6 个互斥的可判定的外部条件，控制存储器的容量为 512×40 位。试设计其微指令格式，并说明理由。

**解**：水平型微指令由操作控制字段、判别测试字段和下地址字段三部分构成。因为微指令采用直接控制（编码）方式，所以其操作控制字段的位数等于微命令数，为 28 位。又由于后续微指令地址由下地址字段给出，故其下地址字段的位数可根据控制存储器的容量（512×40 位）定为 9

位。当微程序出现分支时，后续微指令地址的形成取决于状态条件，6个互斥的可判定外部条件，可以编码成3位状态位。非分支时的后续微指令地址由微指令的下地址字段直接给出。微指令的格式如图10.12所示。

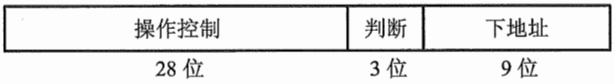



图10.12　例10.5微指令格式

**例10.6**　某机共有52个微操作控制信号，构成5个相斥类的微命令组，各组分别包含5、8、2、15、22个微命令。已知可判定的外部条件有两个，微指令字长28位。

(1) 按水平型微指令格式设计微指令，要求微指令的下地址字段直接给出后续微指令地址。

(2) 指出控制存储器的容量。

**解**：(1) 根据5个相斥类的微命令组，各组分别包含5、8、2、15、22个微命令，考虑到每组必须增加一种不发命令的情况，条件测试字段应包含一种不转移的情况，则5个控制字段分别需给出6、9、3、16、23种状态，对应3、4、2、4、5位(共18位)，条件测试字段取2位。根据微指令字长为28位，则下地址字段取28-18-2=8位，其微指令格式如图10.13所示。

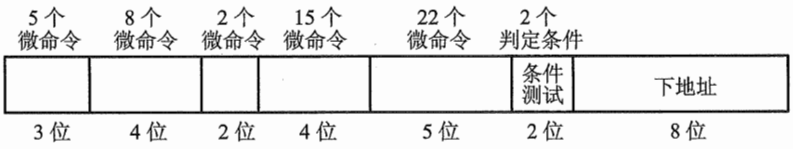



图10.13　例10.6微指令格式

(2) 根据下地址字段为8位，微指令字长为28位，得控制存储器的容量为256×28位。

## 10.2.6　静态微程序设计和动态微程序设计

通常指令系统是固定的，对应每一条机器指令的微程序是计算机设计者事先编好的，因此一般微程序无须改变，这种微程序设计技术即称为静态微程序设计，其控制存储器采用ROM。前面讲述的内容基本上属于这一类。

如果采用EPROM作为控制存储器，人们可以通过改变微指令和微程序来改变机器的指令系统，这种微程序设计技术称为动态微程序设计。动态微程序设计由于可以根据需要改变微指令和微程序，因此可以在一台机器上实现不同类型的指令系统，有利于仿真。但是这种设计对用户的要求很高，目前难以推广。

## 10.2.7　毫微程序设计

微程序可看作是解释机器指令的，毫微程序可看作是解释微程序的，而组成毫微程序的毫微指令则是用来解释微指令的。采用毫微程序设计计算机的优点是用少量的控制存储器空间来达到高度的并行。

毫微程序设计采用两级微程序的设计方法。第一级微程序为垂直型微指令，并行功能不强，但有严格的顺序结构，由它确定后续微指令的地址，当需要时可调用第二级。第二级微程序为水平型微指令，具有很强的并行操作能力，但不包含后续微指令的地址。第二级微程序执行完毕后又返回到第一级微程序。两级微程序分别放在两级控制存储器内。图 10.14 示意了毫微程序控制存储器的基本组成。

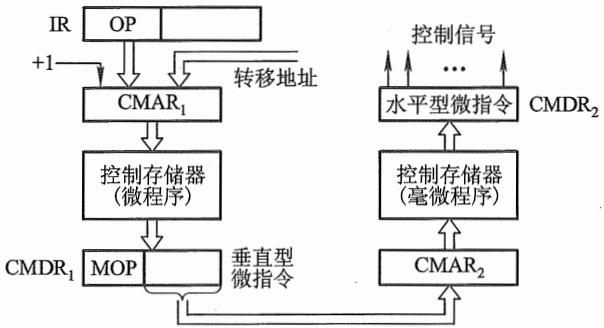



图 10.14　毫微程序控制存储器的基本组成

图中 $CMAR_1$ 为第一级控存地址寄存器，$CMDR_1$ 存放从第一级控制存储器中读出的微指令，如果该微指令只产生一些简单的控制信号，则可以通过译码，直接形成微操作命令，不必调用第二级。如果需调用第二级控制存储器时，则将毫微程序的地址送至 $CMAR_2$，然后由从第二级控制存储器中读出的微指令去直接控制硬件。值得注意的是，垂直型微指令不是和水平型微指令一条一条地对应，而是由水平型微指令（称为毫微指令）组成的毫微程序去执行垂直型微指令的操作。毫微指令与微指令的关系就好比微指令与机器指令的关系一样。

二级控制存储器虽然能减少控制存储器的容量，但因有时一条微指令要访问两次控制存储器，影响了速度。

## 10.2.8　串行微程序控制和并行微程序控制

与机器指令一样，完成一条微指令也分两个阶段：取微指令和执行微指令。如果这两个阶段按图 10.15(a)所示的方式运行，则为串行微程序控制。由于取微指令和执行微指令的操作是在

两个完全不同的部件中完成的，因此可将这两部分操作并行进行，以缩短微指令周期，这就是并行微程序控制，如图 10.15(b)所示，与指令二级流水相似。

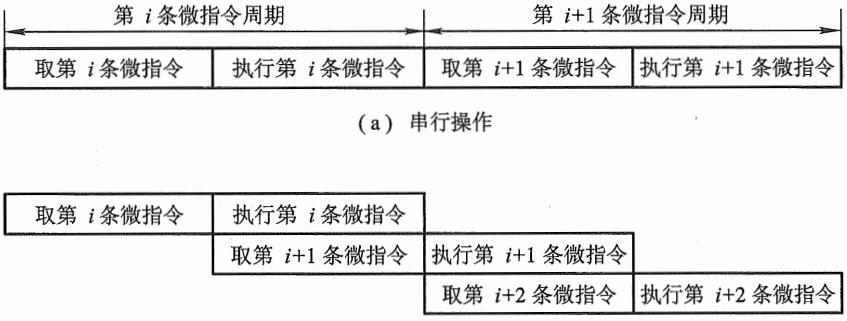



图 10.15　串行微程序和并行微程序控制方式

当采用并行微程序控制时，为了不影响本条微指令的正确执行，需增加一个微指令寄存器来暂存下一条微指令。由于执行本条微指令与取下一条微指令是同时进行的，因此当遇到需要根据本条微指令的处理结果来决定下条微指令的地址时，就不能并行操作，此时可延迟一个微指令周期再取微指令。

## 10.2.9　微程序设计举例

微程序设计控制单元的主要任务是编写对应各条机器指令的微程序，具体步骤是首先写出对应机器指令的全部微操作及节拍安排，然后确定微指令格式，最后编写出每条微指令的二进制代码(称为微指令码点)。

1. 写出对应机器指令的微操作及节拍安排

为了便于与组合逻辑设计比较，仍以 10 条机器指令为例，而且 CPU 结构同组合逻辑设计假设相同。此外，为了简化起见，不考虑间接寻址和中断的情况。下面分别按取指阶段和执行阶段列出其微操作序列。

(1) 取指阶段的微操作及节拍安排

取指阶段的微操作基本与组合逻辑控制相同，不同的是指令取至 IR 后，微程序控制需由操作码形成执行阶段微程序的入口地址，即

$T_0$　PC→MAR，1→R

$T_1$　M(MAR)→MDR，(PC)+1→PC

$T_2$　MDR→IR，OP(IR)→微地址形成部件(编码器)

如果把一个 $T$ 内的微操作安排在一条微指令中完成，上述微操作对应 3 条微指令。