微程序控制器

微程序控制器是一种控制器，同组合逻辑控制器相比较，具有规整性、灵活性、可维护性等一系列优点，因而在计算机设计中逐渐取代了早期采用的组合逻辑控制器，并已被广泛地应用。在计算机系统中，微程序设计技术是利用软件方法来设计硬件的一门技术 。

采用微程序控制方式的控制器称为微程序控制器。所谓微程序控制方式是指微命令不是由组合逻辑电路产生的，而是由微指令译码产生。一条机器指令往往分成几步执行，将每一步操作所需的若干位命令以代码形式编写在一条微指令中，若干条微指令组成一段微程序，对应一条机器指令。在设计CPU时，根据指令系统的需要，事先编制好各段微程序 ，且将它们存入一个专用存储器（称为控制存储器）中。微程序控制器由指令寄存器IR、程序计数器PC、程序状态字寄存器PSW、时序系统、控制存储器CM、微指令寄存器以及微地址形成电路、微地址寄存器等部件组成。执行指令时，从控制存储器中找到相应的微程序段，逐次取出微指令，送入微指令寄存器，译码后产生所需微命令，控制各步操作完成。

1. 微程序控制的基本概念和原理

（1）图-1所示为微程序控制器的一般组成结构

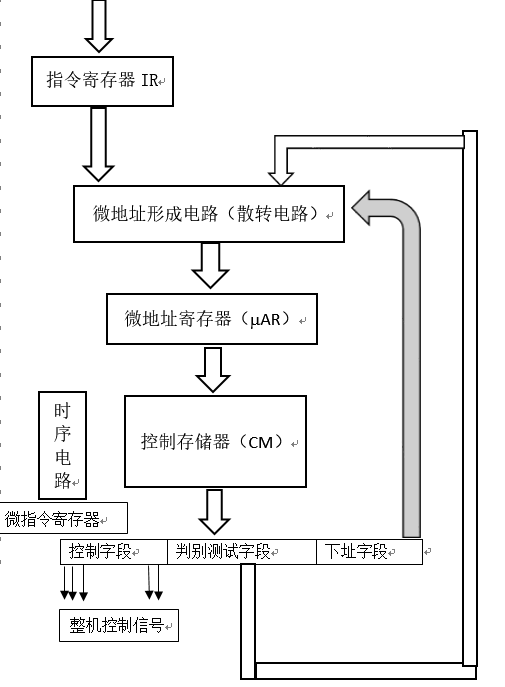


图-1 微程序控制器组成框图

微地址寄存器的输出作为控制存储器的地址，该控制存储器的一个单元中刚好存储一条微指令，所以访问控存一次，其输出就是一条微指令。控存一般是只读的。微指令在控存中的地址叫做微地址。每条微指令包含一组微命令，一个微命令就完成一个微操作，微命令是组成微指令的最小单位，也就是微操作的控制信号，一般都是数据通路上控制门的电位，触发器或寄存器的打入、置位、复位脉冲等。例如，指令在取指令过程中的基本操作也就是微操作有：PC -> AR, PC + 1, RAM -> IR。

微周期：执行一条微指令所需要的时间叫做微周期，一般可以作为一个机器周期。

微程序：微指令的有序集合，微程序是实现一条机器指令功能的程序，每条机器指令的功能都用一段相应的微程序来实现。微程序的流程中包括微程序分支、微程序循环、微子程序等。

（2）微程序控制的基本原理

开机后，首先使微地址寄存器置为取指令的第一条微指令地址，从控制存储器中取出第一条微指令，完成PC -> AR, PC + 1操作，然后根据微指令的下址字段分别取出第二条微指令，完成RAM -> IR，即从内存中读出指令送指令寄存器IR，并且发译码信号使指令译码器工作，即形成该指令的执行指令阶段的微程序入口地址，从控存中取出该指令执行时的第一条微指令送到微指令寄存器，发出控制信号（微命令）实现微操作；然后该指令执行时的其余微指令地址是当前微地址加1或由当前微指令的下址字段确定，依次从控存中取出其余微指令，实现该指令所需的所有微操作，即完成了该指令的执行。每一条指令的最后一条微指令执行完后，均会回到取指令的第一条微指令执行，以取下一条指令，如此重复，直至用户要运行的程序指令执行完为止。

1. 微程序控制器的内部电路组成
2. 指令寄存器IR

指令寄存器(IR, Instruction Register)，是临时放置从内存里面取得的程序指令的寄存器，用于存放当前从主存储器读出的正在执行的一条指令。

当执行一条指令时，先把它从内存取到数据寄存器（DR，Data Register）中，然后再传送至指令寄存器。指令划分为操作码和地址码字段，由二进制数字组成。为了执行任何给定的指令，必须对操作码进行测试，以便识别所要求的操作。指令寄存器的相应位送入指令译码器，根据译码结果产生相应的控制信号，从而完成指令规定的运算、传送数据等操作。

取指过程：

1.现行指令地址送至存储器地址寄存器，记作PC→MAR

2.向主存发送读命令，启动主存读操作，记作1→R

3.将MAR（通过地址总线）所指的主存单元中的内容（指令）经数据总线读至MDR内，记作M（MAR）→MDR

4.将MDR的内容送至IR，记作MDR→IR

5.指令的操作码送至CU译码，记作OP（IR）→CU

6.形成下一条指令的地址，记作（PC）+1→PC

为了提高计算机的并行操作能力和运行速度，目前大多数计算机都将指令寄存器扩充为指令队列，或称指令栈，允许预取多条指令。

1. 控制存储器CM

控制存储器用来存放实现全部指令系统的微程序，是一种只读存储器。一旦微程序固化，机器运行时只读不写。在串行方式的微程序控制器中，微指令周期就是只读存储器的工作周期。控制存储器的字长就是微指令字的长度，其存储容量视机器指令系统而定，即取决于微程序的数量。控制存储器的要求是速度快、周期短。

1. 微指令寄存器

微指令寄存器用来存放由控制存储器读出的一条微指令信息。其中微地址寄存器决定将要访问的下一条微指令的地址，而微命令寄存器则保存微指令的操作控制字段和判别测试字段的信息。

1. 地址转移逻辑

一般情况下，微指令由控制存储器读出后直接给出下一条微指令的地址，这个微地址信息就存放在微地址寄存器中。若微程序不出现分支，那么下一条微指令的地址就直接由微地址寄存器给出。当出现分支时，意味着微程序出现条件转移。在此情况下，通过判别测试字段P和执行部件的“状态条件”反馈信息，去修改微地址寄存器的内容，然后再去读出下一条微指令，地址转移逻辑就承担完成修改微地址的任务。

（注意：计算机内的控制信号一般均由若干个周期状态、若干个节拍电位和若干个节拍脉冲这样的三级时序来对操作信号进行定时。）

1. 微程序控制器控制方法

以两条指令集ADD和JMP为例来说明微程序控制器的控制方法。（其中，加法指令“ADD R, #data”源操作数采用立即数寻址，目的操作数采用寄存器寻址；无条件转移指令“JMP addr”采用直接寻址）

系统的结构为：机器字长8位，地址总线和数据总线8位，存储器存储单元宽度8位，指令为单字节或双字节。

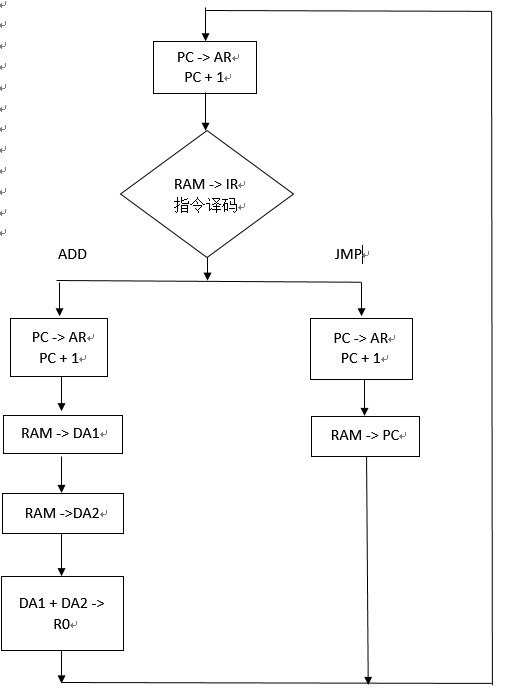
画出CPU的状态图如图-2所示：

图-2 简单CPU状态图

从图-2中可以看出只有8个状态，即可能产生的下址有8个，在取指令周期的第二个状态之后的微地址有两种可能性，微程序控制器采用指令译码的方式来产生下一个微地址，ADD和JMP指令的最后一个状态则采用当前微指令提供下一微地址，其余状态则采用当前微地址+1来提供下一微地址。由此，可以得到简单微程序控制器的组成框图如图-1所示。

根据图-3所示的指令格式，假设ADD指令的操作码I7I6I5I4是0000，JMP指令的操作码I7I6I5I4是0001，对于3位微地址，则微程序的入口地址的逻辑表达式可以写为I410，可以由指令的操作码按逻辑电路映射成该指令的微程序入口地址。由微程序的入口地址的逻辑表达式得到ADD执行指令周期中的第一个状态微地址是010，JMP执行指令周期中的第一个状态的微地址是110. 然后把剩下的微地址分配给图-2中的其他状态，就得到图-4所示的微程序流程图。

I7 I6 I5 I4 I3 I2 I1 I0

|  |  |  |
| --- | --- | --- |
| 操作码 | 源寄存器 | 目的寄存器 |

|  |  |  |
| --- | --- | --- |
| 操作码 | 源寄存器 | 目的寄存器 |

I7 I6 I5 I4 I3 I2 I1 I0

|  |
| --- |
| 立即数 |

图-3 机器指令格式

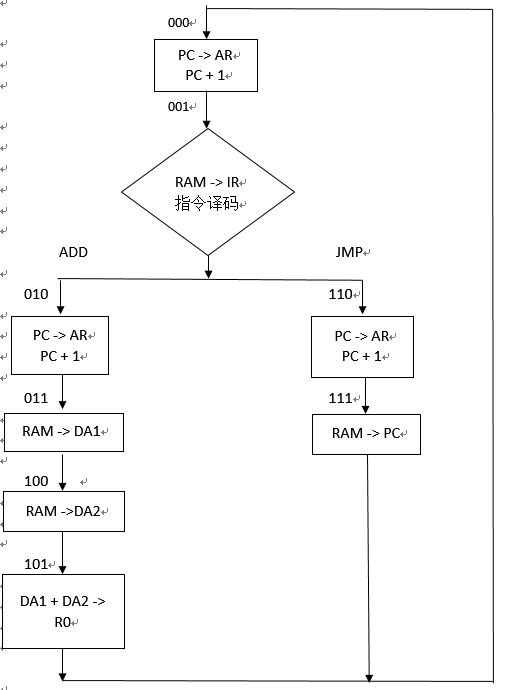


图-4 微程序流程图

将1位判别测试字段中的转移方式命名为，若 = 0 表示下址来源是指令译码， = 1表示下址来源由微指令的下址字段提供，则对于图-4所有状态，只有微地址为001的状态对应的微指令判别测试字段是0，其他状态对应的微指令的判别测试字段都是1，由此得到表-1

表-1 微程序控制器中微指令下址表

|  |  |  |  |
| --- | --- | --- | --- |
| 微地址 | 状态 | 判别测试字段（） | 下址字段 |
| 000 | 取值1：PC->AR, PC+1 | 1 | 001 |
| 001 | 取值2：RAM->IR,译码 | 0 | XXX |
| 010 | ADD1:PC->AR,PC+1 | 1 | 011 |
| 011 | ADD2:RAM->DA1 | 1 | 100 |
| 100 | ADD3:R0->DA2 | 1 | 101 |
| 101 | ADD4:DA1+DA2->R0 | 1 | 000 |
| 110 | JMP1:PC->AR,PC+1 | 1 | 111 |
| 111 | JMP2:RAM->PC | 1 | 000 |

至此，产生了微指令序列。接下来产生正确的微命令，先把系统中每一个微操作列出来，给他们命名，用微指令控制字段中的一位表示一个微命令。对于每条微指令，需要哪些微命令，只需把该微指令中需要的微命令位置为有效，而将不需要的微命令位置为无效，即可产生正确的微命令，下面列出这一例子中涉及的所有微操作：

表-2 控制信号表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 控制信号 | 功能 | 序号 | 控制信号 | 功能 |
| 1 |  | 指令地址送总线 | 10 | S1 | 同上 |
| 2 | B – AR | 总线内容打入地址寄存器 | 11 | S0 | 同上 |
| 3 | PC + 1 | 程序计数器内容加1 | 12 | M | M为‘1’，ALU做逻辑运算  M为‘0’，ALU做算术运算 |
| 4 | B – PC | 总线内容打入程序计数器 | 13 | B – DA1 | 总线内容打入暂存器DA1 |
| 5 | B – IR | 总线内容打入指令寄存器 | 14 | B - DA2 | 总线内容打入暂存器DA2 |
| 6 |  | 存储器写 | 15 |  | 运算器ALU内容送总线 |
| 7 |  | 存储器读 | 16 | Ci | ALU 进位输入 |
| 8 | S3 | 选择ALU 16种运算之一 | 17 | B – DR | 总线内容打入目的寄存器DR |
| 9 | S2 | 同上 | 18 |  | 源寄存器SR内容送总线 |

由表-2可以得出微程序控制器的微指令控制字段有18位，于是就确定了微指令的格式，如图-5

控制字段 判别测试 下址字段

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | ... | 17 | 18 | 19 | 20 | 21 | 22 |

B – AR B – DR

图-5 微程序控制器微指令格式

表-1中微地址为100的微指令ADD3:R0 -> DA2需要R0 – B 信号，而表-2中只有第18条微命令，此时还需要ADD指令码第1字的I3I2 （表示源寄存器SR的编号）2位通过译码来分别产生R0 – B, R1 – B, R2 – B, R3 – B ，前述ADD指令码的第1字中的I3I2为00， 所以经译码产生的是R0 – B 信号。同样，表-1中微地址为101的微指令ADD4: DA1+DA2->R0 需要B – R0 信号，而表-2中只有第17条微命令B – DR, 此时还需要ADD指令码第1字的I1I0（表示目的寄存器DR的编号）2位通过译码来分别产生B – R0, B – R1, B – R2, B – R3 ，前述ADD指令码的第1字中的I1I0为00，所以经译码产生B – R0 信号。

由此得出实现ADD指令和JMP指令的微指令，如图-6 和 图-7所示：

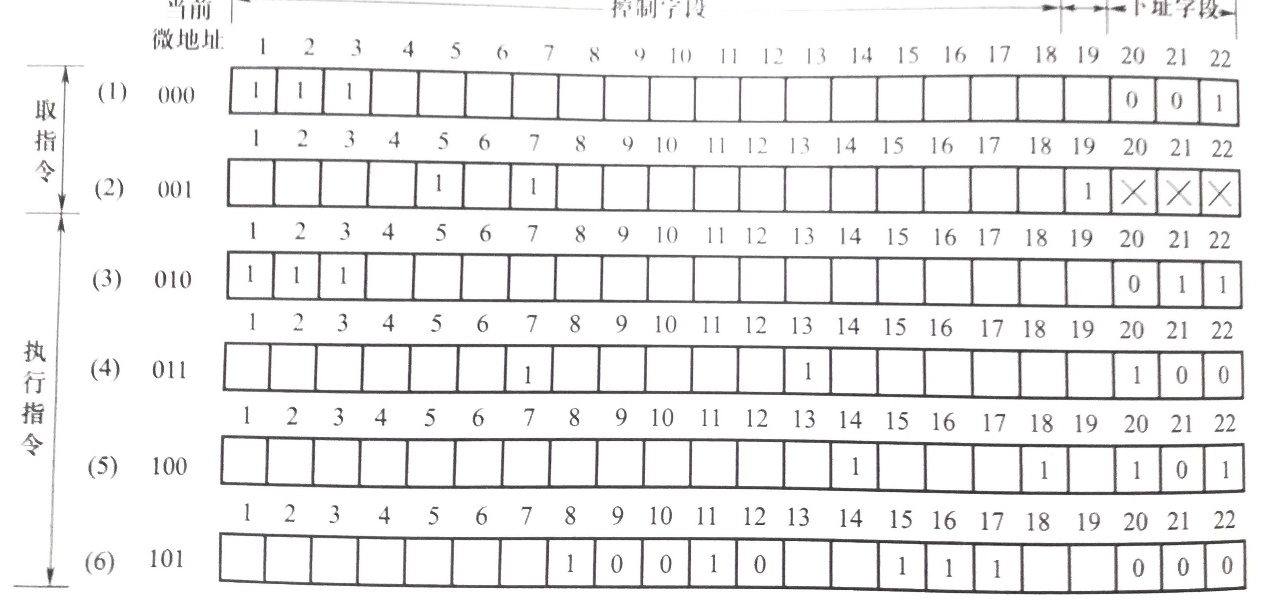


图-6 ADD指令的微指令

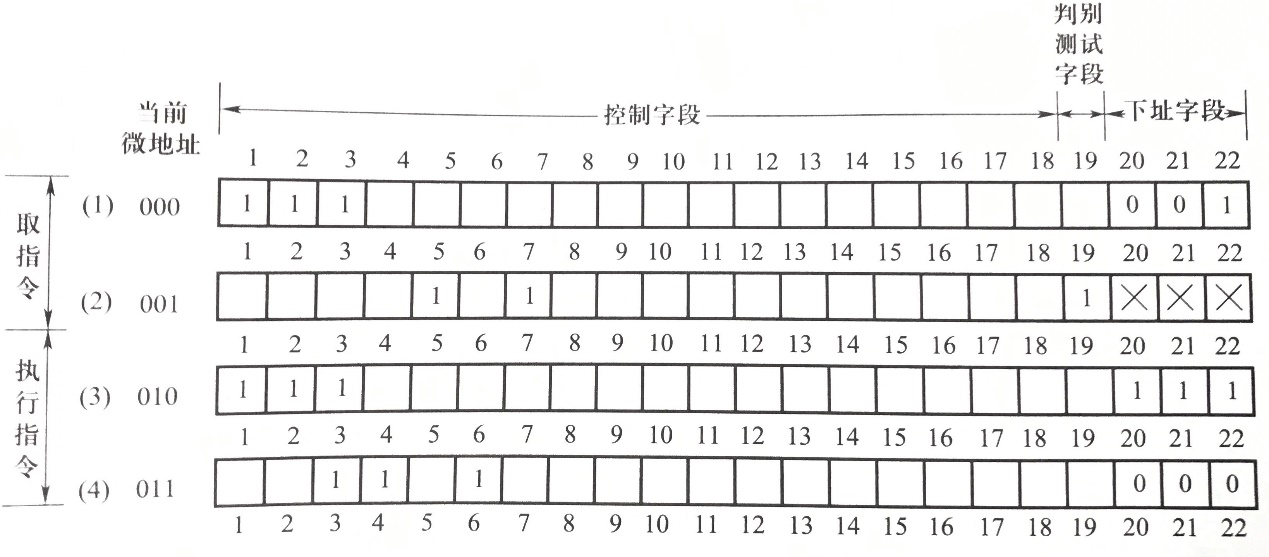


图-7 JMP指令的微指令

按照图-6和图-7的每条微指令都能产生正确的微命令，将这些微指令放在控制存储器中，并根据下一微地址产生的要求依次从控制存储器中取出每一条微指令到微指令寄存器，由微指令寄存器发出每一条微指令的微命令，就可以实现ADD和JMP指令的执行。