[组合逻辑控制单元 2](#_Toc492477545)

[控制信号 2](#_Toc492477546)

[控制单元结构框图 2](#_Toc492477547)

[操作码 3](#_Toc492477548)

[时钟节拍 4](#_Toc492477549)

[状态标志 5](#_Toc492477550)

[输出控制信号 6](#_Toc492477551)

[设计步骤 7](#_Toc492477552)

[微操作命令的操作时间表 7](#_Toc492477553)

[微操作命令的最简逻辑表达式及其逻辑图 8](#_Toc492477554)

[指令执行过程 10](#_Toc492477555)

[微程序控制单元 11](#_Toc492477556)

[控制信号 11](#_Toc492477557)

[控制单元结构框图 11](#_Toc492477558)

[微程序与微指令 12](#_Toc492477559)

[微地址形成方式 15](#_Toc492477560)

[输出控制信号 16](#_Toc492477561)

[设计步骤 17](#_Toc492477562)

[设计微程序的微操作时序表 17](#_Toc492477563)

[确定微指令格式 18](#_Toc492477564)

[编写微指令码点 19](#_Toc492477565)

[指令执行过程 20](#_Toc492477566)

[其他设计理念 21](#_Toc492477567)

[动态微程序设计 21](#_Toc492477568)

[毫微程序的设计 22](#_Toc492477569)

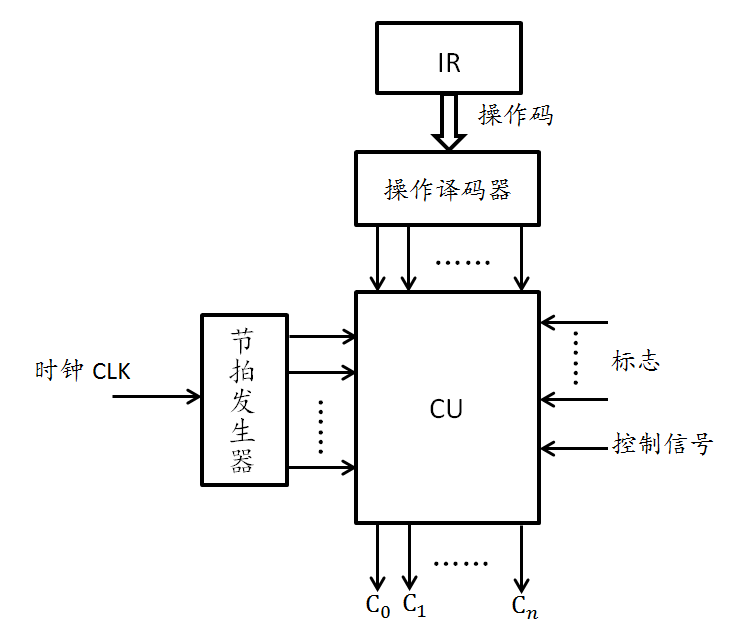
[并行微程序设计 23](#_Toc492477570)

[图片 24](#_Toc492477571)

组合逻辑控制单元

# 控制信号

## 控制单元结构框图



其中只有一个输入是1，其余全为0，

每一个操作码对应一个输出送至CU，其余输出为0

## 操作码

在取指周期的T2节拍执行微操作：OP(IR)🡪CU

IR中的指令的n位操作码送入CU中的操作译码器ID，产生个输出：

100…0、010…0、001…0、……、00…01；

CU的个输入，每一个输入对应一个操作码

CU查看这个输入，只有一个输入是有效的，其余全部无效，这个有效的输入对应一个操作码，CU就知道要执行什么指令

与时钟配合产生不同的控制信号

这是决定控制单元发出不同控制信号的关键

## 时钟节拍

|  |
| --- |
| 时钟脉冲输入到节拍发生器，产生一个与时钟周期等宽的节拍序列  1.完成每个微操做都需要占用一定的时间  2.各个微操作时有先后顺序的，为使控制单元按一定的先后顺序、一定的节奏发出各个控制信号，控制单元必须受时钟控制，即每一个时钟脉冲使控制单元发送一个微操作命令，或发送一组相容性微操作命令  节拍的宽度应该满足数据信息通过数据总线从源到目的 所需要的时间 |
| 在同步时钟控制模式中，每个机器周期含有的时钟周期数是相同的，假设是3  那么节拍发生器就只需要3个输出：T0、T1、T2  输入的时钟信号的每个上升沿改变一次节拍发生器的输出  第1个上升沿，输出：100：T0=1，T1=0，T2=0  第2个上升沿，输出：010：T0=0，T1=1，T2=0  第3个上升沿，输出：001：T0=0，T1=0，T2=1  第4个上升沿，输出：100：T0=1，T1=0，T2=0  …… ……  T0、T1、T2将在逻辑电路中起作用 |

根据指令的要求、当前的时序以及外部和内部的状态情况，按时间的顺序发送一系列微操作控制信号。它由复杂的组合逻辑门电路和一些触发器构成，因此又称为组合逻辑控制器

节拍发生器产生各个机器周期中的节拍信号，是不同的微操做命令按时间的先后发出

个别指令的操作不仅受到操作码的控制，还受到状态标志控制，因此CU的输入来自操作码译码电路ID、节拍发生器以及状态标志，

其输出至CPU内部或外部控制总线上

注意：控制单元还接收来自系统控制总线的控制信号，如中断请求、DMA请求

## 状态标志

|  |  |  |
| --- | --- | --- |
| ALU操作的结果 | 产生控制信号，如BAN指令，控制单元要根据上调指令的结果是否为负而产生不同的控制信号。 |  |
|  |  |  |
| 工作周期标志触发器 | 用于区别4个工作周期  FE：取指周期  IND：间址周期  EX：执行周期  INT：中断周期  1表示有效  分别由：1🡪FE、1🡪IND、1🡪EX、1🡪INT这4个控制信号控制  冯诺依曼计算机根据指令周期的不同阶段来区分从存储器取出的是指令还是数据，更准确地说：  计算机在不同的阶段对存储器进行读操作取出的代码，有不同的用处。  取指周期取出的是指令，执行周期取出的是数据  控制器可以区分存储单元中存放的是指令还是数据，存储器本身无法区分存储单元中存放的是指令还是数据， | |
| 中断请求 |  |  |
| DMA请求 |  |  |
|  |  |  |

## 输出控制信号

用于CPU内寄存器之间的传送和控制ALU实现不同的操作

读碟命令、中断响应

# 设计步骤

## 微操作命令的操作时间表

**列出指令系统中所有微操作的微操作时序表，将这些表结合成一张大表，根据大表来取得各个微操作的逻辑图**

假设某指令系统只含有：LDA、STA、ADD、JMP 、CLA、STP 6个指令

列出它们各自的微操作时序表，将6张微操作时序表结合成一张，即为该控制单元的微操作命令的操作时间表：

将所有的微操作，进行整理：

(1)按照工作周期和节拍分类

(2)去掉同一节拍中重复的微操作

(3)若某指令由表中所列的微操作命令，其对应的空格内为1，否则为空

得到如下表格

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 工作周期标志 | 状态条件 | 节拍 | 微操作命令 | LDA | STA | ADD | CLA | JMP | STP |
| FE |  | T0 | PC🡪MAR | 1 | 1 | 1 | 1 | 1 | 1 |
|  | 1🡪R | 1 | 1 | 1 | 1 | 1 | 1 |
|  | T1 | M(MAR)🡪MDR | 1 | 1 | 1 | 1 | 1 | 1 |
|  | (PC)+1🡪PC | 1 | 1 | 1 | 1 | 1 | 1 |
|  | T2 | MDR🡪IR | 1 | 1 | 1 | 1 | 1 | 1 |
|  | OP(IR)🡪CU | 1 | 1 | 1 | 1 | 1 | 1 |
|  | 1🡪IND | 1 | 1 | 1 |  | 1 |  |
|  | 1🡪EX | 1 | 1 | 1 | 1 | 1 | 1 |
| IND |  | T0 | Ad(IR)🡪MAR | 1 | 1 | 1 |  | 1 |  |
|  | 1🡪R | 1 | 1 | 1 |  | 1 |  |
|  | T1 | M(MAR)🡪MDR | 1 | 1 | 1 |  | 1 |  |
|  | T2 | MDR🡪Ad(IR) | 1 | 1 | 1 |  | 1 |  |
|  | 1🡪EX | 1 | 1 | 1 |  | 1 |  |
| EX |  | T0 | Ad(IR)🡪MAR | 1 | 1 | 1 |  |  |  |
|  | 1🡪R | 1 |  | 1 |  |  |  |
|  | 1🡪W |  | 1 |  |  |  |  |
|  | T1 | M(MAR)🡪MDR | 1 |  | 1 |  |  |  |
|  | ACC🡪MDR |  | 1 |  |  |  |  |
|  | T2 | MDR🡪ACC | 1 |  |  |  |  |  |
|  | MDR🡪M(MAR) |  | 1 |  |  |  |  |
|  | (ACC)+(MDR)🡪ACC |  |  | 1 |  |  |  |
|  | 0🡪ACC |  |  |  | 1 |  |  |
|  | Ad(IR)🡪PC |  |  |  |  | 1 |  |
|  | 0🡪G |  |  |  |  |  | 1 |

## 微操作命令的最简逻辑表达式及其逻辑图

每一个微操作，对应一个控制信号，对应一个逻辑表达式，对应一个逻辑图；

|  |  |
| --- | --- |
| PC🡪MAR |  |
| 1🡪R |  |
| M(MAR)🡪MDR |  |
| (PC)+1🡪PC |  |
| MDR🡪IR |  |
| OP(IR)🡪CU |  |
| 1🡪IND |  |
| 1🡪EX |  |

|  |  |
| --- | --- |
| Ad(IR)🡪MAR |  |
| MDR🡪Ad(IR) |  |
| 1🡪W |  |
| ACC🡪MDR |  |
| MDR🡪ACC |  |
| MDR🡪M(MAR) |  |
| (ACC)+(MDR)🡪ACC |  |
| 0🡪ACC |  |
| Ad(IR)🡪PC |  |
| 0🡪G |  |

## 指令执行过程

继续前面的例子，指令ADD的执行过程如下

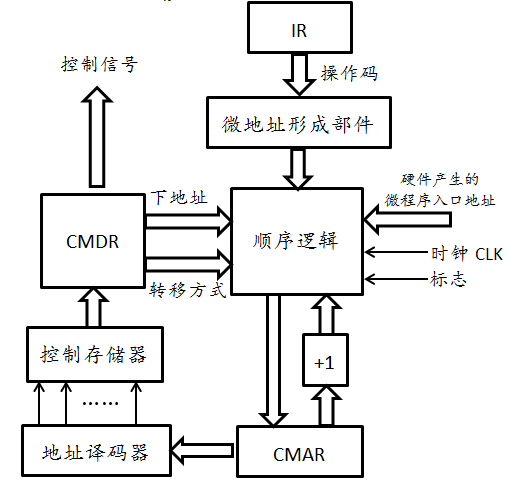
FE=1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 时钟周期 | 时钟节拍 | 满足条件 | 查看逻辑电路  获得有效输出的控制信号 | 说明 |
| 1 | T0 | FE=1  T0=1 | PC🡪MAR |  |
| 1🡪R |  |
| 2 | T1 | FE=1  T1=1 | M(MAR)🡪MDR |  |
| (PC)+1🡪PC |  |
| 3 | T2 | FE=1  T2=1  I=1(举例) | MDR🡪IR |  |
| OP(IR)🡪CU |  |
| 1🡪EX |  |
| 4 | T0 | EX=1  T0=1 | Ad(IR)🡪MAR |  |
| 5 | T1 | EX=1  T1=1 | M(MAR)🡪MDR |  |
| 6 | T2 | EX=1  T2=1  ADD=1 | (ACC)+(MDR)🡪ACC |  |

微程序控制单元

# 控制信号

## 控制单元结构框图



## 微程序与微指令

|  |  |  |
| --- | --- | --- |
| 微程序 | 执行周期  微程序 | 我们已经知道，每条指令都可以看作一个微操作序列  计算机的指令系统是确定的，其包含的指令也是固定不变的；机器有哪些微操作也是确定的  采用微程序设计方法设计控制单元的过程就是：为指令系统中的每一条机器指令的编写一个微程序；  每一个微程序按照，其对应的机器指令，的微操作序列，而进行编写 |
| 取指周期  微程序 | 任何一条机器指令的取指操作都是相同的，因而将取指操作的微操作序列编成一个微程序；  一个指令系统中只需要一个取指微程序，它属于公共微程序  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 |
| 间址周期  微程序 | 如果指令是间接寻址，其操作也是可以预测的，也可以先编出对应间址周期的微程序。  一个指令系统中只需要一个间指微程序，它属于公共微程序 |
| 中断周期  微程序 | 当出现中断时，中断隐指令所需要完成的操作可由一个对应的中断周期的微程序控制完成  一个指令系统中只需要一个取指微程序，它属于公共微程序 |
| 这样，控制存储器中的微程序个数应为机器指令数加上，取指、间址、中断，等公共微程序  指令系统中具有n种机器指令，那么控制存储器中的微程序至少是n+1个，1为公共的取指微程序  完整的地执行一条机器指令，至少需要执行两段微程序：取指周期微程序和执行周期微程序  如果该机器指令是间接寻址的，那么就要执行间址周期微程序，如果是多次间址，那么就得执行多次间址微程序 | |
| 微指令 | 微指令 | 微程序是由微指令构成的，执行一段微程序的过程，其实就是执行一段微指令序列的过程  **一条微指令对应一个时钟周期**  又因为一个时钟周期可能有一个或多个微操作，所以，一条微指令可以对应一个或多个微命令  微指令只是一个二进制代码，控制单元要将微指令进行译码，然后输出其对应控制信号 |
| 微地址 | 控制存储器CM是微程序控制单元的核心部件，用于存储微程序；  CM的每一存储单元存放一个微指令，即每一微指令都对应一个CM微地址  微程序的首地址就是该微程序的第一条微指令的微地址 |
| 编码 | 微指令是一个二进制代码，包含操作控制(高位部分)和顺序控制(低位部分)两个字段  1.操作控制字段：微操作码：用于产生该微指令对应的控制信号  2.顺序控制字段：微地址码：用于产生下一条要执行的微指令的地址  微指令的编码方式又称为微指令的控制方式，其实是微指令控制字段的编码方式，以  编码的目标是在保证速度的情况下，尽量缩短微指令字长，形成对应控制信号 |
| 格式 | 微指令的格式与微指令的编码方式有关  按编码方式的不同，微指令的格式分为：  水平型微指令：直接编码、字段直接编码、字段间接编码、混合编码  垂直型微指令：采用类机器指令操作码的方式进行编码  混合型微指令：在垂直型微指令的基础上增加一些不太复杂的并行操作 |

1.水平型微指令：

|  |  |  |
| --- | --- | --- |
| 直接编码  (直接控制) | 编码 | 操作控制字段中每一位都代表一个微操作命令  n位的操作控制字段，可表示n个微操作命令 |
| 解码 | 查看微指令的操作控制字段，找到值为1的位  发送这些位所对应的控制信号 |
| 特点 | 优点：简单、直观、执行速度快，操作并行性好，  缺点：机器含有较多微操作命令时，控制存储器容量极大 |
| 字段直接编码  (显式编码) | 编码 | 操作控制字段分成若干小字段，每个小字段独立编码，每种编码代表一个微命令  n位的小字段可表示个微操作命令，留出状态00…0 表示不操作  独立编码意味着两个不同小字段的相同的编码表示不同的微操作命令 |
| 分段规则：  因为同一个小字段中的微命令不能同时发出，而不同小字段中的微命令可以同时发出；  相容性的微命令放在不同小字段中，以满足并行执行，即同一个小字段中的微命令是互斥的； |
| 解码 | 每一个小字段配备一个译码电路；  每个译码电路查看各自的输入编码，按照编码输出一个控制信号，  编码00…0 ，表示不输出控制信号  那么n个译码电路，在同一时刻，最多输出n个控制信号 |
| 特点 | 优点：缩短微指令字长  缺点：编码速度慢 |
| 字段间接编码  (隐式编码) | 编码 | 在字段直接编码的基础上  一个字段的某些微命令需由另一个字段中的某些微命令表示，  不是靠字段直接译码发出微命令 |
| 解码 | 每一个小字段配备一个译码电路和一个解释电路  本小字段的译码电路输出至解释电路，同时，其他译码电路也会有输出进入解释电路 |
| 特点 | 优点：进一步缩短微指令字长  缺点：削弱了微指令的并行控制能力，  通常作为直接编码方式的一种辅助手段 |
| 混合编码 | 上述三种编码混合使用 | |
| 特点 | 一条水平型微指令定义并执行集中并行的基本操作  优点：微程序端，执行速度快  缺点：微指令长，编写微程序较麻烦  水平型微指令：操作能力强，效率高、灵活性强  执行一条机器指令所需要的微指令数目少，因此速度比垂直型的速度快  用较短的微程序结构换取较长的微指令结构，  垂直型微指令是正好相反，它以较长的微程序结构换取较短的微指令结构  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 | |

|  |  |
| --- | --- |
| 常数字段 | 设置常数字段，用来提供常数、计数器初值等  常数字段和某些解释位配合，如  解释位是0表示该微指令提供常数，  解释位是1表示该微指令提供某种命令， |

2.垂直型微指令

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 微操作码 | 地址码 | | 其他 | | 微指令 |
| 012 | 3~7 | 8~12 | 13 15 | |
| 000 | 源寄存器 | 目的寄存器 | 其他控制 | | 传送型微指令 |
| 001 | ALU左输入 | ALU右输入 | ALU | | 运算控制型微指令  按ALU字段所规定的功能执行，其结果送暂存器 |
| 010 | 寄存器 | 移位次数 | 移位方式 | | 移位控制型微指令  按移位方式对寄存器中的数据移位 |
| 011 | 寄存器 | 存储器 | 读写 | 其他 | 访存微指令  完成存储器与寄存器之间的传送 |
| 100 | D | | | S | 无条件转移微指令  D为微指令的目的地址 |
| 101 | D |  | 测试条件 | | 条件转移微指令  最低4位为测试条件 |
| 110  111 |  |  |  | | 可定义I/O或其他操作  第3~15位可根据需要定义各种微命令 |

特点是采用类机器指令操作码的方式，在微指令中设置微操作码字段，采用微操做码编译法，由微操作码规定微指令的功能，

基本的指令格式

一条垂直型微指令只能定义并执行一种基本操作

优点：微指令短、简单、规整，便于编写微程序

缺点：微程序长，执行速度慢，工作效率低

3.混合型微指令

微指令较短，仍便于编写，微程序不长，执行速度加快

## 微地址形成方式

微地址送入CM后，CM就会输出相应的微指令至CMDR

显然接下来执行哪条微指令，是由送入CM的微地址码决定的，那么送入CM的微地址是怎么产生的？

一共有以下几种形成方式

**这些方式会将不同的下地址传送给顺序逻辑电路，最终由顺序逻辑电路来决定选用哪个下地址**

|  |  |  |
| --- | --- | --- |
| 硬件产生  微程序入口地址 | 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，这个地址记为取指周期微程序的入口地址 |
| 中断请求 | 若条件满足，CPU相应中断进入中断周期，此时需要中断现行程序，转至对应中断周期的微程序，由于涉及控制单元时，已经安排好中断周期微程序的入口地址，故响应中断，可由硬件产生中断周期微程序的入口地址 |
| 出现间接寻址 | 也可由硬件产生间址周期微程序的入口地址 |
| 微指令  顺序控制字段 | CMDR收到来自CM的微指令，并对微指令进行处理：  处理操作控制字段：产生对应的控制信号，并输出  处理顺序控制字段：获得下一条微指令的微地址，下地址=顺序控制字段，  将下地址送入顺序逻辑电路 | |
| 机器指令  操作码 | 取指阶段结束后(即取指周期微程序执行完毕)，机器指令送入到了IR中，  IR将指令的操作码送入微地址形成部件  微程序控制单元已经为指令系统中的每一条机器指令的编写了一个微程序；  而每一条机器指令对应一个操作码  微地址形成部件的功能就是：根据收到的 操作码 找到其对应的 机器指令 对应的 微程序的首地址  即微程序的第一条微指令的微地址  将该微地址送入顺序逻辑电路 | |
| 增量计数器法 | CMAR接受顺序逻辑电路送来的微地址，即下一条要执行的微指令的微地址  CMAR将该微地址送入地址译码器，再由地址译码器将地址译码，送入CM，以产生下一条要执行的微指令；  **CMAR将**该微地址送入**+1电路**  **+1电路执行：**(CMAR)+1🡪顺序逻辑电路  将该连续微地址(顺序地址)送入顺序逻辑电路 | |
| 分支转移 | CMDR收到来自CM的微指令，并对微指令进行处理：  处理顺序控制字段：转移方式(高位部分)+转移地址(低位部分)  转移方式：指明判别条件，往往是判别某个标志的状态  转移地址：指明判别成功时的转移地址  CMDR将转移方式和转移地址字段都送入顺序逻辑电路，  同时又有标志送入顺序逻辑电路，由顺序逻辑单路来判断判别是否成功； | |
| 处理顺序控制字段：转移方式(高位部分)+转移地址1(中位部分)+转移地址2(低位部分)  转移方式：指明判别条件，往往是判别某个标志的状态  转移地址1：指明判别成功时的转移地址  转移地址2：指明判别失败时的转移地址  CMDR将转移方式和转移地址字段都送入顺序逻辑电路 | |
| 网络测试 | 顺序控制字段：非测试地址(高位部分)+测试地址(低位部分) | |

## 输出控制信号

# 设计步骤

## 设计微程序的微操作时序表

|  |  |  |
| --- | --- | --- |
| 取指微程序 | 节拍 | 微操作 |
| T0 | PC🡪MAR |
| 1🡪R |
| T1 | Ad(CMDR)🡪CMAR |
| T2 | M(MAR)🡪MDR |
| (PC)+1🡪PC |
| T3 | Ad(CMDR)🡪CMAR |
| T4 | MDR🡪IR |
| T5 | OP(IR)🡪微地址形成部件🡪CMAR |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LDA X | 节拍 | 微操作 | STA X | 节拍 | 微操作 | ADD X | 节拍 | 微操作 |
| T0 | Ad(IR)🡪MAR | T0 | Ad(IR)🡪MAR | T0 | Ad(IR)🡪MAR |
| 1🡪R | 1🡪W | 1🡪R |
| T1 | Ad(CMDR)🡪CMAR | T1 | Ad(CMDR)🡪CMAR | T1 | Ad(CMDR)🡪CMAR |
| T2 | M(MAR)🡪MDR | T2 | ACC🡪MDR | T2 | M(MAR)🡪MDR |
| T3 | Ad(CMDR)🡪CMAR | T3 | Ad(CMDR)🡪CMAR | T3 | Ad(CMDR)🡪CMAR |
| T4 | MDR🡪ACC | T4 | MDR🡪M(MAR) | T4 | (ACC)+(MDR)🡪ACC |
| T5 | Ad(CMDR)🡪CMAR | T5 | Ad(CMDR)🡪CMAR | T5 | Ad(CMDR)🡪CMAR |
| CLA X | 节拍 | 微操作 | COM X | 节拍 | 微操作 |  |  |  |
| T0 | 0🡪ACC | T0 | 🡪ACC |  |  |
| T1 | Ad(CMDR)🡪CMAR | T1 | Ad(CMDR)🡪CMAR |  |  |
| SHR X | 节拍 | 微操作 | CSL X | 节拍 | 微操作 |  |  |  |
| T0 | L(ACC)🡪R(ACC)  🡪 | T0 | R(ACC)🡪L(ACC)  🡪 |  |  |
| T1 | Ad(CMDR)🡪CMAR | T1 | Ad(CMDR)🡪CMAR |  |  |
| BAN X | 节拍 | 微操作 | BAZ X | 节拍 | 微操作 | STP | 节拍 | 微操作 |
| T0 | 🡪PC | T0 | 🡪PC | T0 | 0🡪G |
| T1 | Ad(CMDR)🡪CMAR | T1 | Ad(CMDR)🡪CMAR | T1 | Ad(CMDR)🡪CMAR |

## 确定微指令格式

|  |  |
| --- | --- |
| 微指令编码方式 | 微操作数目不多，可使用直接编码方式 |
| 后续微指令  地址形成方式 | 常用的是使用微指令顺序控制字段和机器指令操作码 |
| 微指令字长 | 微指令由操作控制字段和顺序控制字段两部分组成  根据直接编码方式：  20个微操作：对应20个操作控制字段  38条微指令：对应6位顺序控制字段  所以微指令字长取26位 |

38条微指令中

|  |  |
| --- | --- |
| bit | 微操作命令 |
| 0 | PC🡪MAR |
| 1 | 1🡪R |
| 2 | M(MAR)🡪MDR |
| 3 | (PC)+1🡪PC |
| 4 | MDR🡪IR |
| 5 | Ad(IR)🡪MAR |
| 6 | MDR🡪ACC |
| 7 | 1🡪W |
| 8 | ACC🡪MDR |
| 9 | MDR🡪M(MAR) |
| 10 | (ACC)+(MDR)🡪ACC |
| 11 | 0🡪ACC |
| 12 | 🡪ACC |
| 13 | L(ACC)🡪R(ACC)  🡪 |
| 14 | R(ACC)🡪L(ACC)  🡪 |
| 15 | 🡪PC |
| 16 | 🡪PC |
| 17 | 0🡪G |

## 编写微指令码点

|  |  |  |  |
| --- | --- | --- | --- |
| 微程序 | 微地址 | 微指令 | |
| 操作控制字段 | 顺序控制字段 |
| 取指 | 0000 |  |  |
| 0001 |  |  |
| 0010 |  |  |
| LDA | 0011 |  |  |
| STA | 0100 |  |  |
| ADD | 0101 |  |  |
| CLA | 0110 |  |  |
| COM | 0111 |  |  |
| SHR | 1000 |  |  |
| CSL | 1001 |  |  |
| JMP | 1010 |  |  |
| BAN | 1011 |  |  |
| STP | 1100 |  |  |
|  |  |  |  |

## 指令执行过程

目前，大多数计算机都采用微程序设计技术

控制微命令序列

控制形成下一条微指令，即后续指令的地址，其输入与微地址形成部件(与指令寄存器相连)、位地址的下地址字段以及外来的标志有关

有关微指令序列地址的形成在后面极少

|  |  |
| --- | --- |
| 取微指令 | 在机器开始运行时，自动将取指微程序的入口地址送入CMAR，并从CM中读出相应的微指令送入CMDR  取微指令的入口地址一般为CM的0号单元，当取指微程序执行完毕后，从主存中取出的机器指令就已经存入到指令寄存器中了 |
|  | 由机器指令的操作码字段通过微地址形成部件昌盛该机器指令所对应的微程序的入口地址，并送入CMAR |
|  | 从CM中逐条取出对应的微指令并执行 |
|  | 执行完对应于一条机器指令的衣蛾微程序后又回到取指微程序的入口地址，继续第1步 |
|  | 周而复始 |

LDA X

ADD Y

STA Z

STP

|  |  |
| --- | --- |
|  | 取指周期微程序首地址M🡪CMAR |
| 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 |
| 产生位操作命令 | 第一条微指令的操作控制字段中“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 |
| 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 |
| 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 |
| 产生位操作命令 | 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，是对应主存2000H地址单元中的第一条机器指令主存中读出送至MDR中 |
| 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，直到取出取值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”已经存到指令寄存器IR中 |
|  |  |
| 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P送至CMAR，记错OP(IR)🡪微地址形成部件🡪CMAR |
| 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 |
| 产生微操作指令 | 由微操作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 |
| 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(CMAR)🡪CMAR |
| 取微指令 | 即CM(CMAR)🡪CMDR |
| 产生微操作指令 |  |
| …… |  |

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，并发出微操作指令。

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IR中……

微程序控制单元就是这样，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

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和微地址的形成方式有

# 其他设计理念

## 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写好的

因此一般微程序无需改变，这种微程序设计技术称为静态微程序设计，控制存储器采用ROM

前面讲的都是这一类

如果采用EEPROM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，因此可以早一台机器上实现不用的类型的指令系统，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

## 毫微程序的设计

## 并行微程序设计

图片

IR

操作码

操作译码器

……

CU

标志

……

时钟CLK

节拍发生器

……

控制信号

……

顺序逻辑

测试源

……

非测试地址

CMDR

……

测试网络

顺序控制字段高位地址码

测试地址1

硬件产生的

微程序入口地址

时钟CLK

标志

顺序逻辑

转移方式

CMDR

+1

CMAR

控制存储器

……

控制信号

下地址

IR

操作码

微地址形成部件