[指令格式 2](#_Toc492923583)

[基本格式 2](#_Toc492923584)

[操作码OP 3](#_Toc492923585)

[编码--定长操作码 3](#_Toc492923586)

[编码--扩展操作码 4](#_Toc492923587)

[操作类型 5](#_Toc492923588)

[地址码A 6](#_Toc492923589)

[编码 6](#_Toc492923590)

[操作数类型 7](#_Toc492923591)

[操作数寻址方式 8](#_Toc492923592)

[指令长度(指令字长) 10](#_Toc492923593)

[指令格式设计 11](#_Toc492923594)

[MCS-51 11](#_Toc492923595)

指令格式

# 基本格式

|  |  |
| --- | --- |
| 指令  (机器指令) | 一条指令是一组二进制代码  计算机执行某种操作的命令，是计算机运行的最小功能单位 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 操作码  OP | 地址码1的寻址特征 | 地址码1 | 地址码2的寻址特征 | 地址码2 | …… |

一台计算机的所有指令的集合构成该计算机的指令系统，也称为指令集

指令集位于硬件与软件的交集

|  |
| --- |
| **一个指令**是一个机器代码如：10010011，那么这是一个8bit(1B)的指令  除了单字节的指令，还有双字节的、三字节的指令 |
| **一个指令**代表CPU的一个操作，执行该指令就表示执行其对应的操作  **程序**是若干指令的有序集合，程序运行的过程就是按一定的次序逐条执行指令的过程；  举例说明一个程序如：    这是一个由3个指令构成的程序，一般一个程序的指令会放在相邻的存储单元中  如假设指令1放在程序存储器中地址为11H的存储单元中，那么指令2就在12H，指令3在15H |

# 操作码OP

## 编码--定长操作码

|  |  |
| --- | --- |
| 操作码 | 操作码位于一条指令的高位部分  表示该指令代表的操作和功能，操作码区分指令的类型，任意两种指令的操作码必然不同  一个n位操作码的指令系统，最多含有个指令 |

-

|  |  |
| --- | --- |
| 指令格式 | n位定长操作码：该指令系统最大能包含条指令  简化了计算机的硬件设计，提高指令译码和识别速度  当计算机字长为32位或更长时，这是常规用法 |

## 编码--扩展操作码

|  |  |
| --- | --- |
| 指令种类 | 按照操作码的位数将指令系统中的指令分类：  位操作码指令、位操作码指令、…、位操作码指令  其中，N是指令长度 |
| 指令个数 | 位操作码指令的数目：  ……  位操作码指令的数目： |
| 位操作码指令 | 操作码共有： 种  从中选取种 |
| 位操作码指令 | 操作码共有： 种  从中选取种  操作码的高位不能和位操作码指令的操作码相同 |
| …… | …… |
| 使用可变长度操作码，即全部指令的操作码字段的位数不固定，且分散在指令字的不同位置上，则  优点：在指令字长优先的前提下仍保持比较丰富的指令种类，不同地址数的指令可以具有不同长度的操作码，从而在满足需要的前提下，有效地缩短指令字长  缺点：增加指令译码和分析的难度，控制器的设计复杂化 | |
| 注意：  不允许短码是长码的前缀，即短操作码不能与长操作码的前面部分的代码相同  通常，使用频率较高的指令，分配短操作码  使用频率较低的指令，分配长操作码  这样可以减少指令译码和分析的时间  各指令的操作码一定不能重复  设某指令系统的操作数固定是n位，那么该指令系统最多可以提供条指令 | |

指令寻址

无论是指令还是操作数，都是作为数据保存在存储器中的

寻址方式：就是寻找指令或操作数有效地址的方式，就是确定指令或数据所存储的地址

确定本条指令的数据地址，以及下一条将要直性的指令地址的方法

指令中的地址码并不代表操作数的帧数地址，称为形式地址

用形式地址并结合寻址方式，可以计算出操作数在存储器中的真实地址，称为有效地址

## 操作类型

运算性指令寻址的是操作数

转移性指令寻址的是下次与执行的指令的地址

不同的计算机，操作类型也不同，即该计算机有哪些操作，各种操作对应的操作码是什么，这都是由计算机本身决定的

下面介绍的是几类典型的操作

|  |  |  |  |
| --- | --- | --- | --- |
| 数据传送 |  | | |
| 运算 | 算数运算 | 加、减、乘、除、增1、减1、比较、求补  高档机还支持浮点运算、十进制运算等 | |
| 逻辑运算 | 与、或、非、异或等 | |
| 位运算 | 位测试、位清除、位求反，等 | |
| 移位 | 算数移位 | 对有符号数和无符号数的左移(乘) 和 右移(除以) | |
| 逻辑移位 | 对有符号数和无符号数的左移(乘) 和 右移(除以) | |
| 循环移位 |  | |
| 转移 | 无条件转移 |  | |
| 条件转移 | 条件码 |  |
|  |  |
|  |  |
| 调用 |  | |
| 返回 |  | |
| 陷阱 |  | |
| 输入输出 |  | | |
| 其他 |  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
| 特权指令 | 这是仅用于操作系统或其他系统软件的指令。  为确保系统与数据的安全，这类指令不提供给用户使用 | |

# 地址码A

## 编码

|  |  |
| --- | --- |
| 地址码 | 地址码位于一条指令的低位部分  操作数的地址  地址码可能包含多个地址，即该指令包含多个操作数  设某个操作数地址是n位，那么该操作数的寻址范围就是 |

指令的低位部分是地址码，它用来指示本指令所需要使用的操作数的位置；

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 功能 | 地址码分配 | 说明 |
| 零地址  指令 | 不需要操作数的指令 | 无 | 如空操作指令、停机指令、关中断指令等 |
| 零地址运算 | 无 | 仅用在堆栈计算机中  参与运算的两个操作数从栈顶和次栈顶弹出，送到运算器进行运算，运算结果再压入到堆栈中。 |
| 一地址  指令 | 只有目的操作数  单操作数指令 | 单一：目的操作数、**结果** | 从该单一地址读取目的操作数  操作数的地址同时用于保存本次的运算结果 |
| 隐含约定目的的地址  双操作数指令 | 单一：源操作数 | 指令可隐含约定另一个操作数由ACC(累加器)提供，运算结过也放在ACC中 |
| 二地址  指令 | 常用的  算数和逻辑运算指令 | 低位部分：源操作数  高位部分：目的操作数、**结果** | 这种指令往往需要两个操作数，需分别给出目的操作数和源操作数的地址  目的操作数的地址同时用于保存本次的运算结果 |
| 三地址  指令 |  | 最低位部分：结果  中位部分：源操作数  最高位部分：目的操作数 |  |
| 四地址  指令 |  | 最低位部分：下址  次低位部分：结果  中位部分：源操作数  最高位部分：目的操作数 | 下址就是下一条要执行的指令的地址 |

每一个地址码对应一个寻址方式

如：使用单地址指令执行两个操作数的算数运算时，一个操作数由地址码指出，另一操作数(B)由程序指定：B=ACC，ACC是累

## 操作数类型

## 操作数寻址方式

指令的低位部分是地址码，它用来指示本指令所需要使用的操作数的位置；

地址码表示的仅仅是形式地址：A，地址A对应的存储单元存储数据：(A)

地址码也可以是某个寄存器的地址：，该寄存器存储的数据是：

将形式地址按照寻址方式进行相应的计算，才能得到有效地址EA，地址EA对应的存储单元所存储的数据(EA)就是操作数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 寻址方式 | 形式地址 | 有效地址 | 访存次数 |  | | |
| 隐含寻址 | / | 程序指定 | 0 | 优点 | | 缩短指令长度，执行速度快 |
| 缺点 | | 需要增加存储操作数或隐含地址的硬件 |
| 立即寻址 | 立即数  补码表示 | (EA)=A | 0 | 优点 | | 指令执行阶段不访问主存，执行速度快 |
| 缺点 | | 操作数的位数限制了立即数的范围 |
| 直接寻址 | 存储地址 | EA=A | 1 | 优点 | | 执行速度快：指令执行阶段仅访问一次主存，且不需要计算操作数的地址 |
| 缺点 | | A的位数决定了该指令数的寻址范围，操作数的地址不易修改 |
| 寄存器  直接寻址 | 寄存器地址 | EA= | 0 | 优点 | | 执行速度快：指令执行阶段不访问主存，只访问寄存器，而寄存器数量少，对应的地址码位数少，指令长度短 |
| 支持向量/矩阵运算 |
| 缺点 | | 寄存器价格昂贵，计算机中的寄存器有限 |
| 相对寻址 | 偏移量  可正可负  补码表示 | EA=(PC)+A  (PC)：下一条指令地址 | 1 | 优点 | | 编制浮动程序：操作数的地址不固定，随PC值的变化而变化，可以很方便地将程序装入到主存的任意区域  多道程序设计 |
| 广泛应用于转移命令 |
| 基址寻址 | 偏移量  可正可负  补码表示 | EA=(BR)+A  (BR)：基地址 | 1 | 优点 | | 扩大寻址范围 |
| 多道程序设计：用户不必考虑自己的程序存于主存的哪一空间区域 |
| 编制浮动程序 |
| 变址寻址 | 基地址 | EA=A+(IX)  (IX)：偏移量 | 1 | 优点 | | 扩大寻址范围 |
| 处理数组问题：设定A为数组首地址，不断改变变址寄存器IX的内容，形成数组中任意数据的地址， |
| 编制循环程序 |
| 堆栈寻址 |  |  |  |  | | |
| 间接寻址 |  | EA= | 原方式+  括号数 | 使用上述的寻址方式配合间接寻址：  先使用上述的寻址方式得到的结果EA，但此时(EA)不是操作数，而是作为地址码继续寻址，使得到((EA))的最高位是1，继续使用((EA))寻址，直到得到：  的最高位是0，那么就是有效地址；  最后再用有效地址寻址，得到操作数 | | |
| 优点 | 扩大寻址范围：有效地址EA的位数大于形式地址A的位数  便于编制程序：用间接寻址可以方便地完成子程序返回 | |
| 缺点 | 执行速度慢：指令执行阶段要多次访存 | |

机器按字寻址，程序计数器PC给出下一条指令字的访存地址(指令在内存中的地址)，故取决于存储器的字数；

指令寄存器IR用于接受取得的指令，故取决于指令字长

(PC)=(PC)+n

n=本指令占存储单元个数

BR：基址寄存器，以该寄存器存储的数据作为基地址(不变)

BR面向操作系统，内容由操作系统或管理程序确定；程序执行过程中值不可变

BR既可采用专用寄存器，也可由用户决定哪个通用寄存器作为基址寄存器，包含在寻址方式码中

IX：变址寄存器，以该寄存器存储的数据作为偏移量(可变)

IX面向用户，内容由用户设定；程序执行过程中值可变

IX既可采用专用寄存器，也可由用户决定哪个通用寄存器作为变址寄存器，，包含在寻址方式码中

|  |  |
| --- | --- |
| 顺序寻址方式 | 通过程序计数器加1，自动形成下一条指令的地址 |
| 跳跃寻址方式 | 通过转移类指令实现  下一条指令的地址码不是由程序计数器给出，而是由本条指令给出下条指令的地址  注意：是否跳跃可能受到状态寄存器和操作数的控制，  跳跃到的地址分为：  绝对地址：由标记符直接得到  相对地址：对于当前指令地址的偏移量  跳跃的结果是当前指令修改PC值，所以下一条指令仍然是通过程序计数器PC给出 |

转移指令是需要判断转移书否成功，若成功则PC修改位转移目标的地址，否则下一条指令仍然是PC自增后的地址

# 指令长度(指令字长)

|  |  |
| --- | --- |
| 一条指令所包含的二进制代码的bit数  指令字长取决于：操作码的长度、操作数的地址码的长度、操作数地址的个数  因为主存一般是按字节编址，为了便于编址和取指，指令长度多为字节的整数倍 | |
| 指令长度与机器字长  没有固定的关系 | 指令长度可以等于或大于或小于机器字长  指令长度等于机器字长：单字长指令  指令长度等于半个机器字长：半字长指令  指令长度等于两个机器字长：双字长指令 |
| 指令长度与地址个数  没有固定的关系 | 即使是单地址指令可能由于单地址的寻址方式的不同而导致指令长度的不同 |

|  |  |
| --- | --- |
| 定长指令字结构 | 在一个指令系统中，若所有指令的长度都是相等的  执行速度快，控制简单 |
| 变长指令字结构 | 各种指令长度随指令的功能而异 |

# 指令格式设计

## MCS-51

MCS-51是Intel公司于1980年推出的8位单片机系列

主频是12MHZ

MC5-51的一个机器周期包含12个时钟周期

|  |  |  |
| --- | --- | --- |
| MCS-51指令系统  含有111条指令 | 指令大小 | 单字节指令：49条  双字节指令：45条  三字节指令：17条 |
| 指令执行时间 | 1个机器周期的指令：64条  2个机器周期的指令：45条  4个机器周期的指令：2条 |
| 指令功能 | 数据传送类指令：28条  算术操作类指令：24条  逻辑运算类指令：25条  控制转移类指令：17条  位操作指令：17条 |

|  |  |  |  |
| --- | --- | --- | --- |
| 数据传输 | MOV <目的操作数>,<源操作数> | PUSH U |  |
| MOVX , | POP U |  |
| MOVC A,@A+PC | XCH , |  |
| MOVC A,@A+DPTR | XCHD , |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 算术操作 | ADD A,U |  |  |
| ADDC A,U |  |  |
| INC U |  |  |
| SUBB A,U |  |  |
| DEC U |  |  |
| MUL AB |  |  |
| DIV AB |  |  |
|  |  |  |
| 逻辑运算 | CLR U | XRL , |  |
| CPL U |  |  |
| RL U |  |  |
| RLC U |  |  |
| RR U |  |  |
| RRC U |  |  |
| SWAP U |  |  |
| ANL , |  |  |
| ORL , |  |  |
| 位运算 | CLR C | ANL C,/bit | JC rel; |
| CLR bit | ORL C,bit | JNC rel; |
| CPL C | ORL C,/bit | JB bit,rel; |
| CPL bit | MOV C,bit | JNB bit,rel; |
| SETB C | MOV bit,C | JBC bit,rel; |
| SETB bit | MOV C,06H; (20H).6🡪Cy |  |
| ANL C,bit | MOV P1.0,C; Cy🡪P1.0 |  |
| 控制转移 | AJMP | JNZ | JB V,lable |
| LJMP | DJNZ U,lable | JNB V,lable |
| SJMP | CJNE ,, lable | JBC V,lable |
| JMP | JC lable | ACALL addr11 |
| JZ | JNC lable | LCALL addr16 |
| NOP | RETI | RET |