控制器

指令

指令（机器指令）是计算机执行某种操作的命令。
一条指令由**操作码OP**和**地址码A**构成。
一台计算机所有的指令集合构成该计算机的指令系统（指令集）。

字长：

- 指令字长：一条指令的总长度（可能会变）
  - 指令字长会直接影响取指令的时间
  - 按字长分类有：
    - 定长指令字结构
    - 变长指令字结构
- 机器字长：CPU进行一次整数运算所能处理的二进制数据的位数（通常和ALU直接相关）
- 存储字长：一个存储单元中的二进制代码位数（通常和MDR位数相同）

按操作码长度分类：

- 定长操作码：n位 -> 2^n条指令
  - 控制器的译码电路设计简单，但灵活性低
- 可变长操作码
  - 控制器的译码电路设计复杂，但灵活性高
- 定长指令字解构+可变长操作码 -> 扩展操作码指令格式

按操作类型分类：

- 数据传送
  - LOAD：把存储器的数据放到寄存器
  - STORE：把计算器的数据放到存储器
- 算数逻辑操作
  - 算数：加减乘除、±1、求补、福鼎运算、十进制运算
  - 逻辑：与或非、亦或、未操作、微测试、未清除、位求反
- 转移操作
  - 无条件转移：JMP
  - 条件转移：JZ（结果为0），JO（结果溢出），JC（结果有进位）
  - 调用和返回：CALL，RETURN
  - 陷阱Trap和陷阱指令
- 输入输出操作
  - CPU寄存器与IO端口之间的数据传送（端口即IO接口中低寄存器）

零地址指令

| OP |

- 不要操作数的指令（空操作、停机、关中断等）
- 堆栈计算（基于后缀表达式，两个操作数隐含存放在栈顶和次栈顶，计算结果压回栈顶）

一地址指令

| OP | A1 |

- 只需要单操作数（±1、取反、求补等）
  - 指令含义`OP(A1) -> A1`
  - 完成一次指令需要3次访存：取址 -> 读A1 -> 写A1
- 需要两个操作数，但其中一个操作数隐含在某个寄存器（ACC）
  - 指令含义`(ACC)OP(A1) -> ACC`

注：*A1指某个主存地址（类似指针），(A1)表示A1所指向地址中的内容（类似指针所指位置的内容）*

二地址指令

| OP | A1（目的操作数） | A2（源操作数） |

- 常用于两操作数的算术运算、逻辑运算
  - 指令含义：`(A1)OP(A2) -> A1`
  - 完成一条指令需要访存4次：取址 -> 读A1 -> 读A1 -> 写A1

三地址指令

| OP | A1 | A2 | A3（结果） |

- 常用于需要两个操作数的算数运算、逻辑运算
  - 指令含义：`(A1)OP(A2) -> A3`
  - 完成一条指令需要访存4次：取址 -> 读A1 -> 读A1 -> 写A3

四地址指令

| OP | A1 | A2 | A3（结果） | A4（下址） |

- 指令含义：`(A1)OP(A2) -> A3, A4 = 下一条将要指令指令的地址`
- 完成一条指令需要访存4次：取址 -> 读A1 -> 读A1 -> 写A3

通常取指令后`PC+1`，指向下一条指令。
对于四地址指令，执行后PC直接修改为A4所指地址。
`n位地址码的直接寻址范围 = 2^n`

拓展操作码

定长指令字结构 + 可变长操作码 = 扩展操作码

通常对使用频率高的指令，分配较短操作码（类似哈夫曼编码），尽可能减少指令译码和分析的时间。
设计扩展码时需要注意：

1. 不允许短码是长码的前缀：即操作码不能与长操作码的前面部分的代码相同。
2. 各指令的操作码一定不能重复。

E.G. 设指令字长固定16位，设计一套指令系统满足下表左侧：

-	-	-	-	-	-

-	-	-	-	-	-
有15条三地址指令		0000-1110	A1	A2	A3
有12条二地址指令	1111 XXXX XXXX XXXX	1111	0000-1011	A1	A2
有62条一地址指令	1111 11XX XXXX XXXX	1111	1100-1110,1111	0000-1111,0000-1101	A1
有32条零地址指令	1111 1111 111X XXXX	1111	1110-1111	1110-1111	0000-1111

设地址长度为n，上一层留出m种状态，下一层可扩展出m*2^n中状态

指令寻址

确定下一条欲执行指令的存放地址（始终由程序计数器PC给出）。
分为顺序寻址、跳跃寻址。

顺序寻址

对于定长指令字解构：PC按照指令地址的间隔d，在执行完成指令后(PC)+d -> PC

对于变长指令字结构：根据操作码判断指令的总字节数n，修改PC的值(PC)+n -> PC;
根据指令类型，CPU还要进行多次访存，每次读入一个字。

跳跃寻址

若执行到JMP指令，将会改变程序执行流，执行转移指令，直接修改PC值。

数据寻址

确定本条指令的地址码指明的真实地址。

例如对于指令JMP n，根据实际情况，它可以有以下情形：

- 直接跳转到地址为n的指令
- 跳转到相对起始地址+n的指令
- 跳转到相对于当前PC+n的指令(PC)+n -> PC

所以在原有的指令解构上增加若干比特位，用于表示数据寻址的特征（简写为EA）
| 操作码OP | 寻址特征 | 形式地址A |

二地址指令则有
| 操作码OP | 寻址特征 | 形式地址A1 | 寻址特征 | 形式地址A2

寻址方式	**有效地址**	**访存次数（指令执行期间）**
隐含寻址	程序指定	0
立即寻址	A即是操作数	0

寻址方式	有效地址	访存次数（指令执行期间）
直接寻址	EA=A	1
一次间接寻址	EA=(A)	2
寄存器寻址	EA=Ri	0
寄存器间接一次寻址	EA=(Ri)	1
相对寻址	EA=(PC)+A	1
基址寻址	EA=(BR)+A	1
变址寻址	EA=(IX)+A	1
堆栈寻址	入栈/出栈时EA的确定方式不同	硬堆栈不访存，软堆栈访存一次

直接寻址

指令中的形式地址A就是操作数的真实地址EA，即`EA = A`

简单，且指令执行阶段仅访问一次主存，不用撰文计算操作数的地址。
但A的位数决定了该指令操作数的寻址范围，操作数的地址不易修改（灵活性较差）。

间接寻址

指令的地址字段给出的形式地址不是操作数的真正地址，而是操作数有效地址所在的存储单元地址，也就是操作数地址的地址。
即`EA = (A)`

可扩大寻址范围（有效地址EA的位数大于形式地址A的位数）。
（对于多次寻址）便于编制程序（用间接寻址可以方便地完成子程序的返回）。

但指令在执行阶段需要多次访存（一次间接寻址需要两次访存，多次寻址需要根据存储字的最高位确定访存次数）。

寄存器寻址

在指令自重直接给出操作数所在的寄存器编号，即`EA = Ri`， 其操作数在由Ri所指的寄存器内。

这样只需要在取指令时访存一次，执行指令时只要访问寄存器，而不需要缓存。

指令在执行阶段不访问驻村，只访问寄存器，指令字短且执行速度快，执行向量/矩阵运算。
但寄存器昂贵，计算机的寄存器个数有限。

寄存器间接寻址

寄存器Ri中给出的是操作数所在贮存的单元地址，即`EA = (Ri)`。

这样取指令和执行指令各需访存一次（共2次）。

与一般间接寻址相比速度快。
但指令的执行阶段需要访问主存（因为操作数在主存中）。

隐含寻址

在指令中隐含着操作数的地址（不明显给出操作数的地址）。

有助于缩短指令字长。
但需要增加存储操作数或隐含地址的硬件。

立即寻址

形式地址A就是操作数本身（又称为立即数），一般用补码形式，#表示立即寻址特征。

取指令时访存一次（执行指令不需要访问）。

指令执行阶段不访问驻村，指令执行时间最短。
但A的位数限制了立即数的范围。比如A的位数为n，且立即数采用补码时，可表示的数据范围为`-2^(n-1) ~ 2^(n-1)-1`。

偏移寻址（数据寻址的一种）

根据形式地址作为偏移量来得到实际地址。主要有如下几种：
堆栈选址
基址寻址`EA = (BR) + A`，用于转移指令，BR是基址寄存器
变址寻址`EA = (IX) + A`，用于多道程序，IX是变址寻址器
相对选址`EA = (PC) + A`，用于循环程序、数组，PC是地址计数器

基址寻址

将CPU中的基址寄存器BR的内容加上指令格式中的形式地址A，而形成操作数的有效地址`EA = (BR) + A`

有时会使用通用寄存器作为基址寄存器

基址寄存器面向操作系统，其内容有操作系统或管理程序确定（程序员无法接触到）。

便于程序“浮动”，方便多道程序并发运行。

变址寻址

有效地址EA等与指令字中的形式地址A与变址寄存器IX的内容相加之和`EA = (IX) + A`，IX可以是变址寄存器(专用)，也可以用通用寄存器作为变址寄存器。

变址寄存器面向用户，IX的内容可由用户改变，形式地址A不变。

实际使用中往往会需要多种寻址方式符合使用，例如先基址后变址寻址：`EA = (IX) + ((BR)+A)`

相对寻址

把程序计数器PC的内容加上指令格式中的形式地址A而形成操作数的有效地址，即`EA = (PC)+A`，
A是相对于PC所指地址（正在执行指令的下一条指令的位移量），补码表示，可正可负。

操作数的地址会随着PC的变化而变化，与指令地址间总是相差一个固定值，便于程序浮动（一段代码内部的活动）。
广泛应用于转移指令。

堆栈寻址

操作数存放在对战中，隐含使用堆栈指针SP作为操作数地址。

对战是存储器（或专用寄存器组）中的一块特定的按LIFO的原则管理的存储区，区中悲读/写单元的地址是用一个特定的计算器给出，该寄存器称为堆栈指针SP。

有寄存器中的硬堆栈（成本高），和主存中的软堆栈

对战可用于函数调用时保存当前函数相关信息。