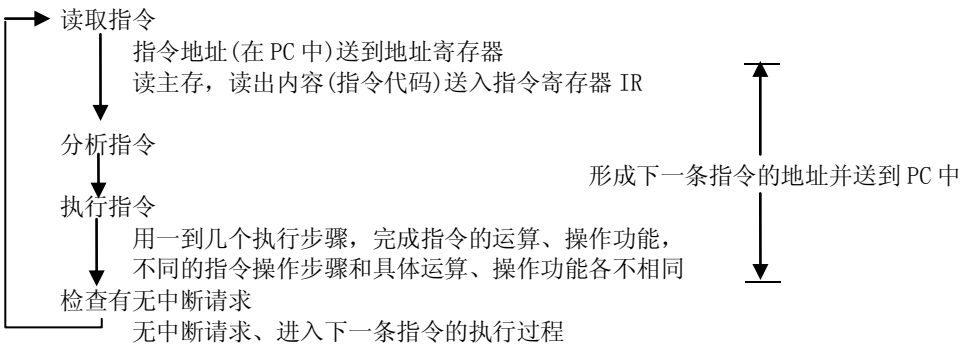
第 7 章　指令系统

人们习惯把每一条机器语言的语句称为机器指令，而又将全部机器指令的集合称为机器的指令系统。

指令的执行过程：



(一) 指令格式

1.指令的基本格式

操作码字段	地址码字段

计算机是通过执行指令来处理各种数据的。为了指出数据的来源，操作结果的去向及所执行的操作，一条指令必须包含下列信息：

(1)操作码，具体说明了操作的性质及功能。

(2)操作数的地址。

(3)操作结果的存储地址。

(4)下一条指令的地址。

从上述分析可知，一条指令实际上包括两种信息即操作码和地址码。

操作码(operation code)用来表示该指令所要完成的操作(如加，减，乘，除，数据传送等)，其长度取决于指令系统中的指令条数。如操作码占 7 位，则该机器最多包含 2^7=128 条指令。

地址码用来描述该指令的操作对象，或直接给出操作数或指出操作数的存储器地址或寄存器地址(即寄存器名)。

操作码的长度不固定会增加指令译码和分析难度，使控制器的设计复杂。

操作码	寻址地址	形式地址 A

形式地址　指令字中的地址

有效地址　操作数的真实地址

约定　指令字长=存储字长=机器字长

2. 定长操作码指令格式

（1）零地址指令

格式：

OP

OP——操作码

指令中只有操作码，而没有操作数或没有操作数地址。这种指令有两种可能：

① 无需任何操作数，如空操作指令，停机指令等。

② 所需的操作数是默认的。如堆栈结构计算机的运算指令，所需的操作数默认在堆栈中，由堆栈指针 SP 隐含指出，操作结果仍然放回堆栈中。又如 Intel 8086 的字符串处理指令，源，目的操作数分别默认在源变址寄存器 SI 和目的变址寄存器 DI 所指定的存储器单元中。

（2）一地址指令

格式：

8	24
OP	A_1

2 次访存

(ACC) OP (A_1) ⟶ACC　　寻址范围 $2^{24} = 16$ M

OP——操作码

A——操作数的存储器地址或寄存器名

指令中只给出一个地址，该地址既是操作数的地址，又是操作结果的存储地址。如加 1，减 1 和移位等单操作数指令均采用这种格式，对这一地址所指定的操作数执行相应的操作后，产生的结果又存回该地址中。

在某些字长较短的微型机中(如早期的 Z80，Intel8080，MC6800 等)，大多数算术逻辑指令也采用这种格式，第一个源操作数由地址码 A 给出，第二个源操作数在一个默认的寄存器中，运算结果仍送回到这个寄存器中，替换了原寄存器内容，通常把这个寄存器称累加器。

（3）二地址指令

格式：

8	12	12
OP	A_1	A_2

或 (A1) OP (A2) ⟶A1　　4 次访存

(A1) OP (A2) ⟶A2　　寻址范围 $2^{12} = 4$ K

若结果存于 ACC　3次访存　若ACC 代替 A_1（或A_2）

OP——操作码

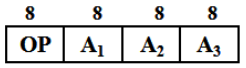
A1——第一个源操作数的存储器地址或寄存器地址。

A2——第二个源操作数和存放操作结果的存储器地址或寄存器地址。

这是最常见的指令格式，两个地址指出两个源操作数地址，其中一个还是存放结果的目的地址。对两个源操作数进行操作码所规定的操作后，将结果存入目的地址，在本例中即为 A2 指定的地址

（4）三地址指令

格式：



4 次访存

寻址范围 $2^8 = 256$

(A_1) OP $(A_2) \longrightarrow A_3$

若 A_3 用 A_1 或 A_2 代替

OP——操作码

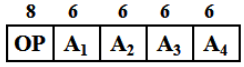
A1——第一个源操作数的存储器地址或寄存器地址

A2——第二个源操作数的存储器地址或寄存器地址

A3——操作结果的存储器地址或寄存器地址

其操作是对 A1，A2 指出的两个源操作数进行操作码(OP)所指定的操作，结果存入 A3 中。

（5）四地址指令



A_1 第一操作数地址

A_2 第二操作数地址

A_3 结果的地址

A_4 下一条指令地址

(A_1) OP $(A_2) \longrightarrow A_3$

设指令字长为 32 位

操作码固定为 8 位

4 次访存

寻址范围 $2^6 = 64$

若 PC 代替 A_4

OP——操作码

A1——第一个源操作数的存储器地址或寄存器地址

A2——第二个源操作数的存储器地址或寄存器地址

A3——操作结果的存储器地址或寄存器地址

A4——下一条指令地址

（6）多地址指令

在某些性能较好的大，中型机甚至高档小型机中，往往设置一些功能很强的，用于处理成批数据的指令，如字符串处理指令，向量，矩阵运算指令等。

为了描述一批数据，指令中需要多个地址来指出数据存放的首地址，长度和下标等信息。

3. 扩展操作码指令格式

设某机器的指令长度为 16 位，包括 4 位基本操作码字段和三个 4 位地址字段，

其格式下：

OP(4)	A_1(4)	A_2(4)	A_3(4)

4 位基本操作码有 16 个码点(即有 16 种组合)，若全部用于表示三地址指令，则只有 16 条。但，若三地址指令仅需 15 条，两地址指令需 15 条，一地址指令需 15 条，零地址指令需 16 条，共 61 条指令，应如何安排操作码？

显然，只有 4 位基本操作码是不够的，必须将操作码的长度向地址码字段扩展才行。

一种可供扩展的方法和步骤如下：

① 15 条三地址指令的操作码由 4 位基本操作码从 0000～1110 给出，剩下一个码点 1111 用于把操作码扩展到 A1，即 4 位扩展到 8 位;

② 15 条二地址指令的操作码由 8 位操作码从 11110000～11111110 给出，剩下一个码点 11111111 用于把操作码扩展到 A2，即从 8 位扩展到 12 位;

③ 15 条一地址指令的操作码由 12 位操作码从 111111110000～111111111110 给出，剩下的一个码点 111111111111 用于把操作码扩展到 A3，即从 12 位扩展到 16 位;

④ 16 条零地址指令的操作码由 16 位操作码从 1111111111110000～1111111111111111 给出。

指令字长取决于**操作码的长度**、**操作数地址的长度**和**操作数地址的个数**。为了提高指令的运行速度和节省存储空间，通常尽可能的吧常用的指令(如数据传输指令、算逻运算指令等)设计成单字长或短字长格式的指令。

指令字长决定于
- **操作码的长度**
- **操作数地址的长度**
- **操作数地址的个数**

1. 指令字长 固定

指令字长＝存储字长

2. 指令字长 可变

按字节的倍数变化

小结

➢ **当用一些硬件资源代替指令字中的地址码字段后**

- **可扩大指令的寻址范围**
- **可缩短指令字长**
- **可减少访存次数**

➢ **当指令的地址字段为寄存器时**

三地址　OP　R_1，R_2，R_3

二地址　OP　R_1，R_2

一地址　OP　R_1

- **可缩短指令字长**
- **指令执行阶段不访存**

操作数类型	
地址	地址实际也可以看做是一种数据，在许多情况下要计算操作数的地址。这时地址可看作无符号的整数
数字	计算机中常见的数字有定点数、浮点数和十进制数字
字符	在应用计算机时，文本或者字符串也是一种常见的数据类型
逻辑数据	计算机除了做算术运算外，有时还做逻辑运算，此时 n 个 0 和 1 的组合不是被看做算术数字而被看做逻辑数
奔腾 Pentium 处理器的数据类型有逻辑数、有符号数(补码)、无符号数、压缩和未压缩的 BCD 码、地址指针、位串以及浮点数(符合 IEEE754 标准)等	

指令操作类型		
1.数据传送	包括寄存器与寄存器，寄存器与存储单元，存储单元与存储单元之间的传送	
2.算术逻辑操作	可实现算术运算(加，减，乘，除，增 1，减 1，取负即求补)逻辑运算(与，或，非，异或)	
3.移位	移位可分为算术移位，逻辑移位和循环移位三种	
4.转移	无条件转移	不受任何约束条件直接把程序转移到下一条需执行指令的地址
	条件转移	根据当前指令的执行结果决定是否需要转移
	调用与返回	● 子程序可在多处被调用 ● 子程序调用可出现在子程序中，即允许子程序嵌套 ● 每个 CALL 指令都对应一条 RETURN 指令
		CPU 必须记住返回地址，使子程序能准确返回，返回地址存放在以下 3 处 ● 寄存器内。机器内设有专用寄存器，专用于存放返回地址 ● 子程序的入口地址内 ● 栈顶内。现代计算机都设有堆栈，执行 RETURN 指令后，便可自动从堆栈内取出应返回的地址
	陷阱(Trap)与陷阱指令	其实是一种意外事故的中断，一般不提供给用户使用，作为隐指令，再出现故障时，由 CPU 自动产生并执行
5.输入输出	对于 I/O 单独编址的计算机而言，通常设有输入输出指令，他完成从外设中的寄存器读入一个数据到 CPU 寄存器内，或将数据从 CPU 的寄存器输出至某外设的寄存器中	
6.其它	包括等待指令、停机指令、空操作指令、开中断指令、关中断指令、置条件码指令等	
备注	有些大型或巨型机还设有向量指令，可对整个向量或矩阵进行求和求积运算	

(二) 指令的寻址方式

1. 有效地址的概念

操作数的真实地址称为有效地址，记做 EA，它是寻址方式和形式地址共同来决定的。

2. 数据寻址和指令寻址

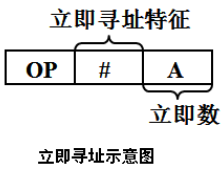
寻址方式是指确定本条指令的数据地址以及下一条将要执行的指令的地址，与硬件结构密切相关，寻址方式分为指令寻址和数据寻址两大类

指令寻址分为顺序寻址和跳跃寻址两种。

顺序寻址可以通过程序计数器 PC 加 1 自动形成下一条指令的地址，跳跃寻址则通过转移类指令实现，是通过对 PC 的运算得到新的下一条指令的地址。

3. 常见寻址方式

（1）立即寻址

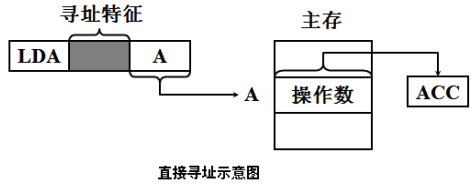


立即寻址示意图

所需的操作数由指令的地址码部分直接给出，就称为立即数(或直接数)寻址方式。这种方式的特点是取指时，操作码和一个操作数同时被取出，不必再次访问存储器，提高了指令的执行速度。但是由于这一操作数是指令的一部分，不能修改，而一般情况下，指令所处理的数据都是在不断变化的(如上条指令的执行结果作为下条指令的操作数)，故这种方式只能适用于操作数固定的情况。通常用于给某一寄存器或存储器单元赋初值或提供一个常数等。(图中“#”表示立即寻址的标记，A 的位数限制了这类指令所能表述的立即数的范围)

（2）直接寻址

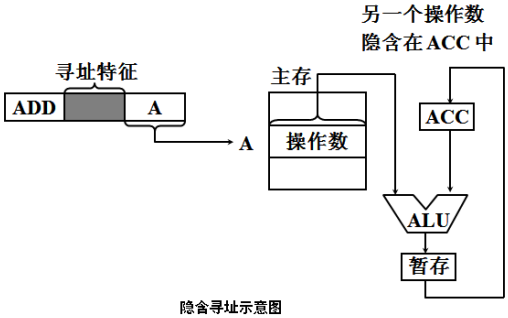
指令的地址码部分给出操作数在存储器中的地址。



直接寻址示意图

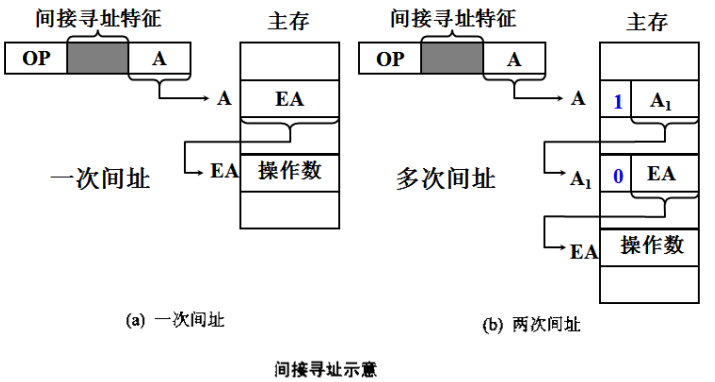
（3）隐含寻址

操作数的地址隐含在操作码或者某个寄存器中。



隐含寻址示意图

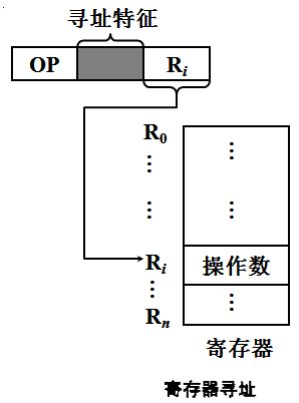
（4）间接寻址



间接寻址示意

在寻址时，有时根据指令的地址码所取出的内容既不是操作数，也不是下一条要执行的指令，而是操作数的地址或指令的地址，这种方式称为间接寻址或间址。

（5）寄存器寻址



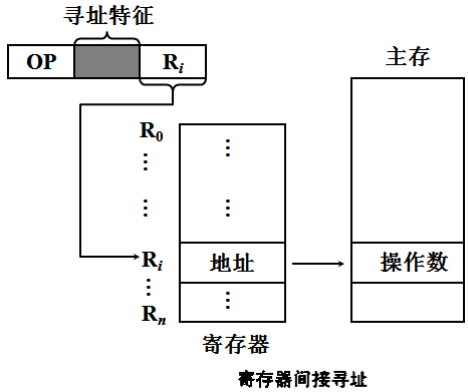
寄存器寻址

计算机的中央处理器一般设置有一定数量的通用寄存器，用以存放操作数，操作数的地址或中间结果。假如指令地址码部分给出某一通用寄存器地址，而且所需的操作数就在这一寄存器中，则称为寄存器寻址。通用寄存器的数量一般在几个至几十个之间，比存储单元少很多，因此地址码短，而且从寄存器中存取数据比从存储器中存取快得多，所以这种方式可以缩短指令长度，节省存储空间，提高指令的执行速度，在计算机中得到广泛应用。

（6）寄存器间接寻址

EA = (Ri) 有效地址在寄存器中

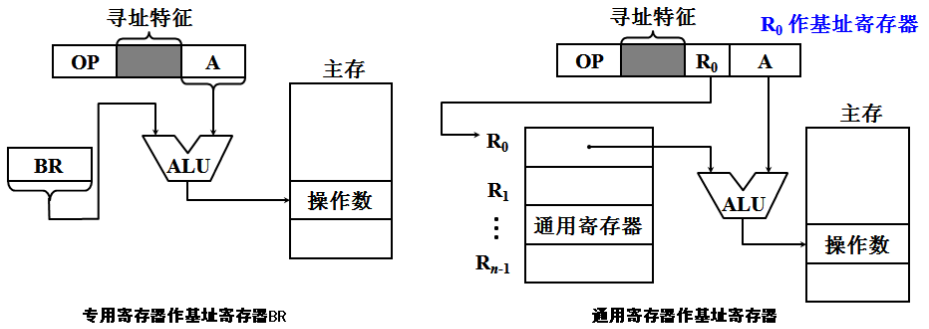
寄存器中给出的是操作数的地址，因此还需要访问一次存储器才能得到操作数。



寄存器间接寻址

（7）基址寻址

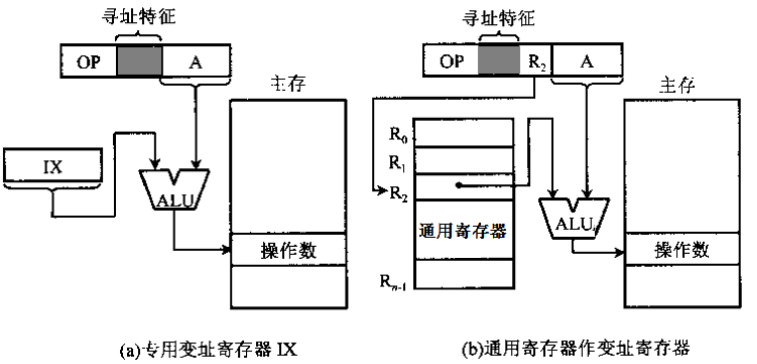
在计算机中设置一个专用的基址寄存器，或由指令指定一个通用寄存器为基址寄存器。操作数的地址由基址寄存器的内容和指令的地址码 A 相加得到



专用寄存器作基址寄存器BR　　通用寄存器作基址寄存器

基址寻址示意图

（8）变址寻址

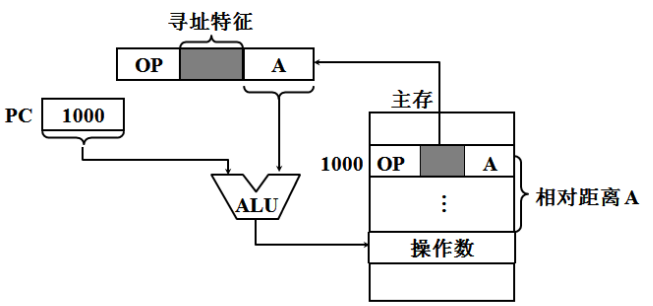


(a)专用变址寄存器 IX　　(b)通用寄存器作变址寄存器

变址寻址示意

指令地址码部分给出的地址 A 和指定的变址寄存器 X 的内容通过加法器相加，所得的和作为地址从存储器中读出所需的操作数。这是几乎所有计算机都采用的一种寻址方式。

（9）相对寻址



相对寻址示意图

把程序计数器 PC 的内容(即当前执行指令的地址)与指令的地址码部分给出的位移量(disp)之和作为操作数的地址或转移地址，称为相对寻址。

主要用于转移指令，执行本条指令后，将转移到(PC)＋disp，(PC)为程序计数器的内容。相对寻址有两个特点：

① 转移地址不是固定的，它随着 PC 值的变化而变化，并且总是与 PC 相差一个固定值 disp，因此无论程序装人存储器的任何地方，均能正确运行，对浮动程序

很适用。

② 位移量可正，可负，通常用补码表示。如果位移量为 n 位，则这种方式的寻址范围在 $(PC)-2^{n-1}$ ～$(PC)+2^{n-1}-1$ 之间。

计算机的程序和数据一般是分开存放的，程序区在程序执行过程中不允许修改。在程序与数据分区存放的情况下，不用相对寻址方式来确定操作数地址。

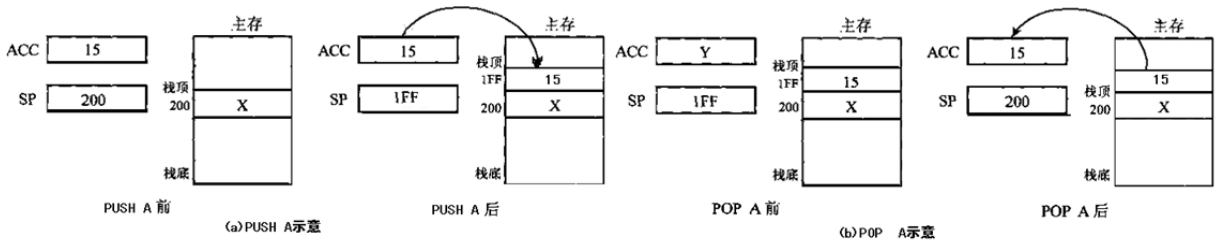
（10）堆栈寻址

在一般计算机中，堆栈主要用来暂存中断和子程序调用时现场数据及返回地址，用于访问堆栈的指令只有压入(即进栈)和弹出(即退栈)两种，它们实际上是一种特殊的数据传送指令：

压入指令(PUSH)是把指定的操作数送入堆栈的栈顶；

弹出指令(POP)的操作刚好相反，是把栈顶的数据取出，送到指令所指定的目的地。

一般的计算机中，堆栈从高地址向低地址扩展，即栈底的地址总是大于或等于栈顶的地址(也有少数计算机刚好相反)当执行压入操作时，首先把堆栈指针(SP)减量(减量的多少取决于压入数据的字节数，若压入一个字节，则减 1;若压入两个字节，则减 2，以此类推)，然后把数据送人 SP 所指定的单元;当执行弹出操作时，首先把 sp 所指定的单元(即栈顶)的数据取出，然后根据数据的大小(即所占的字节数)对 SP 增量。



堆栈寻址示意图

设计指令格式应考虑的各种因素

指令系统集中反映了机器的性能，又是程序员编程的依据，高档机必须能兼容低档机的程序运行，称之为“向上兼容”。

指令格式集中体现了指令系统的功能。为此，在确定指令系统时，必须从以下几个方面综合考虑。

① 操作类型：包括指令数及操作的难易程度

② 数据类型：确定哪些数据类型可以参加操作

③ 指令格式：包括指令字长、操作码位数、地址码位数、地址个数、寻址方式类型、以及指令字长和操作码位数是否可变等。

④ 寻址方式：包括指令和操作数具体有哪些寻址方式。

⑤ 寄存器个数：寄存器的多少直接影响指令的执行时间。

<table>
<tr><td>寻址方式</td><td colspan="3">详情</td></tr>
<tr><td rowspan="2">指令寻址</td><td>顺序寻址</td><td colspan="2">顺序寻址可通过程序计数器 PC 加 1 自动形成下一条指令的地址</td></tr>
<tr><td>跳跃寻址</td><td colspan="2">跳跃寻址则通过转移类指令实现</td></tr>
<tr><td rowspan="13">数据寻址</td><td rowspan="2">1.立即寻址</td><td colspan="2">操作数本身设在指令字内，即形式地址 A 不是操作数地址而是操作数本身</td></tr>
<tr><td colspan="2">● 指令执行阶段不访存
● A 的位数限制了这类指令所能表述的立即数的范围</td></tr>
<tr><td rowspan="2">2.直接寻址</td><td colspan="2">指令中的形式地址 A 就是操作数的真实地址 EA，即 EA=A</td></tr>
<tr><td colspan="2">● 执行阶段访问一次存储器
● 缺点在于 A 的位数限制了操作数的寻址范围而且必须修改 A 的值才能修改操作数的地址</td></tr>
<tr><td rowspan="2">3.隐含寻址</td><td colspan="2">指令字中不明显给出操作数的地址，其操作数的地址隐含在操作码或某个寄存器中</td></tr>
<tr><td colspan="2">● 由于隐含寻址在指令字中少了一个地址，因此，这种寻址方式的指令有利于缩短指令字长</td></tr>
<tr><td rowspan="3">4.间接寻址</td><td colspan="2">倘若指令字中的形式地址不直接指出操作数的地址，而是指出操作数有效地址所在的存储单元的地址，也就是说，有效地址是由形式地址间接提供的，即为间接地址，即 EA=(A)</td></tr>
<tr><td>优点</td><td>1. 与直接寻址相比，扩大了操作数的寻址范围，因为 A 的位数通常小于指令字长，而存储字长可与指令字长相等
2. 它便于编制程序</td></tr>
<tr><td>缺点</td><td>● 指令的执行阶段需要访存两次(一次间接寻址)或多次(多次间接寻址)，致使指令执行时间延长</td></tr>
<tr><td rowspan="2">5.寄存器寻址</td><td colspan="2">在寄存器寻址的指令字中，地址码字段直接指出了寄存器的编号，即 EA=R</td></tr>
<tr><td colspan="2">● 由于地址字段只需指明寄存器编号(计算机中寄存器数有限)故指令字
● 较短，节省了存储空间，因此寄存器寻址在计算机中得到广泛应用
● 执行阶段不访存，只访问寄存器，执行速度快
● 寄存器个数有限，可缩短指令字长</td></tr>
</table>

	6. 寄存器间接寻址	有效地址 EA+=(R_i)，因有效地址	
		● 有效地址在寄存器中， 操作数在存储器中，执行阶段访存 ● 便于编制循环程序	
	7. 基址寻址	基址寻址需设有基址寄存器 BR，其操作数的有效地址 EA 等于指令字中的形式地址与基址寄存器中的内容(称为基地址)相加，即 EA=A+(BR)	
		采用专用寄存器作基址寄存器	● 可扩大寻址范围 ● 有利于多道程序 ● BR 内容由操作系统或管理程序确定 ● 在程序的执行过程中 BR 内容不变，形式地址 A 可变
		采用通用寄存器作基址寄存器	● 由用户指定哪个通用寄存器作为基址寄存器 ● 基址寄存器的内容由操作系统确定 ● 在程序的执行过程中 R_0 内容不变，形式地址 A 可变
	8. 变址寻址	变址寻址与基址寻址极为相似。其有效地址 EA 等于指令字中的形式地址 A 与变址寄存器 IX 的内容相加之和，即 EA=A+(IX)	
		● 可扩大寻址范围 ● IX 的内容由用户给定 ● 在程序的执行过程中 IX 内容可变，形式地址 A 不变 ● 便于处理数组问题	
	9. 相对寻址	～的有效地址是将 PC 的内容(即当前指令地址)与指令字中的形式地址 A(A 是相对于当前指令的位移量(可正可负，补码)相加而成，即 EA=(PC)+A	
		● A 的位数决定操作数的寻址范围 ● 程序浮动 ● 广泛应用于转移指令	
	10. 堆栈寻址	要求计算机中设有堆栈。堆栈既可用寄存器组(称为硬堆栈)来实现，也可利用主存的一部分空间作堆栈(称为软堆栈) ● 硬堆栈 多个寄存器 ● 软堆栈 制定的存储空间	

(三) CISC 和 RISC 的基本概念

1．CISC(复杂指令集计算机)

随着 VLSI 技术的发展，计算机的硬件成本不断下降，软件成本不断提高，使得人们热衷于在指令系统中增加更多的指令和复杂的指令，来提高操作系统的效率，

并尽量缩短指令系统与高级语言的语义差别，以便于高级语言的编译和降低软件成本。

另外，为了做到程序兼容，同一系列计算机的新机器和高档机的指令系统只能扩充而不能减去任意一条，因此，促使指令系统越来越复杂，某些计算机的指令多达几百条。例如，DEC 公司的 VAX 11/780 计算机有 303 条指令，18 种寻址方式，我们称这些计算机为复杂指令系统计算机(complex instruction set computer，简称 CISC)。Intel 公司的 180X86 微处理器，IBM 公司的大，中计算机均为 CISC。

2．RISC(简单指令集计算机)

(1)RISC 的产生

1975 年 IBM 公司开始研究指令的合理性问题，IBM 的 John cocke 提出了 RISC 的想法。对 CISC 的测试表明：最长使用的是一些简单指令，占指令总数的 20%，但在程序中出现的频率却占 80%。而占 20%的复杂指令，为实现其功能而设计的微程序代码却占总代码的 80%。CISC 研制时间长，成本高，难于实现流水线；因此出现了 RISC 技术。

计算机执行程序所需的时间 P 可用下式表述：

$$P=I\times C\times T$$

其中，I 是高级语言程序编译后在机器上运行的机器指令数；C 为执行每条机器指令所需的平均机器周期；T 是每个机器周期的执行时间。

(2)RISC 的特点

1)优先选取使用频率最高的一些简单指令；	选用使用频度较高的一些 简单指令，复杂指令的功能由简单指令来组合
2)指令长度固定；	指令 长度固定、指令格式种类少、寻址方式少
3)只有取数/存数指令(load/store)访问内存；	只有 LOAD / STORE 指令访存
4)CPU 中的寄存器数量很多；	CPU 中有多个 通用 寄存器
5)大部分指令在一个或小于一个机器周期完成；	采用流水技术，一个时钟周期内完成一条指令
6)硬布线控制逻辑为主，不用或少用微码控制；	采用 组合逻辑 实现控制器
7)一般用高级语言编程，特别重视编译优化，以减少程序执行时间。	采用 优化 的 编译 程序

(3)RISC 的发展

1983 年，一些中小型公司开始推出 RISC 产品，由于其高性能价格比，市场占有率不断提高。1987 年 SUN 公司用 SPARC 芯片构成工作站;目前一些大公司，IBM，

DEC，Intel，Motorola 以将部分力量转移到 RISC 方面。

(4)CISC 机与 RISC 机的主要特征对比

	CISC	RISC
指令系统	复杂，庞大	简单，精简
指令数	一般大于 200	一般小于 100
指令格式	一般大于 4	一般小于 4
指令字长	一般大于 4	一般小于 4
寻址方式	不固定	固定 32 位
可访问指令	不加限制	只有 LOAD/STORE 指令
各种指令使用频率	相差很大	相差不大
各种指令执行时间	相差很大	绝大多数在一个机器周期完成
优化编译实现	很难	较容易
程序源代码长度	较短	较长
控制逻辑实现方式	绝大多数为微程序控制	绝大多数为硬连线控制
RISC 机的主要优点可归纳如下 ①充分利用 VLSI 芯片的面积 ②提高了计算机运行速度 ③便于设计，降低成本，提高可靠性 ） 有效支持高级语言程序		