

计算机组成原理与系统结构

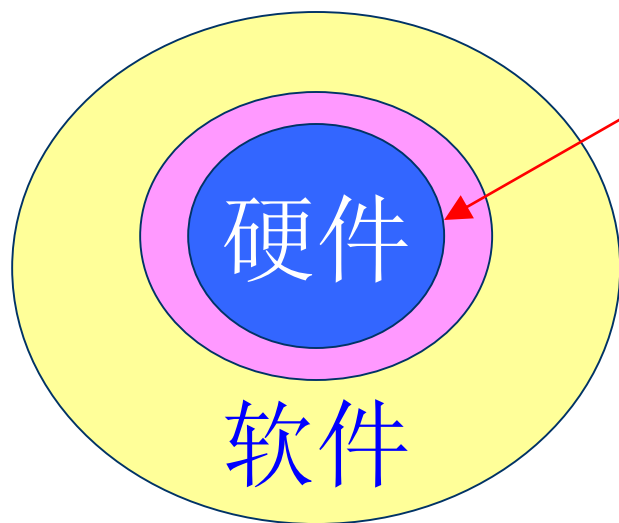
第六章 指令系统

<http://jpkc.hdu.edu.cn/computer/zcyl/dzkjdx/>



本章概述

- ❖ **指令（Instruction）**：指挥计算机执行某种操作的命令。
- ❖ **指令系统**：计算机所有指令的集合。它既表明了计算机所具有的最基本的硬件功能，又为程序员呈现了计算机的主要属性。



指令系统

指令系统是软件、硬件之间的界面
按指令系统功能构造硬件组织；
硬件支持指令系统功能的实现；
在指令系统的基础上构造系统软件。

要点：机器指令系统分类、寻址方式、格式、设计思想。
RISC技术（精简指令集计算机）



第六章 指令系统

6.1

指令格式

6.2

寻址方式

6.3

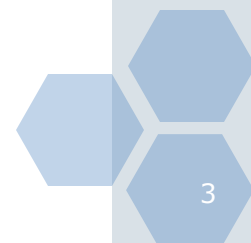
指令类型

6.4

指令系统

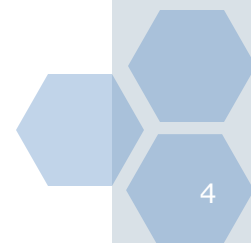
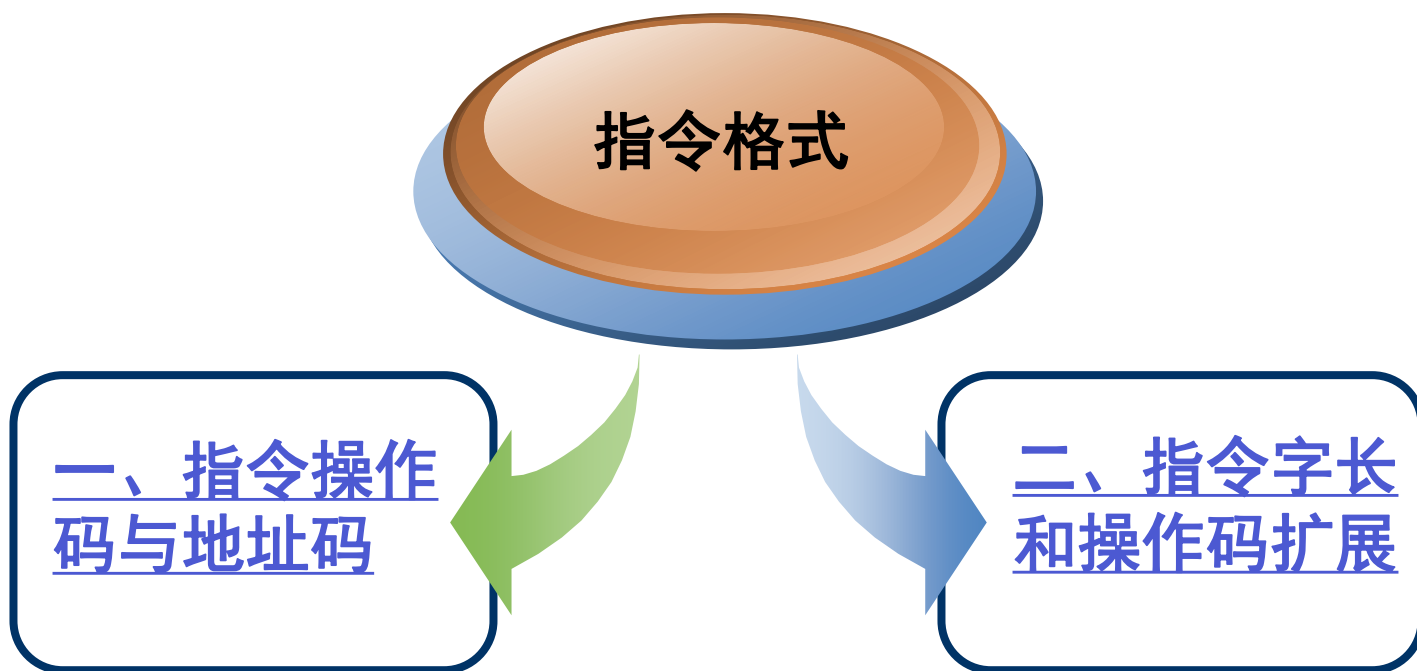
本章小结

BACK





6.1 指令格式





一、指令操作码与地址码

- ❖ 指令是由操作码和地址码两部分组成的：

操作码字段（OP）	地址码字段（A）
-----------	----------

- ❖ **操作码**：用来指明该指令所要完成的操作，如加法、减法、传送、移位、转移等等。
 - 位数反映了机器的操作种类，也即机器允许的指令条数，如果操作码有 n 位二进制数，则最多可表示 2^n 种指令。
- ❖ **地址码**：用来寻找运算所需要的操作数（源操作数和目的操作数）。
 - 地址码包括：可以直接给出操作数，也可以指出源操作数地址、目的操作数地址和下一条指令的地址。
 - 地址含义：主存的地址、寄存器地址或者I/O设备地址。



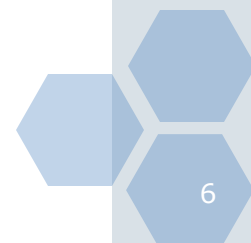
一、指令操作码与地址码

1、操作码

2、地址码

指令操作码
与地址码

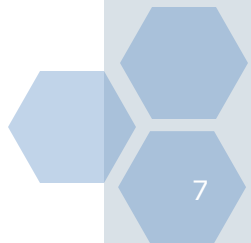
3、操作数类型





1、操作码

- ❖ 操作码长度固定：将操作码集中放在指令字的一个字段内。
 - 这种格式便于硬件设计，指令译码时间短，广泛应用于字长较长的、大中型计算机和超级小型计算机以及RISC（Reduced Instruction Set Computer）中。如IBM370和VAX-11系列机，操作码长度均为8位。
- ❖ 操作码长度不固定：指令操作码分散在指令字的不同字段中。
 - 这种格式可有效地压缩操作码的平均长度，在字长较短的微机中被广泛采用。如PDP-11，Intel8086/80386等。

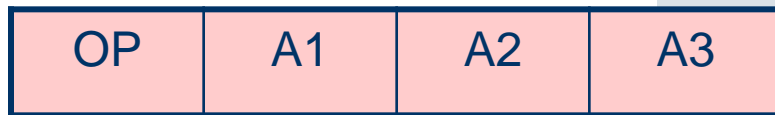




2、地址码

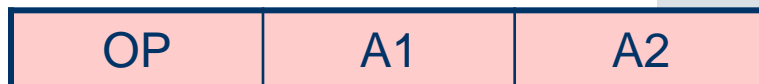
(1) 三地址指令：

- $(A1) \text{ OP } (A2) \rightarrow A3$



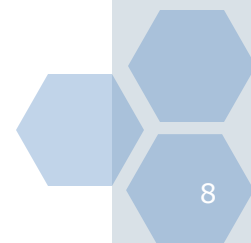
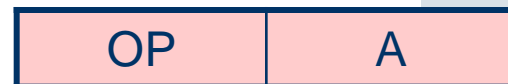
(2) 二地址指令：

- $(A1) \text{ OP } (A2) \rightarrow A1$
- A1：既是源操作数，又是目的操作数地址
- A2：源操作数地址



(3) 单地址指令：

- $(ACC) \text{ OP } (A) \rightarrow ACC$
- $\text{OP } (A) \rightarrow A$
 - 单目操作：如NEG(取反，即加负号)、INC(自增1)等指令





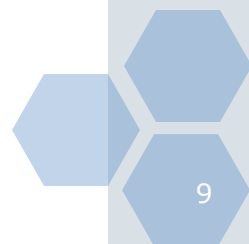
2、按照地址码分类

(4) 零地址指令

- 不涉及操作数：如NOP、HLT指令
- 操作数隐含：如PUSH、POP指令



❖对于寄存器类型的操作数，地址A指寄存器编号。





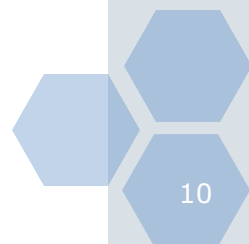
3、操作数类型

❖按照指令处理的操作数存放位置分：

- **存储器类型**：操作数存放在主存中，A为其地址信息
- **寄存器类型**：操作数存放在CPU的通用寄存器中，A为寄存器号
- **立即数类型**：操作数存放在指令（地址字段）中

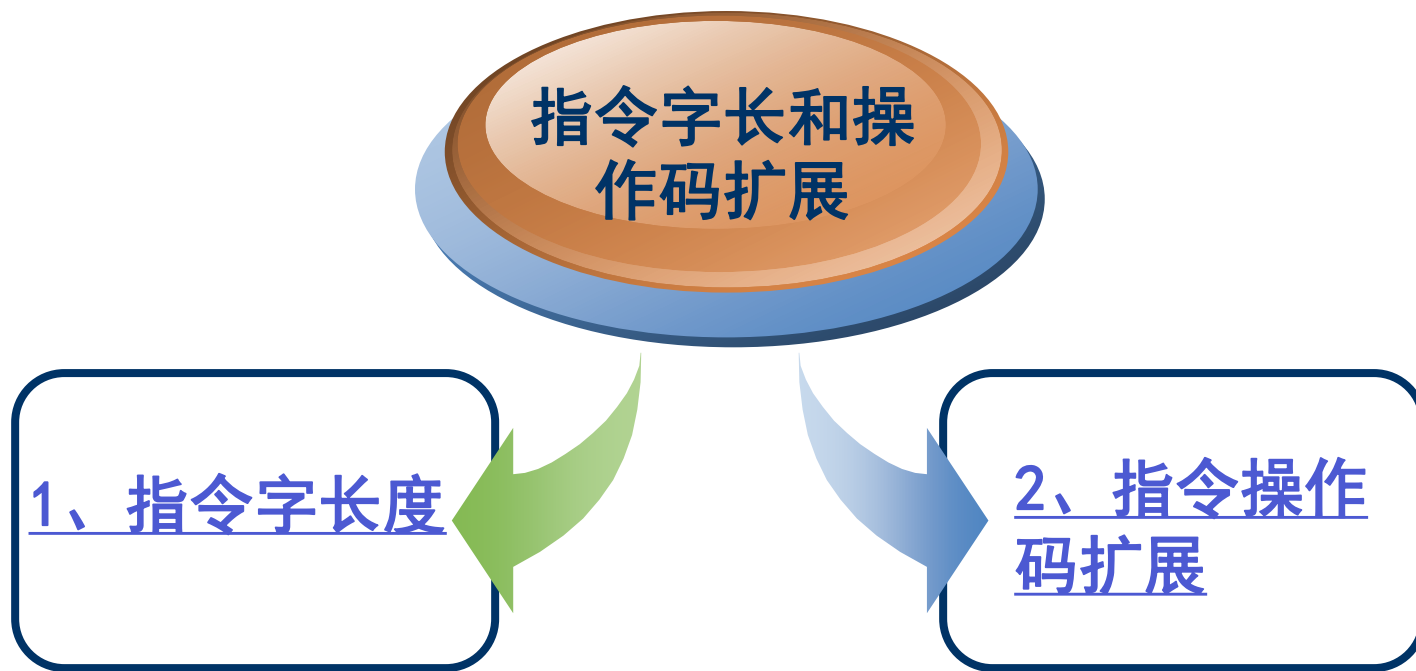
❖按照指令处理的操作数性质分：

- **地址（addresses）**：存储器地址，是无符号整数。
- **数字（numbers）**：整数、浮点数、十进制数。
- **字符（characters）**
- **逻辑数据**：真假两种状态





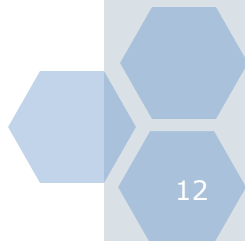
二、指令字长和操作码扩展





1、指令字长度

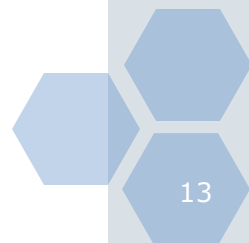
- ❖ 机器指令是用二进制机器字来表示的，表示一条指令的机器字，就称为指令字。一条指令中所包含的二进制码的位数，称为指令字长度或指令字长。它主要取决于操作码的长度、操作数地址的长度和操作数地址的个数。不同机器的指令字长是不相同的。
- ❖ 按指令长度固定与否可以分为：
 - 固定指令字长的指令：所有指令的字长均相等，一般等于机器字长。
 - 可变指令字长的指令：指令字长不固定，通常取字节的整数倍。（8位的整数倍）





1、指令字长度

- ❖ 按照指令字长与机器字长的关系分类：
 - 短格式指令：指令字长小于或等于机器字长。
 - 长格式指令：指令字长大于机器字长。
 - 一个机器的指令系统中，**短格式指令和长格式指令可以并存**，通常将最常用的指令设计成短格式指令，可以节省存储空间、提高指令的执行速度。





2、指令操作码扩展

- ❖ 指令的操作码通常有两种编码格式：固定操作码长度的格式和可变操作码长度格式
- ❖ 在设计操作码不固定的指令系统时，应安排指令使用频度高的指令占用短的操作码，对使用频度低的指令可占用较长的操作码，这样可以缩短经常使用的指令的译码时间。
- ❖ 采用**扩展操作码**技术，使操作码的长度随地址数的减少而增加，即不同地址数的指令可以具有不同长度的操作码，从而可以有效地缩短指令字长。**指令操作码扩展技术是一种重要的指令优化技术，它可以缩短指令的平均长度，增加指令字所能表示的操作信息。但指令操作码扩展技术需要更多的硬件支持，它的指令译码更加复杂，使控制器设计难度增大。**



举例

4位操作码,15条三地址指令

OP	A_1	A_2	A_3
0000	A_1	A_2	A_3
0001	A_1	A_2	A_3
:	:	:	:
1110	A_1	A_2	A_3

8位操作码,15条二地址指令

1111	0000	A_2	A_3
1111	0001	A_2	A_3
:	:	:	:
1111	1110	A_2	A_3

12位操作码,15条一地址指令

1111	1111	0000	A_3
1111	1111	0001	A_3
:	:	:	:
1111	1111	1110	A_3

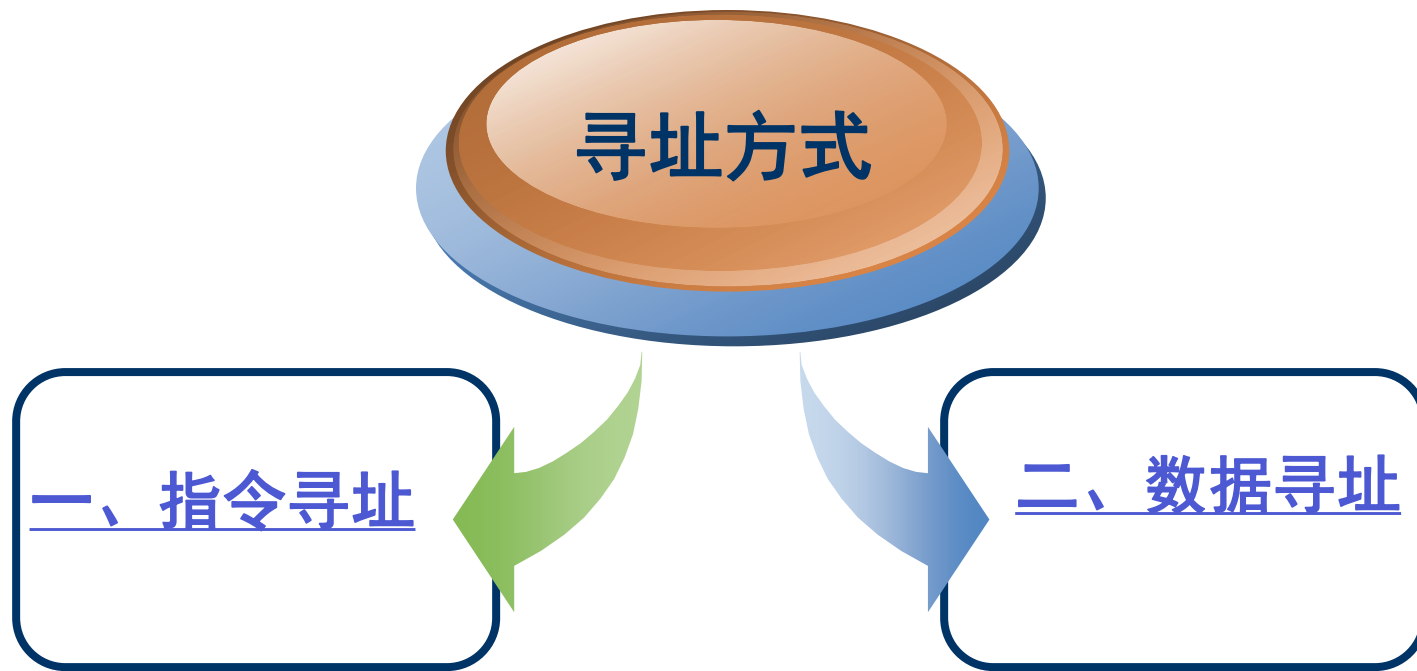
16位操作码,16条零地址指令

1111	1111	1111	0000
1111	1111	1111	0001
:	:	:	:
1111	1111	1111	1111





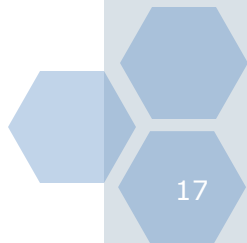
6.2 寻址方式





寻址方式

- ❖ **定义：**指确定本指令的操作数地址以及下一条将要执行的指令的地址的方法。
- ❖ **硬件完成。**可以看做是访存方式
- ❖ **好处：**丰富程序设计手段+压缩程序占用空间





一、指令寻址

❖ 1、顺序寻址方式

- 控制器中使用程序计数器PC来指示指令在内存中的地址。在程序顺序执行时，指令的地址码由PC自加1得出。
- 指令在内存中按顺序存放，当顺序执行一段程序时，根据PC从存储器取出当前指令，PC自动+1，然后执行这条指令；接着又根据PC指示从存储器取出下一条指令，PC自动+1，执行……。

❖ 2、跳跃寻址方式

- 当程序执行转移指令时，程序不再顺序执行，而是跳转到另一个地址去执行，此时，由该条转移指令的地址码字段可以得到新指令地址，然后将其置入PC中。





二、数据寻址

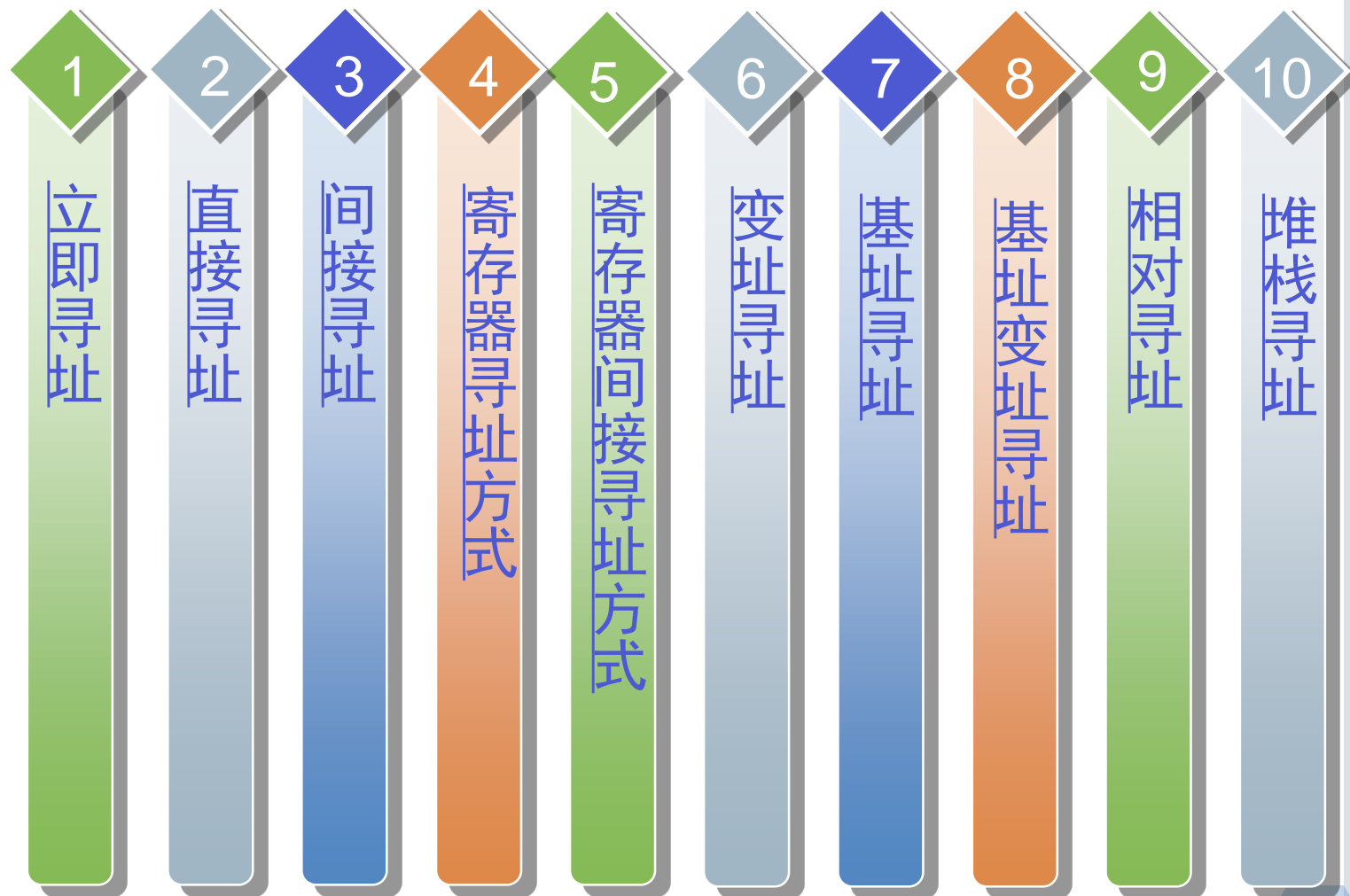
- ❖ 指令的地址码字段，通常都不代表操作数的真实地址，把它称作**形式地址**，记为A。操作数的真实地址称为**有效地址**，记作EA，它是由寻址方式和形式地址共同来确定的。

操作码	寻址特征	形式地址 A
-----	------	--------

图 6-5 一种一地址指令的格式



二、数据寻址





1、立即寻址 (Immediate Addressing)

❖ 操作数在指令的地址码字段，即：

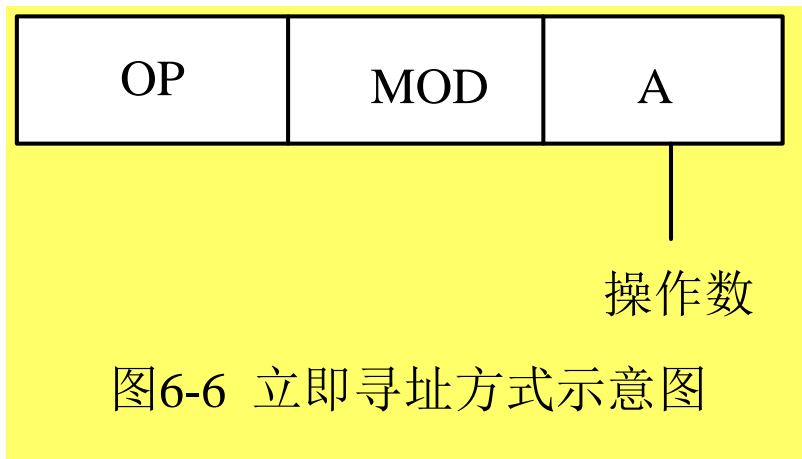
■ $DATA = A$

❖ 例如：

`MOV AL, 5`

`MOV AX, 3064H`

`MOV AL, 'A'`

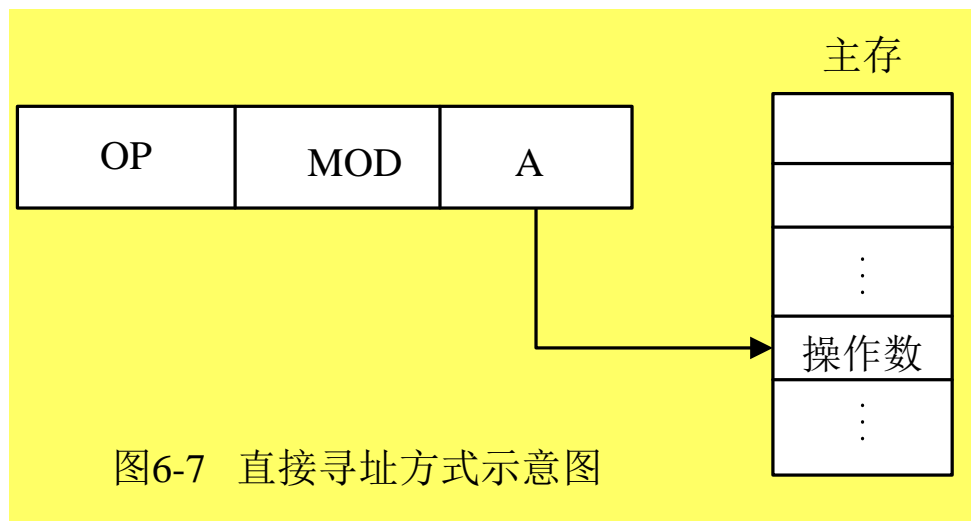




2. 直接寻址 (Direct Addressing)

❖ 操作数位于存储器中，操作数所在的存储器单元的地址存放在指令的地址字段A中，即：

- $DATA = (EA)$
- $EA = A$





3、间接寻址(Indirect Addressing)

❖ 操作数位于存储器中，操作数所在的存储器单元地址也存放在存储器中，该存储器地址则存放在指令的地址字段中，即：

■ $DATA = (EA)$

■ $EA = (A)$

❖ 即：A为操作数地址的地址

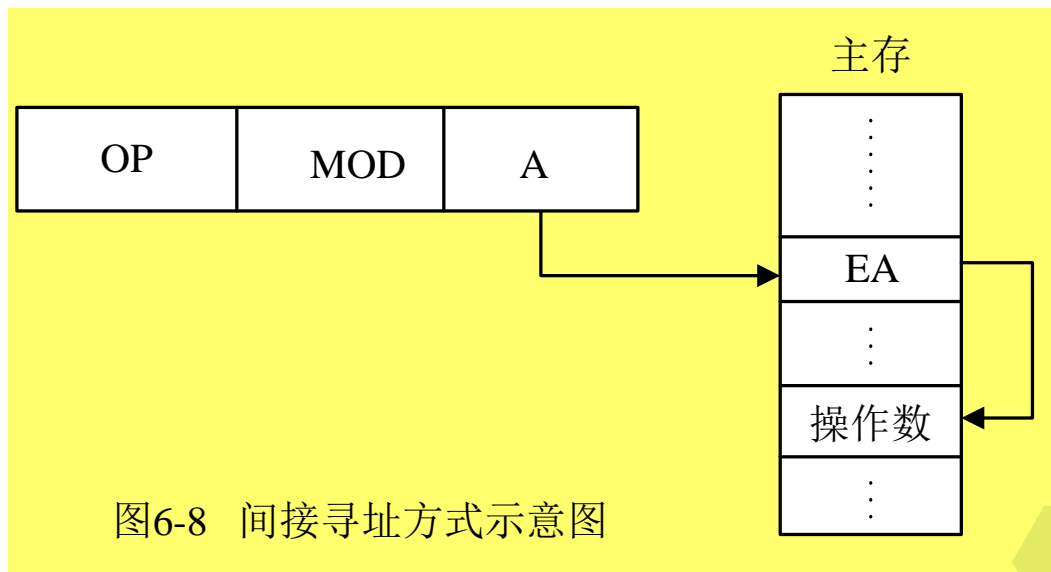
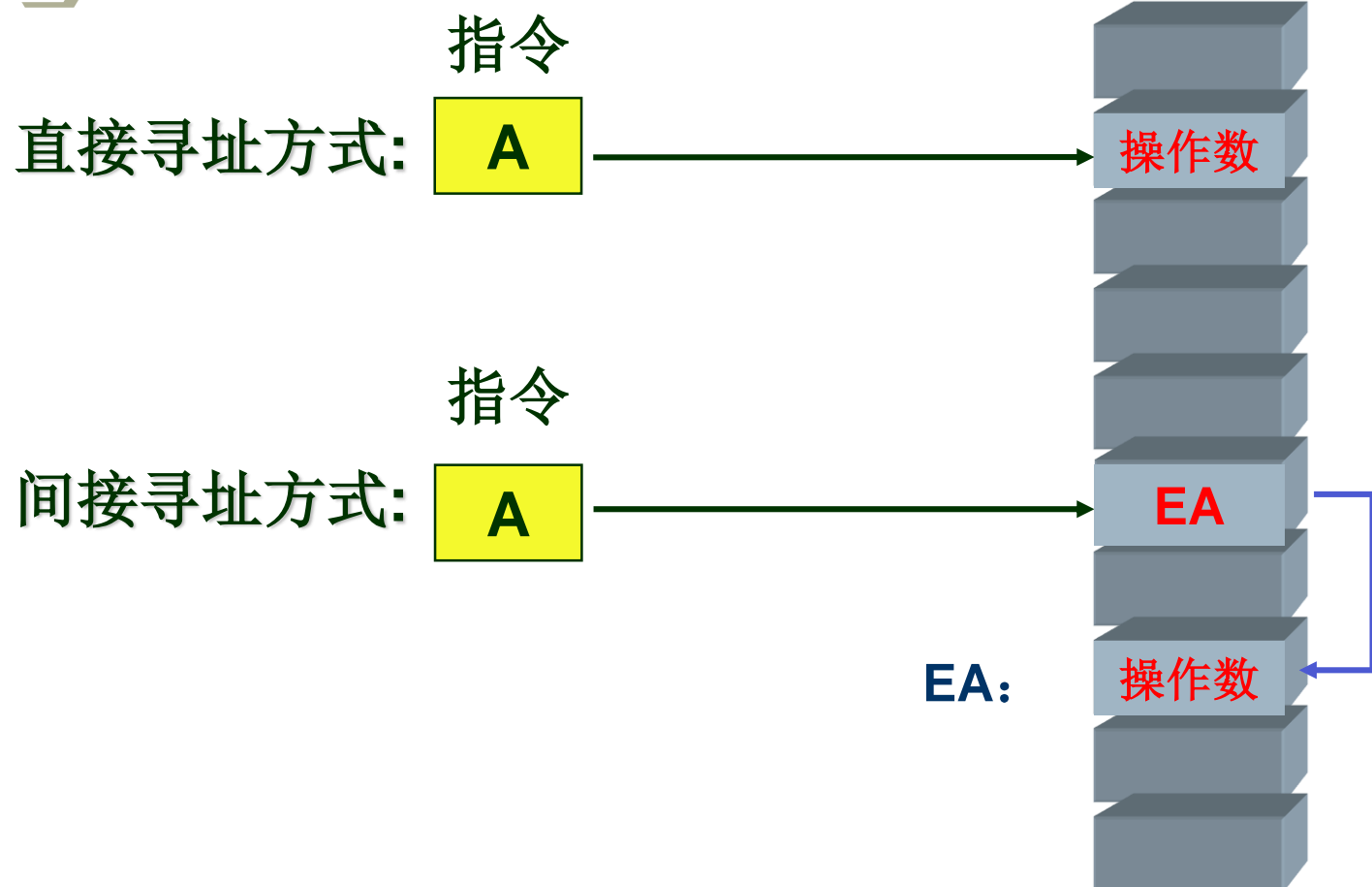


图6-8 间接寻址方式示意图



存储器





4、寄存器寻址方式（ Register Addressing ）

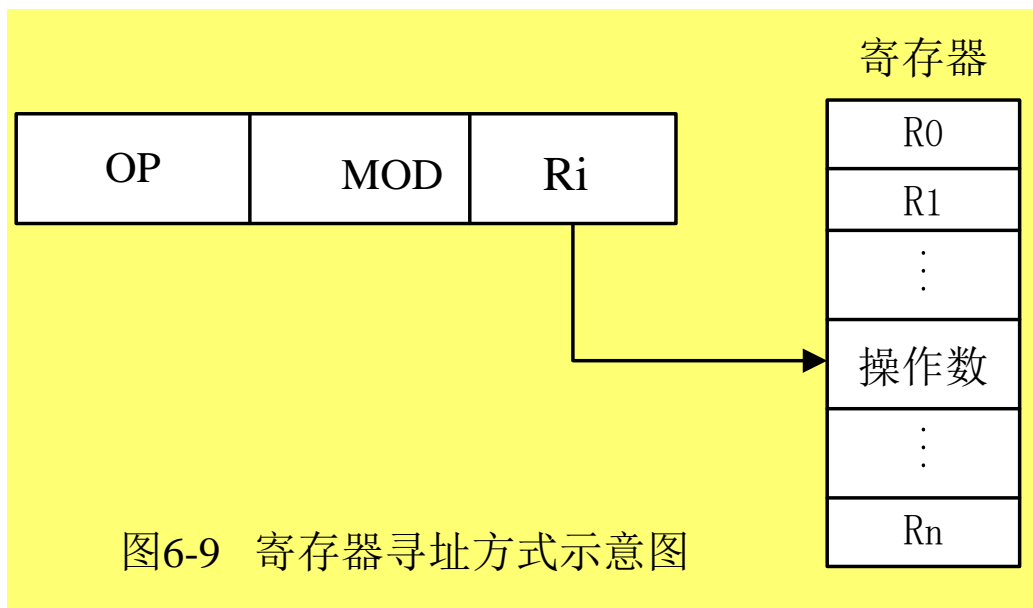
❖ 操作数位于寄存器中，操作数所在的寄存器编号存放在指令的地址字段A中，即：

■ $DATA = (R_i)$

■ $EA = R_i$

MOV AX, BX

MOV AL, BH





5、寄存器间接寻址方式

❖ 操作数位于存储器中，操作数所在的存储器地址存放在寄存器中，而该寄存器编号存放在指令的地址字段A中，即：

- $DATA = (EA)$

- $EA = (R_i)$

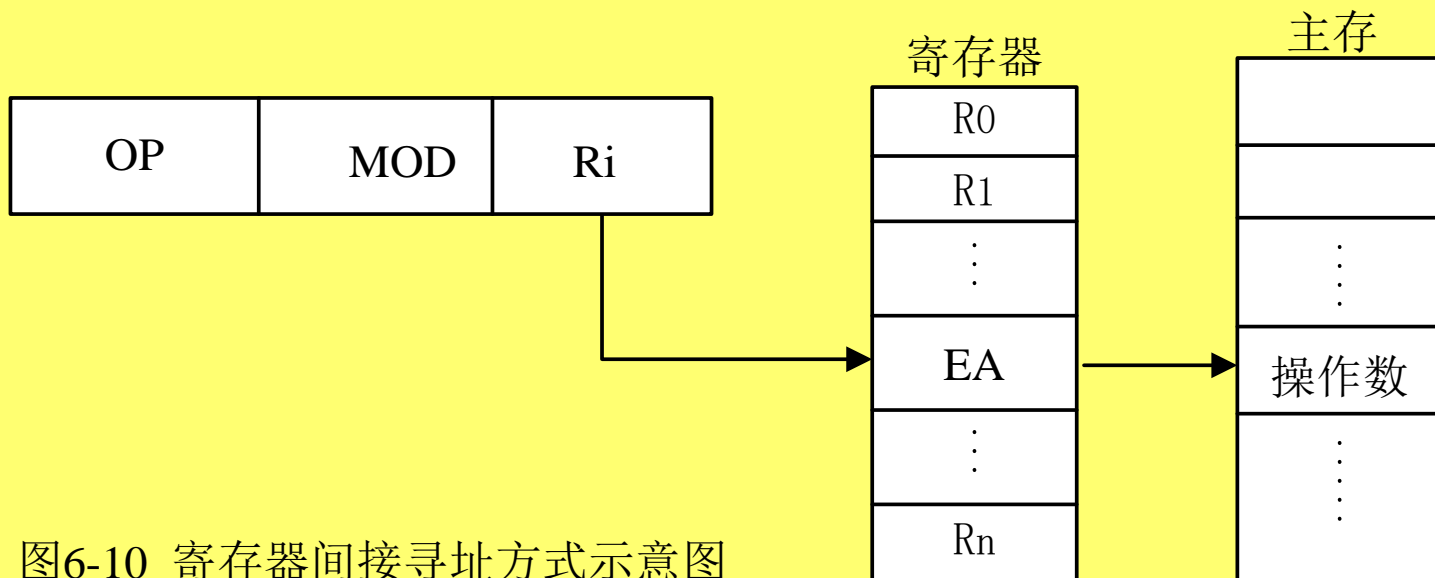
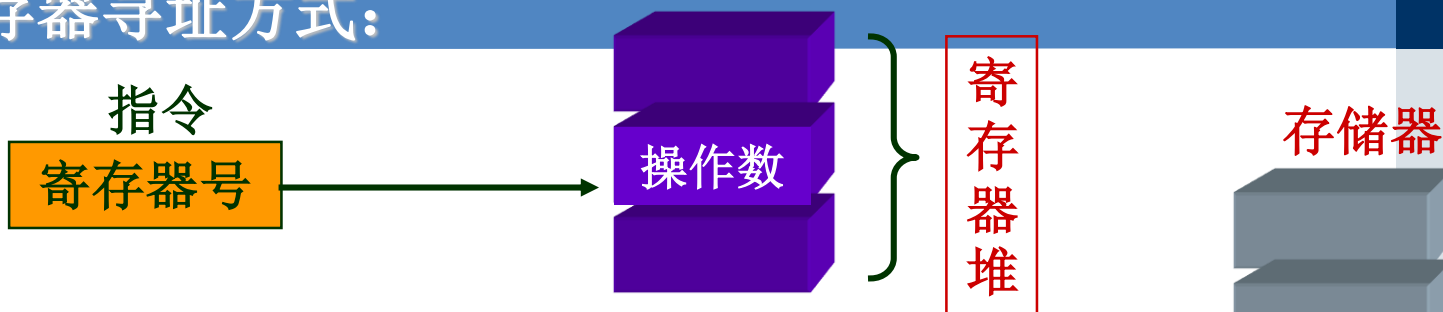


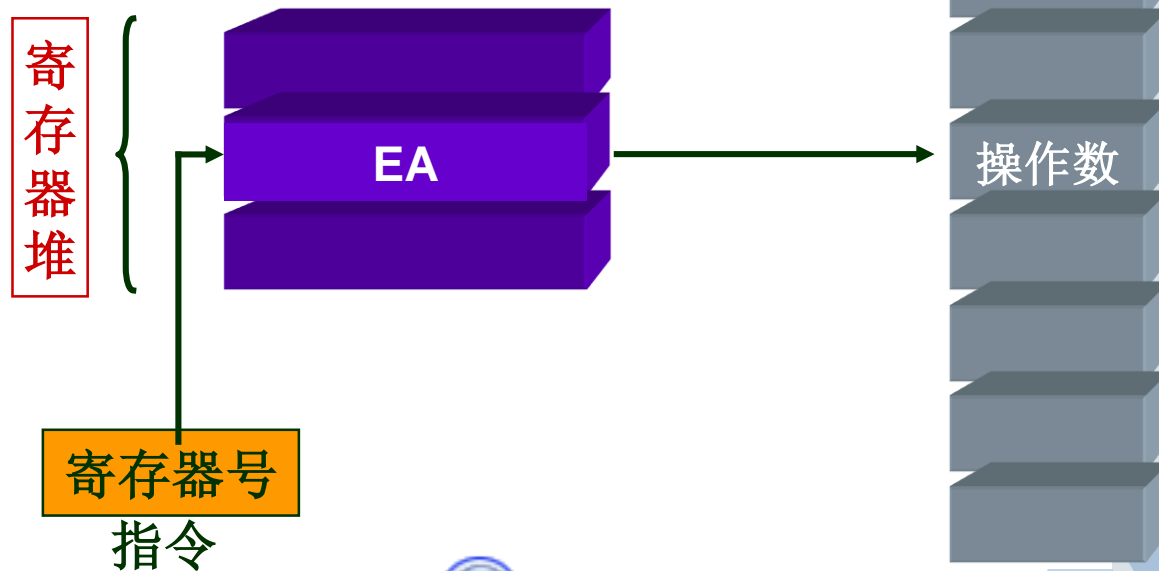
图6-10 寄存器间接寻址方式示意图



寄存器寻址方式:



寄存器间接寻址方式:





6、变址寻址（Indexed Addressing）

❖ 操作数位于存储器中，操作数所在的存储器地址EA由变址寄存器 R_i 和指令的地址字段A指出：

- $DATA = (EA)$
- $EA = (R_i) + A$

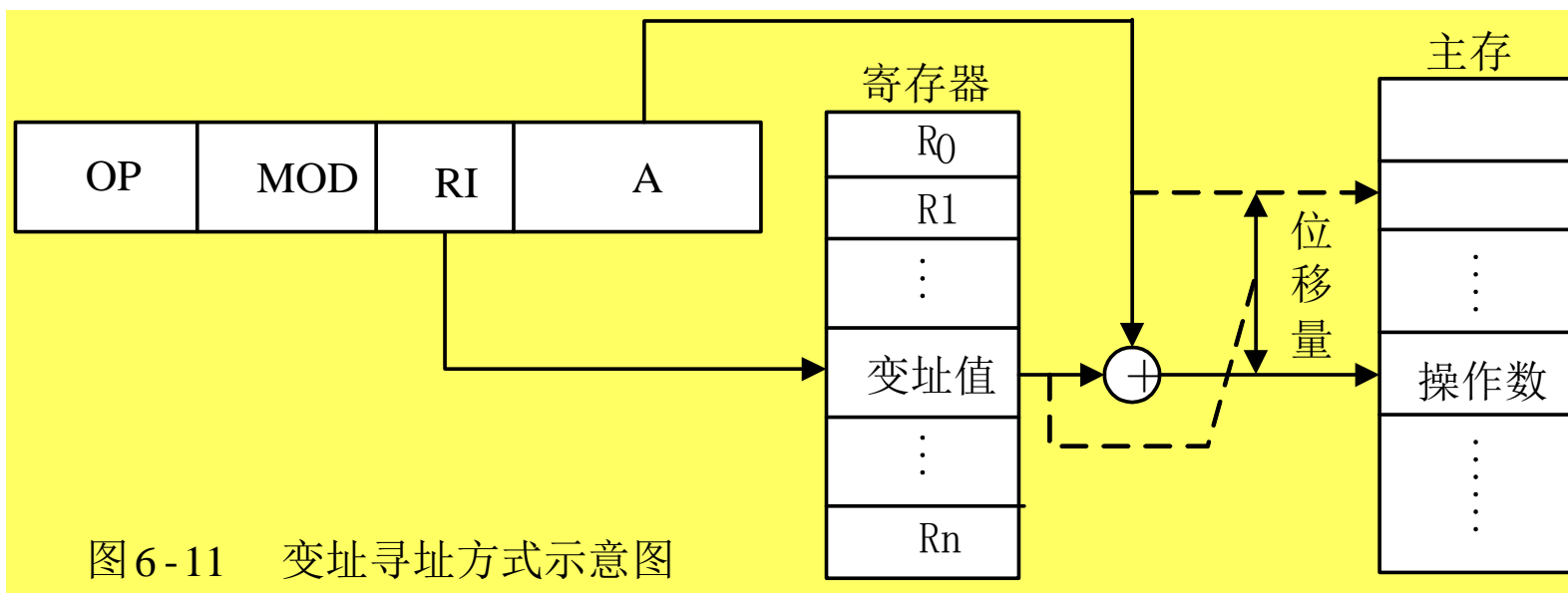


图6-11 变址寻址方式示意图

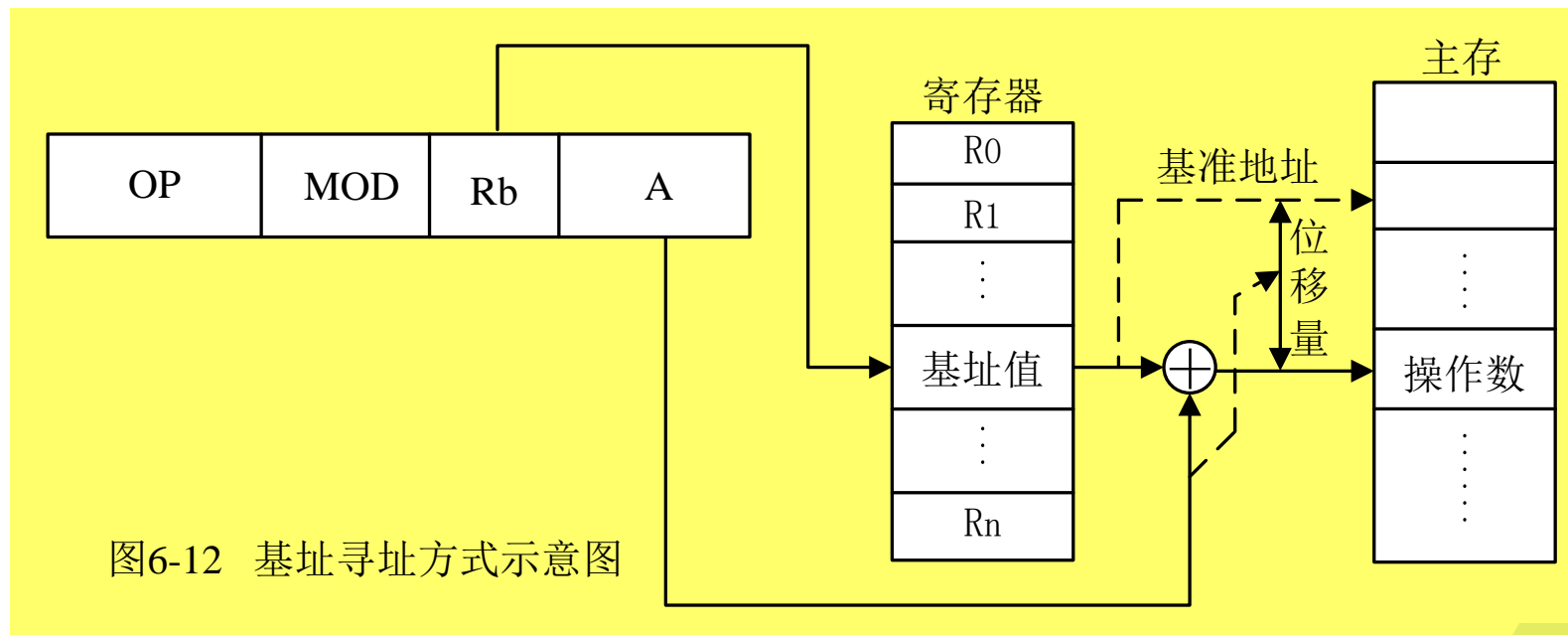




7、基址寻址（Based Addressing）

❖ 操作数位于存储器中，操作数所在的存储器地址EA由基址寄存器 R_b 和指令的地址字段A指出：

- $DATA = (EA)$
- $EA = (R_b) + A$





基址寻址与变址寻址的**不同**之处：

变址寻址在于实现程序块的规律变化。变址寻址面向用户，多用于字符串处理、数组运算等成组数据处理。

基址寻址可以扩大寻址范围（整个主存范围）。

将整个存储空间分成若干个段，段的首地址存放在基址寄存器中，操作数的存储地址与段的首地址的距离即段内偏移量由指令直接给出。操作数存储单元的实际有效地址就等于基址寄存器的内容与段内偏移量之和。

改变基址寄存器的内容(基准量)并由指令提供位移量就可以访问存储器的任一单元。基址寄存器用于程序装配，可为浮动程序分配存储单元。

基址寻址面向系统，解决程序的存储定位和扩大寻址空间等问题。





8、基址变址寻址

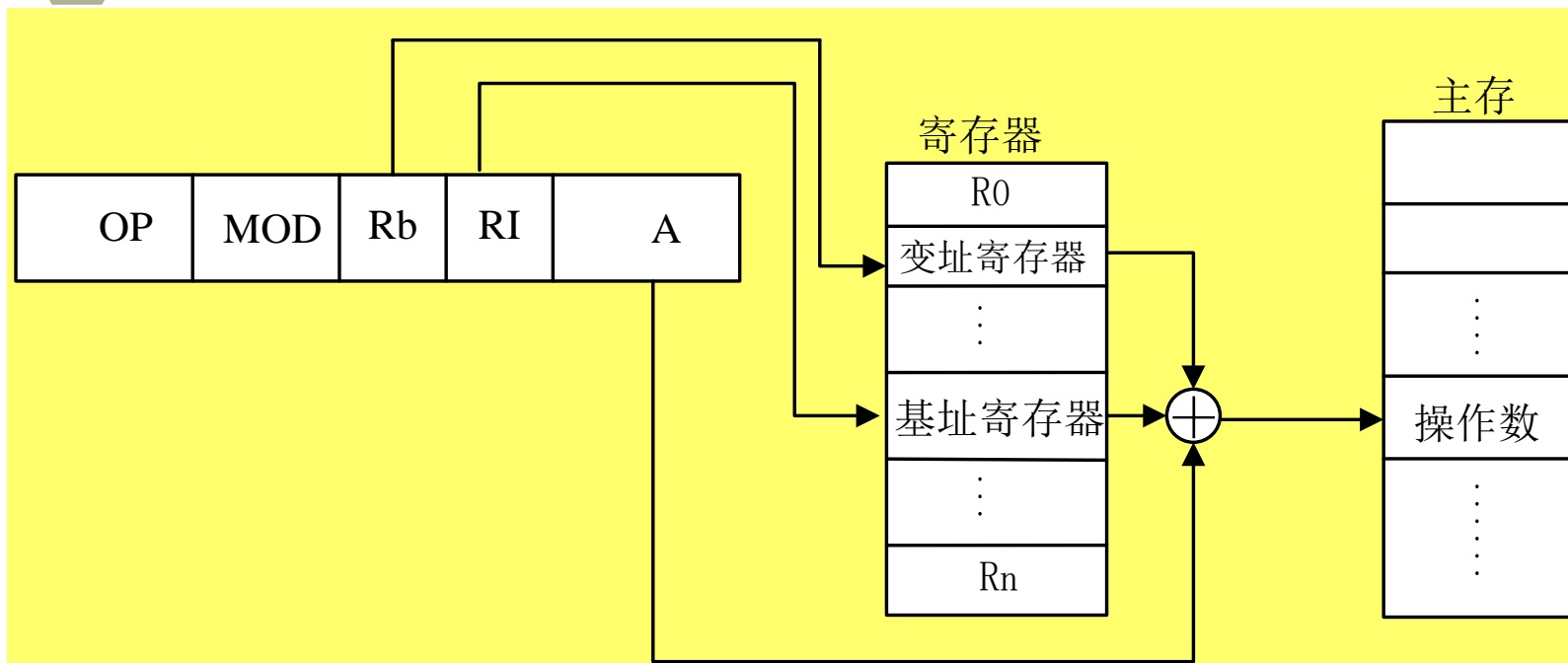
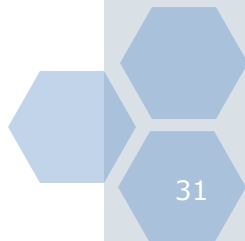


图6-13 基址变址寻址方式示意图





9、相对寻址 (Relative Addressing)

❖ 操作数位于存储器中，操作数所在的存储器地址EA由**程序计数器PC**和指令的地址字段A指出：

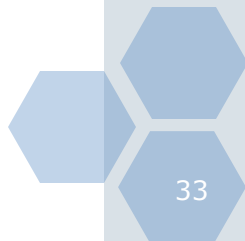
- $DATA = (EA)$
- $EA = (PC) + A$
- A通常称作**相对偏移量**。
- 相对寻址主要用于转移指令，执行之后，程序将转移到 **(PC) + 偏移量** 为地址的指令去执行。
- **偏移量可正、可负**，通常用补码表示，即可相对PC值向后或向前转移。
- 相对寻址是以当前指令地址为基准，还是以下一条指令为基准？绝大多数计算机在执行当前指令时，因为PC的内容已经增加到指向下一条指令的地址，因此在计算偏移量时，要考虑下一条指令地址为基准，根据转移目的地址与基准地址的差值来计算偏移量，即 $偏移量 = 转移目的地址 - (PC)$ 。





10、堆栈寻址（ Stack Addressing ）

- ❖ 操作数位于存储器中，操作数所在的存储器地址EA由堆栈指针寄存器SP隐含指出，通常用于堆栈指令。
- ❖ 堆栈是由若干个连续主存单元组成的先进后出（first in last out，即FILO）存储区，第一个放入堆栈的数据存放在栈底，最近放入的数据存放在栈顶。栈底是固定不变的，而栈顶是随着数据的入栈和出栈在时刻变化。栈顶的地址由堆栈指针SP指明。
- ❖ 一般计算机中，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，称为上推堆栈；也有少数计算机相反，称为下推堆栈。
- ❖ 堆栈寻址主要用来暂存中断和子程序调用时现场数据及返回地址。



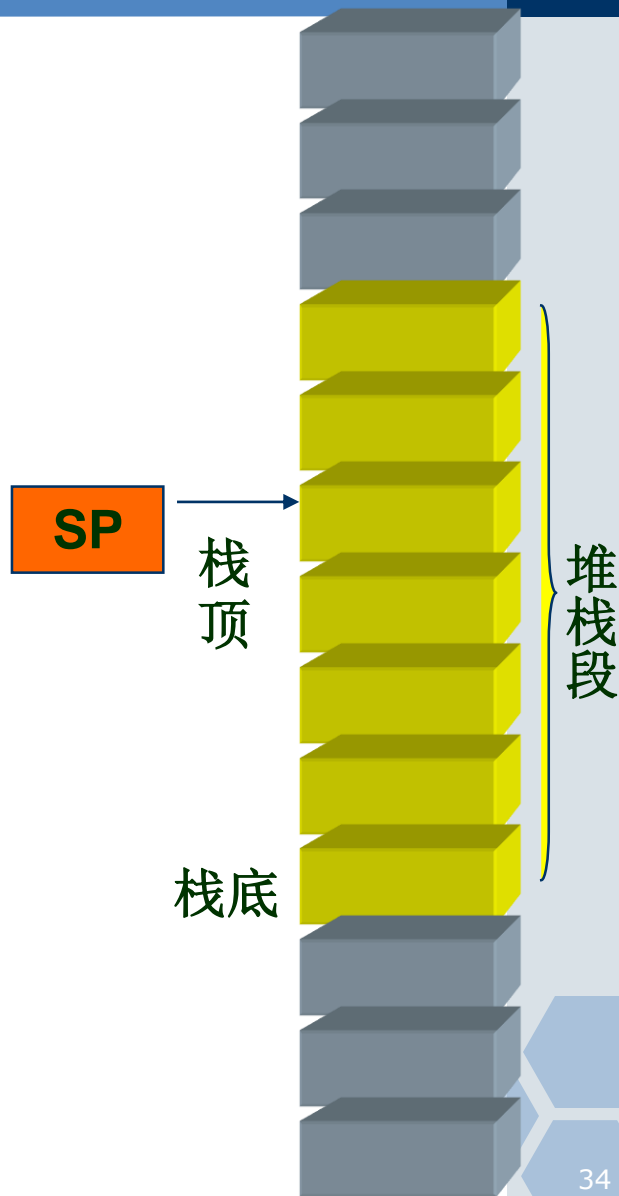


堆栈的结构

存储器

❖堆栈的操作：**压入**（PUSH）和**弹出**（POP），对应PUSH和POP指令，假设数据字长为1B

- 压入指令 PUSH Ri：将Ri寄存器内容压入堆栈。其操作是：
 - $(SP) - 1 \rightarrow SP, (Ri) \rightarrow (SP)$
- 弹出指令POP Ri：从堆栈中弹出1个数据送Ri寄存器，其操作是：
 - $((SP)) \rightarrow Ri, (SP) + 1 \rightarrow SP$
- 其中(SP)表示堆栈指针SP的内容；
((SP))表示SP所指的栈顶的内容。





6.3 指令类型

❖ 1. 数据传送指令

- 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。数据能够被从源地址传送到目的地址，而源地址中数据不变，——拷贝。

❖ 2. 算术逻辑运算指令

- 实现算术运算（加、减、乘、除等）和逻辑运算（与、或、非、异或）。有些计算机还设置有位操作指令，如位测试（测试指定位的值）、位清零、位求反指令等。

❖ 3. 移位操作指令

- 可分为算术移位、逻辑移位和循环移位。

算术移位：左移时空位补0而符号位进标志位，右移时空位复制符号位而溢出位进标志位。

逻辑移位：整体移位，空位补0，溢出进标志位。

循环移位：有不带进位循环和带进位循环。前者循环后的溢出位进标志位，后者与标志位一起循环。

（详情在书109-110页）



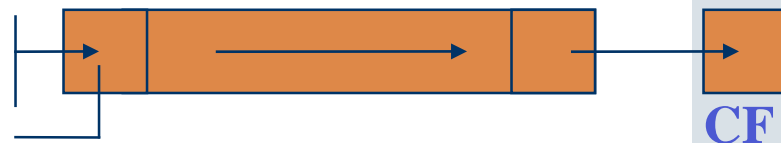
逻辑左移、算术左移



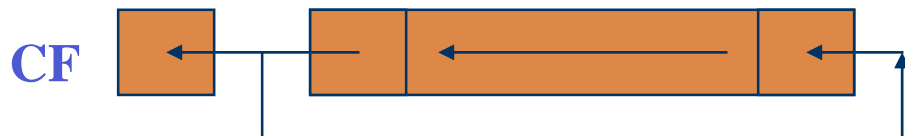
逻辑右移



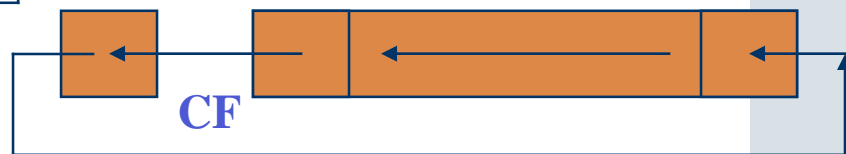
算术右移



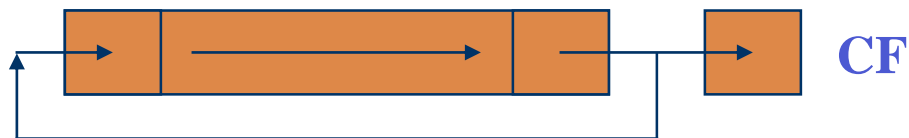
循环左移



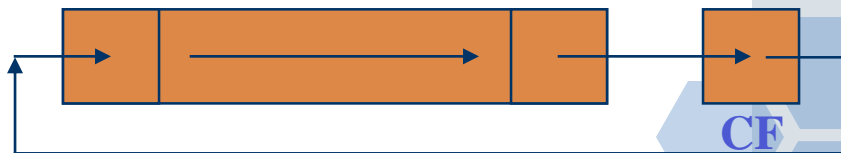
带进位循环左移



循环右移



带进位循环右移

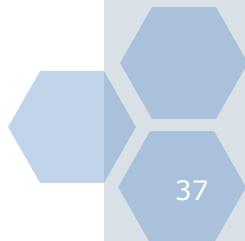




6.3 指令类型

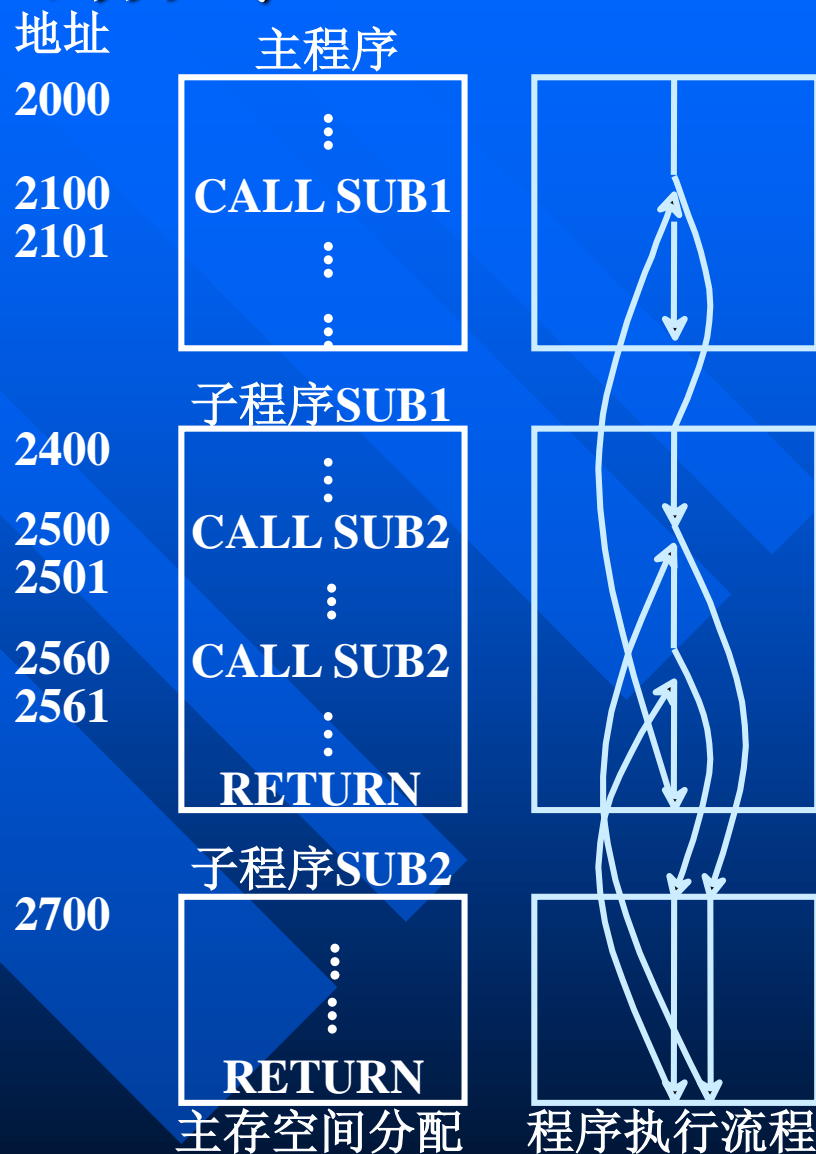
❖ 4. 程序控制类指令：控制程序的流向

- ① 无条件转移指令：无条件转至目的地址处执行。
- ② 条件转移指令：条件满足转至目的地址处执行，否则顺序执行
- ③ 陷阱指令
 - 陷阱其实是一种意外事故的中断。当计算机出现错误或故障致使计算机不能正常工作时，计算机就发出陷阱信号，暂停当前程序的执行，转入故障处理程序进行相应的处理。一般是作为隐指令（即指令系统中不提供的指令）使用。
- ④ 调用指令和返回指令（详见下页）



④调用指令和返回指令

- 调用指令**CALL**用于从当前的程序位置转至子程序的入口；
- 返回指令**RETURN**用于子程序执行完后重新返回到原程序的断点。





6.3 指令类型

- ❖ **5. 堆栈操作指令：**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，而弹出指令（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。
- ❖ **6. 输入输出指令：**它完成从外设端口读入一个数据到CPU的寄存器内，或将数据从CPU的寄存器输出到某外设的端口中。
- ❖ **7. 处理器控制指令：**包括等待指令、停机指令、空操作指令、开中断指令等
- ❖ **8. 特权指令：**主要用于分配和管理系统资源。特权指令只能给操作系统或其他系统软件，而不能提供给用户使用，以防止破坏系统或其他用户信息





6.4 指令系统的设计技术

一

指令系统的要求

二

指令系统的发展

三

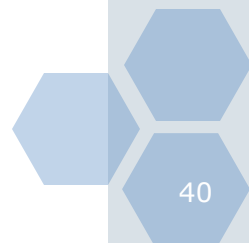
CISC的特点

四

RISC的特点

五

指令系统举例





一、指令系统的要求

指令系统的设计是计算机系统设计中的一个核心问题

设计一个完善的指令系统应满足如下4个方面的要求：

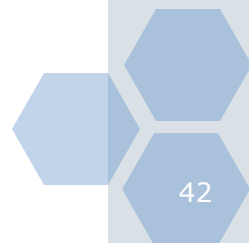
- ❖ 1、完备性：指指令系统直接提供的指令足够使用，而不必用软件来实现。
- ❖ 2、有效性：是指利用该指令系统所编写的程序能够高效地运行。程序占据存储空间小、执行速度快。
- ❖ 3、规整性：
 - 对称性：所有的指令都可使用各种寻址方式；
 - 匀齐性：指令可以支持各种数据类型；
 - 指令格式和数据格式的一致性：指令长度和数据长度有一定的关系，以方便处理和存取。
- ❖ 4、兼容性：系列机各机型之间具有相同的基本结构和共同的基本指令集，且“向上兼容”，即低档机上运行的软件可以在高档机上运行。





二、指令系统的发展

- ❖ “复杂指令集计算机”，简称CISC（Complex Instruction Set Computer）
 - 指令格式不固定，寻址方式丰富，功能复杂
 - 一些比较简单的指令，在程序中仅占指令系统中指令总数的20%，但出现的频率却占80%；占指令总数20%的最复杂的指令，却占用了控制存储器容量的80%，且使用频率却不高。





二、指令系统的发展

精简指令集计算机 (Reduced Instruction Set Computer, 简称RISC)

RISC由来

计算机的不断升级扩充，同时又兼容过去产品使指令系统日趋复杂，形成了“复杂指令集计算机(CISC)”。如VAXII / 780有303条指令，18种寻址方式。Pentium机有191条，9种寻址方式。

复杂指令系统增加硬件复杂性，降低机器运行速度。

经实际分析发现：

- 1、各种指令使用频率相差悬殊。80%指令使用很少。
- 2、指令系统的复杂性带来系统结构的复杂性，增加了设计时间和售价，也增加了VLSI设计负担，不利于微机向高档机器发展。
- 3、复杂指令操作复杂、运行速度慢。

由此提出“精简指令集计算机(RISC)”的概念。

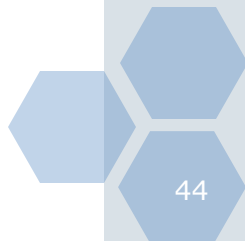




三、CISC的特点

❖ 早期CISC指令系统的主要特点是：

1. 指令系统复杂。具体表现为指令条数多、寻址方式多、指令格式多。
2. 指令串行执行，大多数指令需要多个时钟周期完成。
3. 采用微程序控制，因为微程序控制器适合于实现CISC指令执行过程的控制。
4. 有较多的专用寄存器，大部分运算所需的数据均需访问存储器获取。
5. 编译程序难以用优化措施生成高效的目标代码程序。

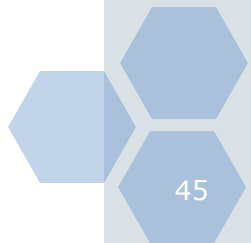




四、RISC的特点

❖ 大部分RISC机具有以下特点：

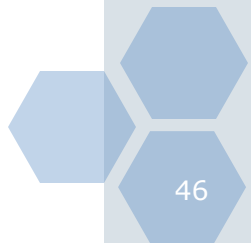
- （1） 指令系统设计时选择一些使用频率较高的简单指令，且选择一些很有用但不复杂的指令。
- （2） 指令长度固定，指令格式种类少，寻址方式种类少。
- （3） 只有取数/存数指令访问存储器，其余指令的操作都在寄存器之间进行。





四、RISC的特点

- (4) 采用流水线技术、超级标量及超级流水线技术，增加了指令执行的并行度，使得一条指令的平均指令执行时间小于一个机器周期。
- (5) CPU中通用寄存器数量相当多，可以减少访存次数。
- (6) 以硬布线控制逻辑为主，不用或少用微码控制。
- (7) 采用优化的编译程序，力求有效地支持高级语言程序。





同CISC比较，RISC的优点

- ❖ (1) 可以充分利用VLSI芯片面积
- ❖ (2) 可以提高计算机运算速度
 - 指令数、寻址方式和指令格式的种类都较少，且指令的编码很有规律，使指令译码加快。
 - 在简化指令的情况下，硬布线连接比微程序控制的延迟小，可缩短CPU的周期。
 - CPU的通用寄存器多，减少了访存次数，加快了速度
 - 大部分指令能在一个周期内完成，特别适合于流水线工作。
 - 有的RISC机采用寄存器窗口重叠技术，程序嵌套时不必将寄存器内容保存到存储器中，加快了速度。

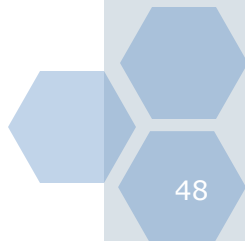


同CISC比较，RISC的优点

- ❖ (3) 设计容易，可降低成本，提高可靠性。
- ❖ (4) 能有效支持高级语言程序

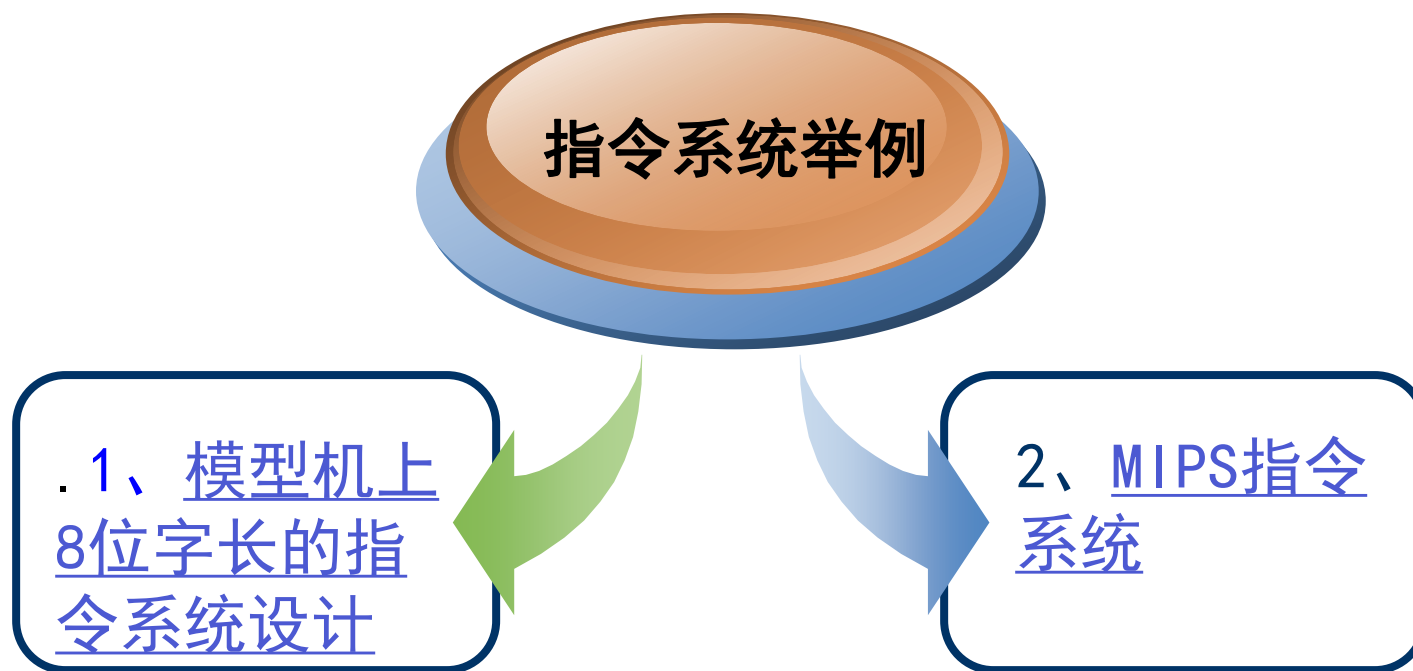
RISC靠编译程序的优化来支持高级语言程序。

- 指令少，寻址方式少，反而使编译程序容易选择更有效的指令和寻址方式。
- 通用寄存器多，可尽量安排快速的寄存器操作，使编译程序的代码优化效率较高。
- 有的RISC机采用寄存器窗口重叠技术，使过程间的参数传送快，且不必保存与恢复现场，因而能直接支持调用子程序和过程的高级语言程序。
- 在编译时尽量做好程序优化工作，从而缩短程序执行时间。





五、指令系统举例





1、模型机上8位字长的指令系统设计



模型机指
令格式



模型机寻
址方式



模型机
指令系
统设计





① 模型机指令格式

❖ 格式1：一般指令格式

17 16 15 14 13 12 11 10

OP: 指令操作码, 4位, 用于对16条机器指令进行编码, 是识别指令的标志。

OP	SR	DR
DATA/ADDR/DISP/X		

SR: 源寄存器号, 2位, 用于对4个通用寄存器R0、R1、R2、R3的选择, 其内容送总线, 作为源操作数之一。

DR: 目的寄存器号, 2位, 用于对4个通用寄存器R0、R1、R2、R3的选择, 其内容可送总线, 也可以从总线上接收数据, 通常作为目的操作数。

DATA/ADDR/DISP/X: 指令的第二个字, 可有可无, 其含义也可以由用户自定义, 可以是立即数, 可以是直接/间接地址, 也可以是其它寻址方式用到的地址信息, 如相对偏移量、形式地址等等。



格式2：带寻址方式码的指令格式

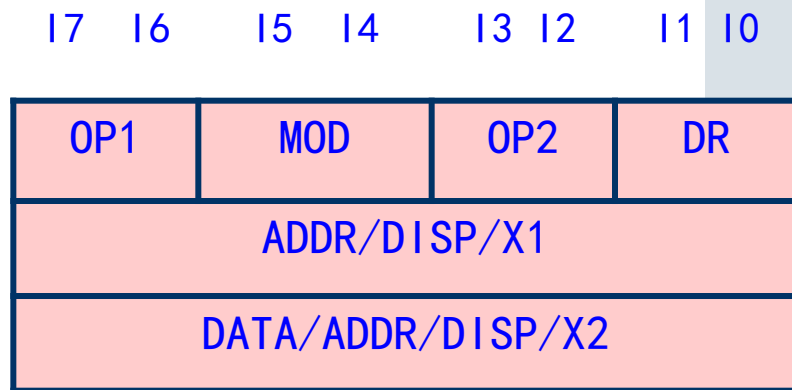
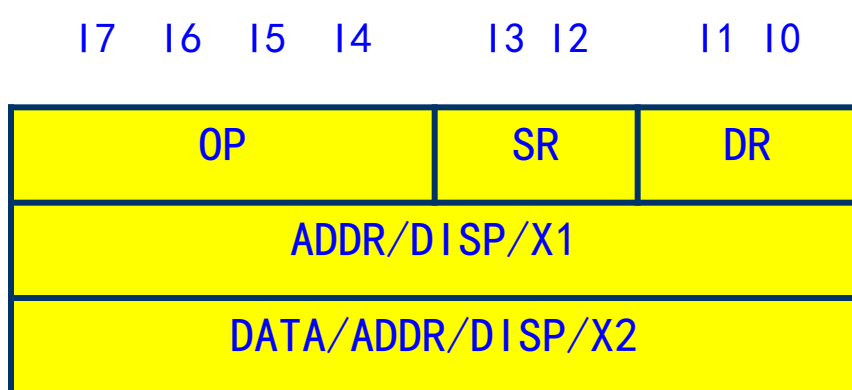
17 16 15 14 13 12 11 10

OP1	MOD	OP2	DR
ADDR/DISP/X			

- ❖ **OP1**：第一指令操作码，2位，是带寻址方式码的指令（4条）的特征位。
- ❖ **MOD**：寻址方式码，2位，用于对4种寻址方式的编码，至于4种寻址方式的定义，可以自行设计，例如：可设计为直接、间接、变址、相对寻址。
- ❖ **OP2**：第二指令操作码，2位，是4条带寻址方式码的指令本身的编码。
- ❖ **DR**：同格式一。
- ❖ **ADDR/DISP/X**：指令的第二个字，为寻址方式中所用到的直接/间接地址ADDR，或者是相对寻址的偏移量DISP，或者是变址寻址的形式地址X



格式3：三字指令



❖ 指令包含三字：

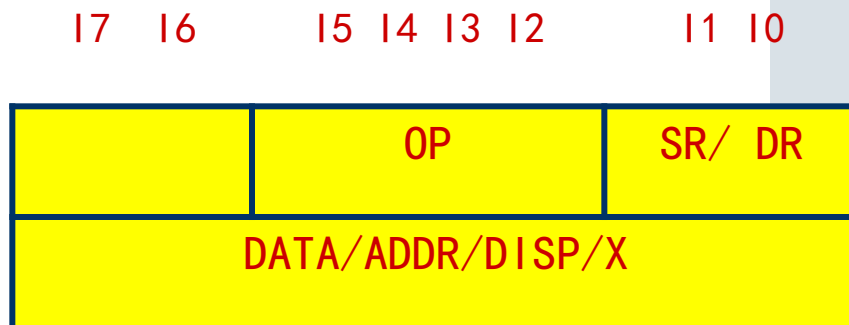
- 指令第一字：包含操作码、寻址方式、寄存器号
- 指令第二字和第三字：为寻址方式中所用到的直接/间接地址ADDR，或者是相对寻址的偏移量DISP，或者是变址寻址的形式地址X，也可以是立即数DATA

❖ 该指令格式为双存储器操作数的指令：既指令的两个操作数均在存储器内。其余同格式2。



格式4：操作码扩展指令格式

- ❖ **OP**——指令操作码，4位，是单寄存器地址指令（16条）的操作码，可通过 $I_7 I_6$ 为11方式实现散转。



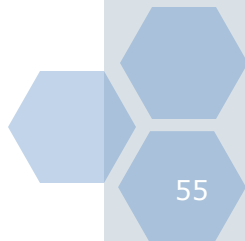
- ❖ **SR/DR**同上。
- ❖ **ADDR/DISP/X**——指令的第二个字，为寻址方式中所用到的立即数DATA、直接/间接地址ADDR，或者是相对寻址的偏移量DISP，或者是变址寻址的形式地址X。





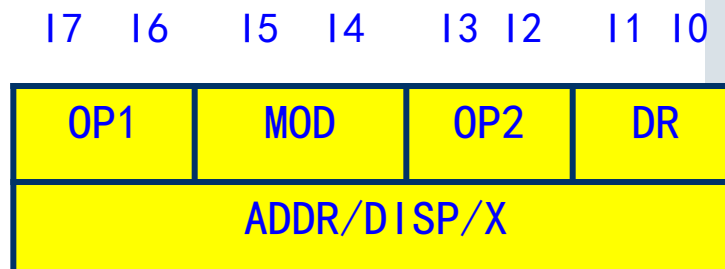
② 模型机寻址方式

- ❖ 模型机的指令系统，可实现：寄存器直接、寄存器间接、直接、间接、相对、变址、立即数**7种基本寻址方式**。
- ❖ 对于其中相对复杂的寻址方式（直接、间接、相对、变址），可以由指令中的MOD字段来定义。
- ❖ 其他相对简单的寻址方式可以直接由指令操作码指定。
- ❖ 注意：任何一种寻址方式，均可以直接由指令操作码隐含指定。
- ❖ 用户也可以根据需求，**自行设计**一些特殊的寻址方式，例如相对SR的偏移量寻址方法，即 $EA = (SR) + ADDR$ 。





带寻址方式MOD的指令格式（格式2）



❖ 对于指令格式2，假设定义：

- MOD=00：直接寻址，则有效地址EA=ADDR，操作数=（ADDR）；
- MOD=01：间接寻址，则有效地址EA=（ADDR），操作数=（（ADDR））；
- MOD=10：变址寻址，则有效地址EA=（SI）+X，操作数=（（SI）+X）；其中SI为变址寄存器，隐含为R2；
- MOD=11：相对寻址，则有效地址EA=（PC）+DISP，操作数=（（PC）+DISP）；





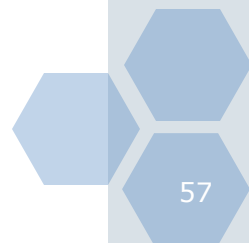
③ 模型机指令系统设计

❖ 指令设计原则

- 指令的格式必须按照上述规定的格式设计，即操作码OP、源寄存器号SR、目的寄存器号DR必须按格式规定固定长度和位置，若按照格式2设计指令，则操作码OP分为两段。
- 寻址方式的设计，可以根据需要，或由MOD字段定义，或由操作码隐含指定。
- 指令类型及功能的设计，只需满足程序设计的要求和需求即可。
- 指令操作码的分配设计，要注意规整性。

❖ 模型机指令设计举例1

❖ 模型机指令设计举例2





指令系统1举例

- ❖ 不用专门的MOD字段指出寻址方式，寻址方式由指令码定义。

1. MOV1 #DATA , Rd ;
DATA→Rd

0000	**	Rd
DATA		

2. MOV2 Rs, [Addr];
(Rs)→Addr

0001	Rs	**
Addr		

3. ADD [Addr], Rd;
(Addr)+(Rd)→Rd

0100	**	Rd
Addr		

返回

4. IN Rd,[Addr];
(Port Addr)→Rd

11	0000	Rd
Port Addr		

5. OUT [Addr],Rs;
(Rs)→ Port Addr

11	0001	Rs
Port Addr		

6. Jmp [Addr];
(Addr)→ PC

11	0010	**
Addr		

7. HLT

11	0011	**
----	------	----

8. DEC Rd; (Rd)-1 →Rd

11	0100	Rd
----	------	----







9. INC Rd; (Rd)+1 →Rd

11	0101	Rd
----	------	----

10. JZ Addr; FZ=1,则
Addr→PC; 否则结束

11	0110	**
Addr		

返回

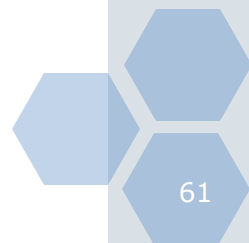
程序	功能	汇编结果 (M地址: 机器指令)
MOV1 #04H, R1	 04H→R1	
MOV2 R1, [11H]	 (R1) →11H	
IN [01H], R1	 (INPUT DEVICE) →R1	
MOV2 R1, [10H]	(R1) →10H	
IN [01H], R1	(INPUT DEVICE) → (R1)	
ADD [10H], R1	 (10H) + (R1) →R1	
OUT R1, [02H]	 (R1) →OUTPUT DEVICE	
JMP [11H]	 (11H) →PC	





指令系统2举例

- ❖ 共有12条指令，分为：
- ❖ 5条双寄存器算术逻辑运算类指令
- ❖ 3条单寄存器指令
- ❖ 4条存储器访问类指令
- ❖ 2条I/O指令
- ❖ 2条过程控制类指令
- ❖ 程序设计





5条双寄存器算术逻辑运算类指令

❖ 格式：

17 16 15 14 13 12 11 10

❖ 操作码及功能：

OP	SR	DR
----	----	----

助记符	操作码OP	功能
MOV DR, SR	0000	$(SR) \rightarrow DR$
ADD DR, SR	0001	$(SR) + (DR) \rightarrow DR$
SUB DR, SR	0010	$(DR) - (SR) \rightarrow DR$
AND DR, SR	0011	$(SR) \wedge (DR) \rightarrow DR$
RRC DR, SR	0100	(SR) 进行带进位循环右移 $\rightarrow DR$





3条单寄存器指令

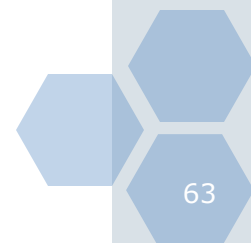
❖ 格式：

❖ 操作码及功能：

17 16 15 14 13 12 11 10

11	OP	DR/SR
----	----	-------

助记符	操作码OP	功能
INC DR	1101	$(DR)+1 \rightarrow DR$
DEC DR	1110	$(DR)-1 \rightarrow DR$
CLR DR	1111	$0 \rightarrow DR$



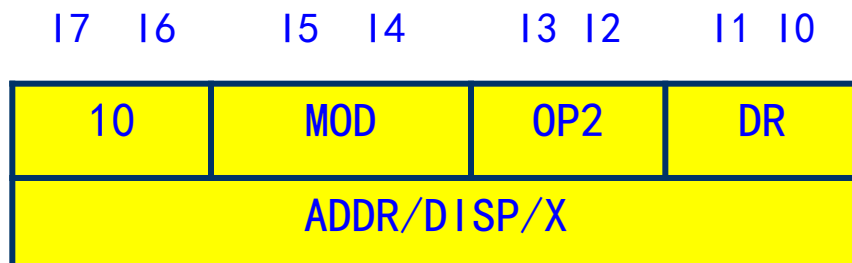


4条存储器访问类指令

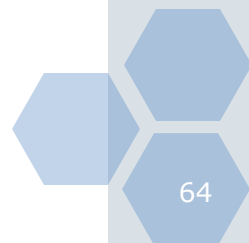
❖ 格式:

❖ 操作码及功能:

❖ SI隐含为R2



MOD	寻址方式	EA	助记符	OP ₂	功能
00	直接寻址	ADDR	LDA	00	[EA]→DR
01	间接寻址	[ADDR]	STA	01	(DR) →EA
10	变址寻址	(SI) +X	JMP	10	EA→PC
11	相对寻址	(PC) +DISP	JZC	11	若FC+FZ=1, 则EA→ PC, 否则, 结束指令





2条I/O指令

❖ 格式：

❖ 操作码及功能：

17 16 15 14 13 12 11 10



助记符	操作码OP	功能
IN DR, [PORTAR]	0000	(PORTAR)→DR
OUT DR, [PORTAR]	0001	(DR)→ PORTAR





2条过程控制类指令

❖ 格式:

❖ CALL ADDR

❖ $(SP) - 1 \rightarrow SP,$

❖ $(PC) \rightarrow (SP),$
 $ADDR \rightarrow PC$

17 16 15 14 13 12 11 10

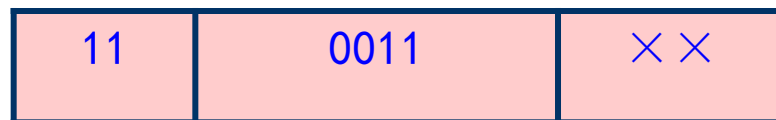


❖ 格式:

❖ RET

❖ $((SP)) \rightarrow PC$
 $(SP) + 1 \rightarrow SP,$

17 16 15 14 13 12 11 10





地址	机器码	助记符	备注
00H	11111100	CLR R ₀	R ₀ 当作累加器。
01H	10000010 00101011	LDA R ₂ , [2BH] 直接地址2BH	R ₂ 当作计数器/变址寄存器;其初值0AH存放在单元2BH中。
02H			
03H	10100001 00011111	L1: LDA R ₁ , [SI+1FH]	取出需要累加的数据;采用变址寻址方式;第1次地址=29H。
04H			
05H	00010100	ADD R ₀ , R ₁	累加。
06H	11111010	DEC R ₂	计数器递减; 并影响标志FZ、FC
07H	10111100 00000101	JZC L2 相对位移05H	FC+FZ=1循环, FC+FZ=0 (无借位不为0) 退出循环。
08H			
09H	10000100 00101010	STA [2AH],R ₀ 直接地址2AH	存储累加和; 采用直接寻址方式。
0AH			
0BH	10001000 00000011	JMP L1 直接地址03H	无条件转移; 采用直接寻址方式。
0CH			
0DH 0EH	10001000 00000000	L2: JMP [00H]	转移至00H单元。



地址	机器码	助记符	备注
...		
20H		N1	数据1
21H		N2	数据2
...		
29H		N10	数据10
2AH		$N1+N2+.....+N10$	累加和
2BH	00001010B	计数值0AH	

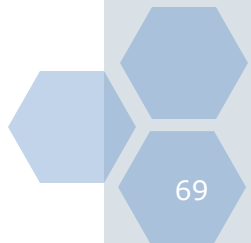




2、MIPS指令系统

What's MIPS?

- ❖ MIPS (Microprocessor without Intellocked Pipeline Stages, 无内部互锁流水级的微处理器) 是一种采用精简指令集 (RISC) 架构的处理器。
- ❖ MIPS指令系统有多个版本存在, 包括MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS16, MIPS32和MIPS64。目前的版本是MIPS32 (对于32位实现) 和MIPS64 (64位实现) 。

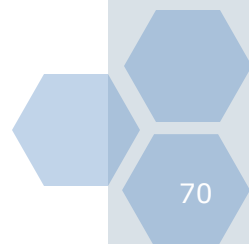




2、MIPS指令系统

MIPS指令系统有以下特点：

- 1)简单的load/store结构
- 2)易于流水线CPU设计
- 3)易于编译器开发
- 4)MIPS指令的寻址方式非常简单，每条指令的操作也非常简单 >

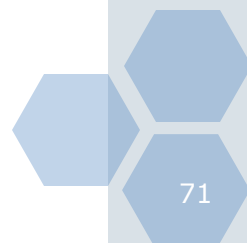




2、MIPS指令系统

MIPS体系结构提供的数据类型有：

- ❖ 位 (bit) ;
- ❖ 字节 (Byte) : 长度是8bit;
- ❖ 半字 (Half Word) : 长度是16bit;
- ❖ 字 (Word) : 长度是32bit;
- ❖ 双字 (Double Word) : 长度是64bit;
- ❖ 此外, 还有32位单精度浮点数、64位双精度浮点数等。



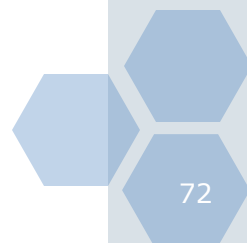


2、MIPS指令系统

MIPS32的寄存器：

MIPS寄存器分为3类：通用寄存器、专用寄存器和浮点寄存器。

- ❖ **MIPS32定义了32个通用寄存器，在汇编语言中，使用\$0、\$1.....\$31表示，或用寄存器的名字来表示，如\$sp、\$tl、\$ra等，都是32位。其中\$0一般用做常量0。**





2、MIPS指令系统

MIPS32 32个通用寄存器的序号、名称和用途

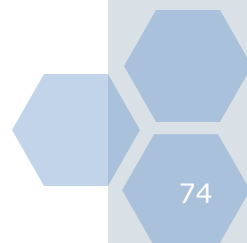
序号	寄存器名称	用途
\$0	\$zero	常量0 (constant value 0)
\$1	\$at	保留给汇编器 (Reserved for assembler)
\$2-\$3	\$v0-\$v1	函数调用返回值 (values for results and expression evaluation)
\$4-\$7	\$a0-\$a3	函数调用参数 (arguments)
\$8-\$15	\$t0-\$t7	保存暂时变量
\$16-\$23	\$s0-\$s7	保存函数寄存器变量 (如果用, 需要SAVE/RESTORE的)
\$24-\$25	\$t8-\$t9	保存暂时变量
\$26-\$27	\$k0-\$k1	保留给操作系统内核
\$28	\$gp	全局指针 (Global Pointer)
\$29	\$sp	堆栈指针 (Stack Pointer)
\$30	\$fp (\$s8)	帧指针 (Frame Pointer)
\$31	\$ra	函数返回地址 (return address)



2、MIPS指令系统

MIPS32的专用寄存器：

- ❖ **两个32位的乘商寄存器hi和lo**，hi和lo一起可实现64位寄存器。用来保存乘法和除法运算时的计算结果。
- ❖ **程序计数器（PC）**：用于指出下一条指令的地址。因为MIPS存储器按字节编制，而每条指令的固定长度是32位，因此MIPS取指令后PC值自动加4，指向存储器中的下一条指令。



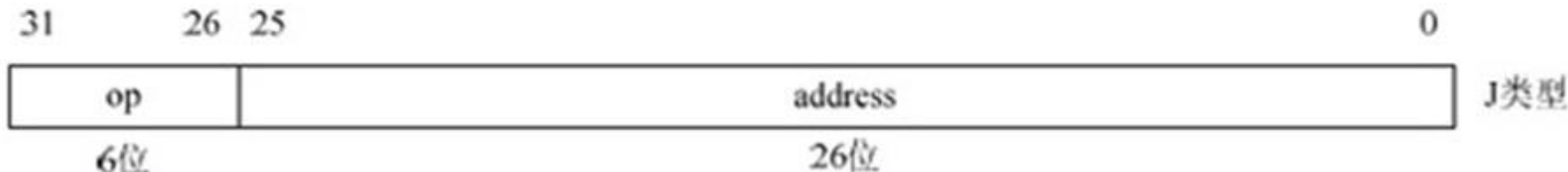
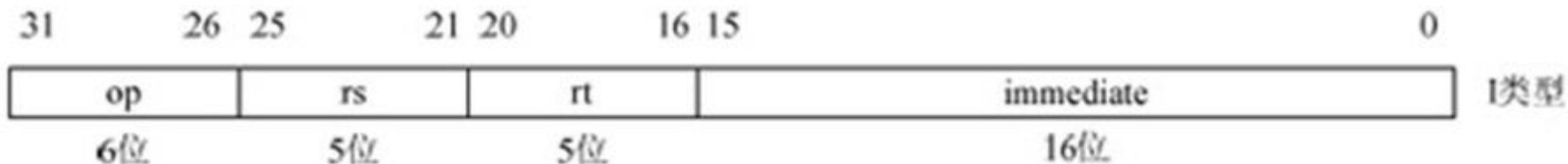
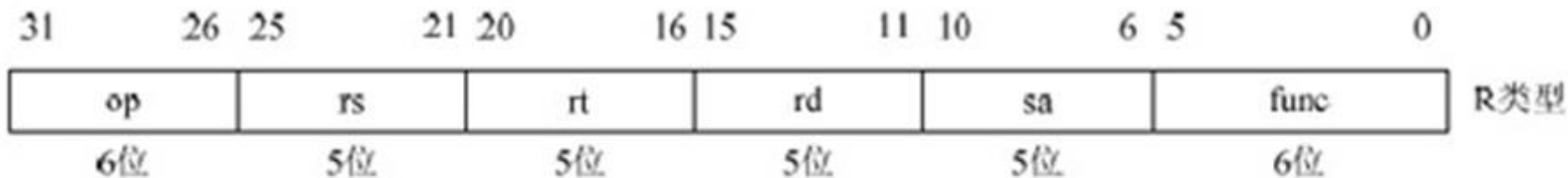


2、MIPS指令系统

MIPS32指令格式

MIPS32的每条指令长度固定是32位。其指令格式只有三种类型：

- 1)**R型指令**：也叫RR型指令。从寄存器堆中取出源操作数，计算结果（目的操作数）写回寄存器堆。
- 2)**I型指令**：也称为R-I型指令，是立即数型指令。用一个16位的立即数作为一个源操作数。
- 3)**J型指令**：是无条件转移指令。用一个26位的立即数作为跳转的目标地址。

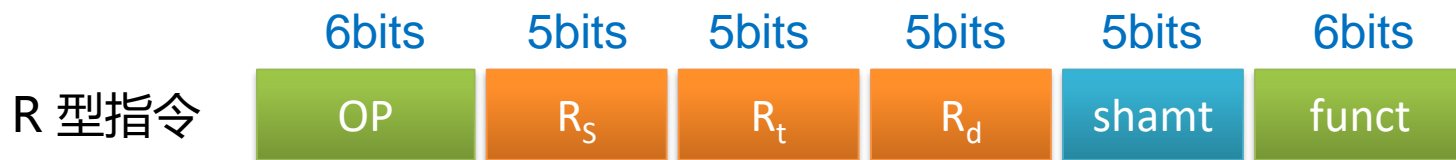


MIPS指令格式

R型和I型指令为**三地址指令**，J型指令为**单地址指令**。

指令类型	指令格式（共32位）						备注
	31-26	25-21	20-16	15-11	10-6	5-0	
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)	算术指令
I	opcode (6)	rs (5)	rt (5)	offset/immediate (16)			数据传输、分支、立即数指令
J	opcode (6)	address (26)					跳转指令

32位定长MIPS指令格式（R型指令）



- OP: 指令的基本操作---操作码
- R_s : 第一个源操作数寄存器
- R_t : 第二个源操作寄存器
- R_d : 存放结果的目的地操作寄存器
- Shamt: 偏移量, 用于移位指令
- Funct: 功能码, 对操作码进行补充

MIPS指令格式 (R型指令)

指令	格式	6bits	5bits	5bits	5bits	5bits	6bits
		OP	rs	rt	rd	shamt	funct
add	R	0	Reg	Reg	Reg	0	32 ₁₀
sub减	R	0	Reg	Reg	Reg	0	34 ₁₀
and	R	0	Reg	Reg	Reg	0	36
or	R	0	Reg	Reg	Reg	0	37
nor	R	0	Reg	Reg	Reg	0	39
sll	R	0	Reg	Reg	Reg		0
srl	R	0	Reg	Reg	Reg		2
jr	R	0	REG	0	0	0	8

例：

add	R	0	18	19	17	0	32
-----	---	---	----	----	----	---	----

■ add \$s1,\$s2,\$s3 # machine code 0x2538820

MIPS指令格式 (I、J型指令)

		6bits	5bits	5bits	5bits	5bits	6bits
指令	格式	OP	rs	rt	rd	shamt	funct
add	R	0	Reg	Reg	Reg	0	32 ₁₀
addi	I	8	Reg	Reg	16bits 立即数		
lw	I	35	Reg	Reg	16bits 立即数		
sw	I	43	Reg	Reg	16bits 立即数		
andi	I	12	Reg	Reg	16bits 立即数		
ori	I	13	Reg	Reg	16bits 立即数		
beq	I	4	Reg	Reg	16bits 立即数 (相对寻址)		
bne	I	5	Reg	Reg	16bits 立即数 (相对寻址)		
j	J	2	26bit 立即数(伪直接寻址)				
jal	J	3	26bit 立即数(伪直接寻址)				

MIPS寻址方式(共5种)

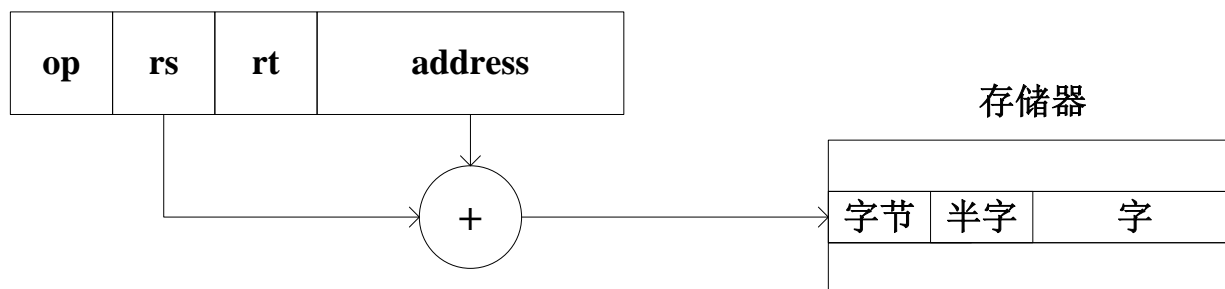
- (1) 寄存器寻址：操作数是寄存器。



例如: **or \$s1,\$s2,\$s3** # **\$s1=\$s2 | \$s3**

- (2) 基址寻址或相对寄存器寻址或偏移量寻址:

操作数在存储器中，存储器地址是指令中基址寄存器和常数之和



例如: **lw \$t0,addr(\$t3)**

如果addr=0,也可以写成 **lw \$t0,(\$t3)**

MIPS寻址方式(共5种)

- (3) 立即数寻址：操作数是指令给出的常数。



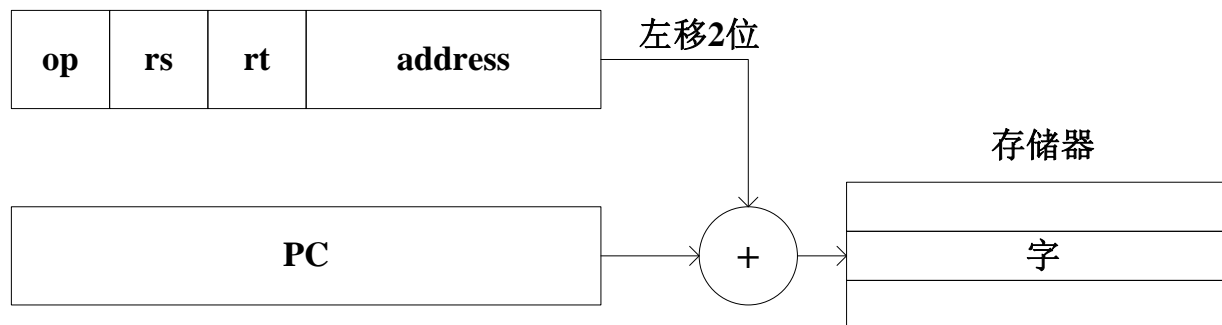
例如：sra \$s1, \$s2, 10 # \$s2算术右移10位→\$s1

addi \$s1, \$s2, 100 # \$s1 = \$s2 + 100

MIPS寻址方式

■ (4) PC相对寻址或相对转移寻址或相对寻址

□ 目标地址：16位偏移地址经符号扩展成32位后左移两位+PC



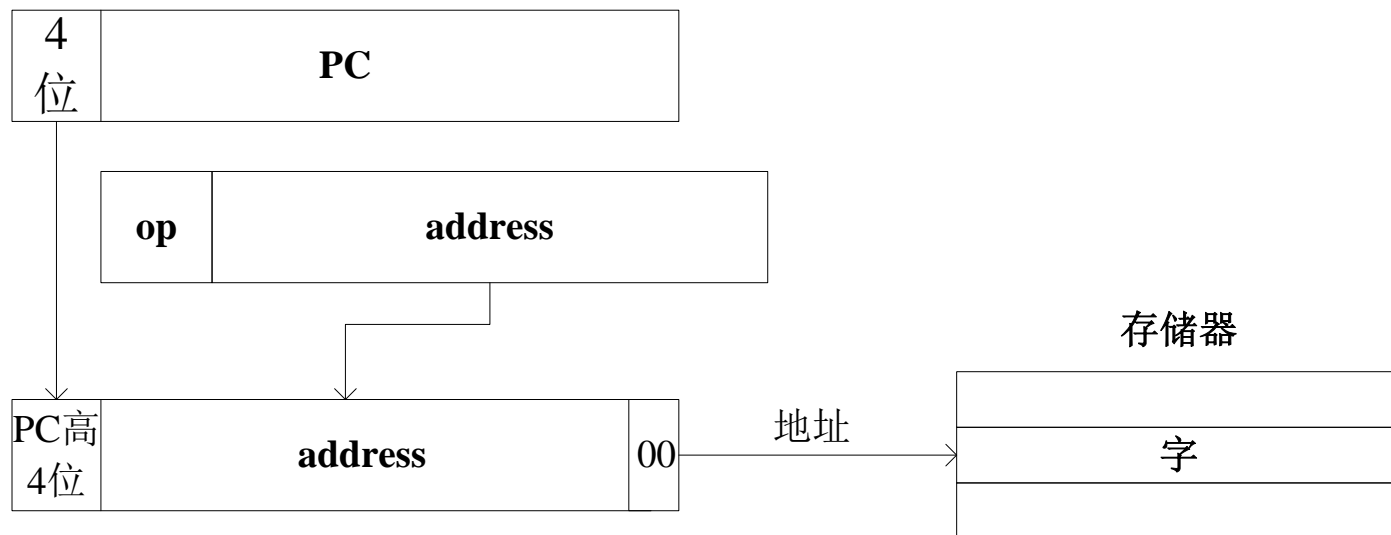
例如: `bne $s1, $s2, 25`

if `$s1 ≠ $s2` then goto `25*4 + 当前PC值` (即**bne**指令的PC+4)

MIPS寻址方式

■ (5) 伪直接寻址或页面寻址

- 26位偏移地址高位拼上PC高4位→30位字地址，30位字地址末位拼上两位0→32位字节地址



例如: **J 1000 #跳转到PC | 1000**

MIPS寻址方式

■ 从R、I、J三种指令类型的角度讨论寻址方式

□ R型指令：三个操作数均是寄存器寻址方式。

例如：

or \$s1,\$s2,\$s3 # \$s1=\$s2 | \$s3

MIPS寻址方式

□ **I型指令**：有寄存器寻址、立即数寻址、相对转移寻址或相对寻址、基址寻址或相对寄存器寻址或偏移量寻址四种寻址方式。

◆ ①若I型指令是**双目运算指令**，则**rs寄存器和立即数**分别作为源操作数，**rt寄存器**是目的操作数；

例如：

`addi $sp,$sp,4 # $sp=$sp+4, 立即数寻址`

MIPS寻址方式

- ◆②若I型指令是访存指令load/store，则存储器地址由rs寄存器加上符号扩展的立即数得到，load指令将存储内容读出到rt寄存器中，store指令将rt寄存器写入存储器中。

例如：

lw \$t0,32(\$s1) # 假如s1里是数组A的起始地址，存储器按字节编址，因此\$*s1*+32指向A[8]，连续读4个字节，读出A[8]的值置入t0寄存器。这是寄存器相对寻址（基址寻址）方式，s1是基地址寄存器，用来装载数组的起始地址，常数32是偏移量，表示数组元素的下标值。

MIPS寻址方式

- ③若I型指令是**条件转移指令**，则对rs和rt寄存器的内容进行指定运算，根据结果决定是否跳转。**转移目标地址**等于当前PC值加上符号扩展后的立即数，这是**相对寻址或相对转移寻址**。偏移量可正可负。在对数组和堆栈寻址时，相对转移寻址比其他寻址方式更具优势。
- 例如：
- bne \$s0,\$s1,L0 #如果#\$s0≠s1，则跳转到L0处。这是相对转移寻址。PC=(PC)+符号扩展成32位的偏移量，请注意：与偏移量相加的PC值是bne指令的后一条指令的PC值，即bne指令的PC值+4。

MIPS寻址方式

- J型指令：只有页面寻址或伪直接寻址这一种寻址方式。因为J型指令都是无条件转移指令，而MIPS系统采用32位定长指令，每条指令占4个存储单元，指令地址总是4的倍数。所以只要将当前PC值的高4位拼上26位直接地址，末尾两位置0，即可得到32位的目标转移地址。



MIPS32指令分类：

❖ (1) 空操作指令：nop, ssnop。

❖ (2) 数据传送指令：

- 寄存器间数据传送指令：

move, movf, movt, movn, movz。

- 常数加载指令：dla, la, dli、li, lui。

❖ (3) 算术 / 逻辑运算指令：

- 加法指令和减法指令：add、addi、dadd、daddi、addu、addiu、daddu、daddiu、dsub、sub、subu；

- 绝对值：abs, dabs；

- 取相反数：dneg、neg、negu；

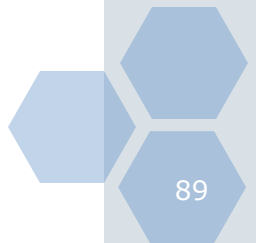
- 逐位逻辑操作指令：and、andi、or、ori、xor、nor；

- 循环移位指令：drol、rol、ror；

- 移位：sll、srl、sra。

- 整数乘法、除法和求余数：div、mul、rem等等。

- 整数乘加（累加）：mad等。





MIPS32指令分类：

❖ (4) 访存指令：

- 加载和存储：
lb、ld、ldl、ldr、sdl、sdr、lh、lhu、ll、sc、pref、sb等操作。
- 浮点加载和存储：
l.d、l.s、s.d、s.s等

❖ (5) 程序控制类指令：

- 跳转、子程序调用和分支

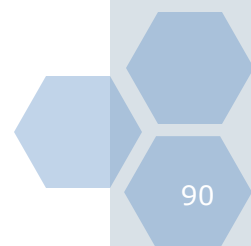
❖ (6) 断点及陷进指令

❖ (7) 协处理器0的功能指令

❖ (8) 浮点操作指令：

- 支持IEEE754的单精度和双精度格式

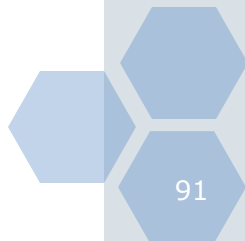
❖ (9) 用户模式下对“底层”硬件的有限访问指令





指令机器码的编码、取指与译码

- ❖ **指令的编码**：就是实现汇编语言到二进制机器码的过程，由汇编器实现（编译器是将C语言转为汇编语言）。MIPS32 CPU都是32位字长，因此指令编码是32比特。
- ❖ **取指**：就是根据当前程序计数寄存器PC的值从存储器中取出二进制机器码指令。
- ❖ **指令的译码**：取指电路将从存储器中取出的二进制机器码指令送给译码器电路进行译码。译码器的主要功能就是根据指令的编码格式来分析二进制机器码是什么指令？从而进行相应的操作。





本章小结

- ❖ 机器指令由操作码字段和地址码字段组成。扩展操作码技术实现指令优化，但也增加了硬件设计难度。
- ❖ 指令的寻址方式包括指令寻址和数据寻址，指令寻址主要是顺序和跳跃两种方式；数据寻址有许多种寻址方式，其目的是获得本条指令执行所需要的操作数。
- ❖ 根据指令的功能，可将指令分类为数据传送、算术逻辑运算类、移位操作类、转移类、堆栈操作类、输入输出类等指令。
- ❖ 指令系统的设计应满足完备性、有效性、规整性、兼容性四个方面的要求。CISC指令系统庞大的指令集及其存在问题，RISC指令以它简洁、高效等特点而得到快速地发展。
- ❖ 通过两种指令系统例子，使读者结合实际理解和掌握指令系统的设计方法。





The End!

