



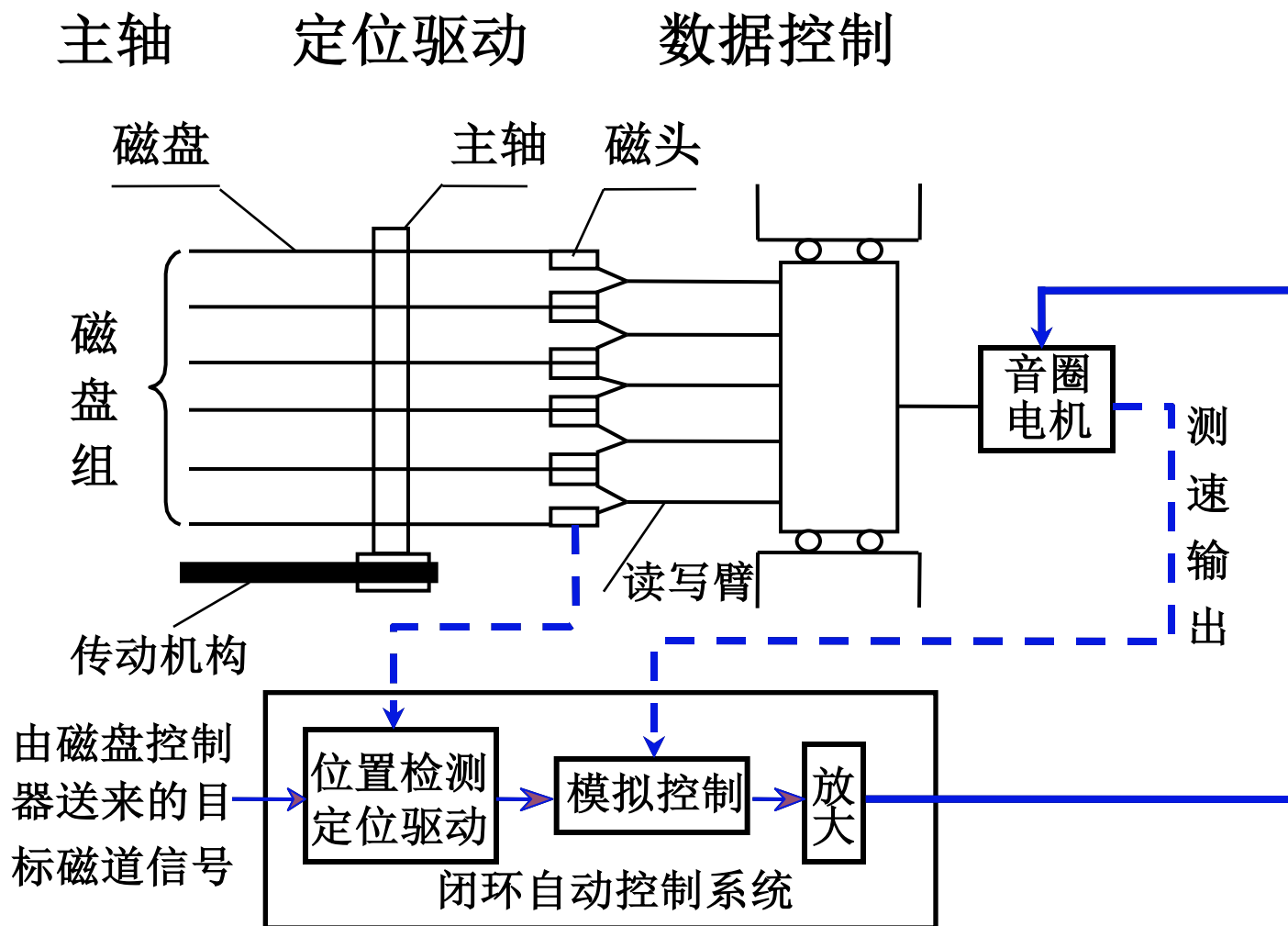
# 计算机组成原理

## 第 12讲

左德承

哈尔滨工业大学计算学部  
容错与移动计算研究中心

# (1) 磁盘驱动器



## (2) 磁盘控制器

- 接收主机发来的命令，转换成磁盘驱动器的控制命令
- 实现主机和驱动器之间的数据格式转换
- 控制磁盘驱动器读写

磁盘控制器 是

主机与磁盘驱动器之间的 接口 { 对主机 通过总线  
对硬盘 (设备)

## (3) 盘片

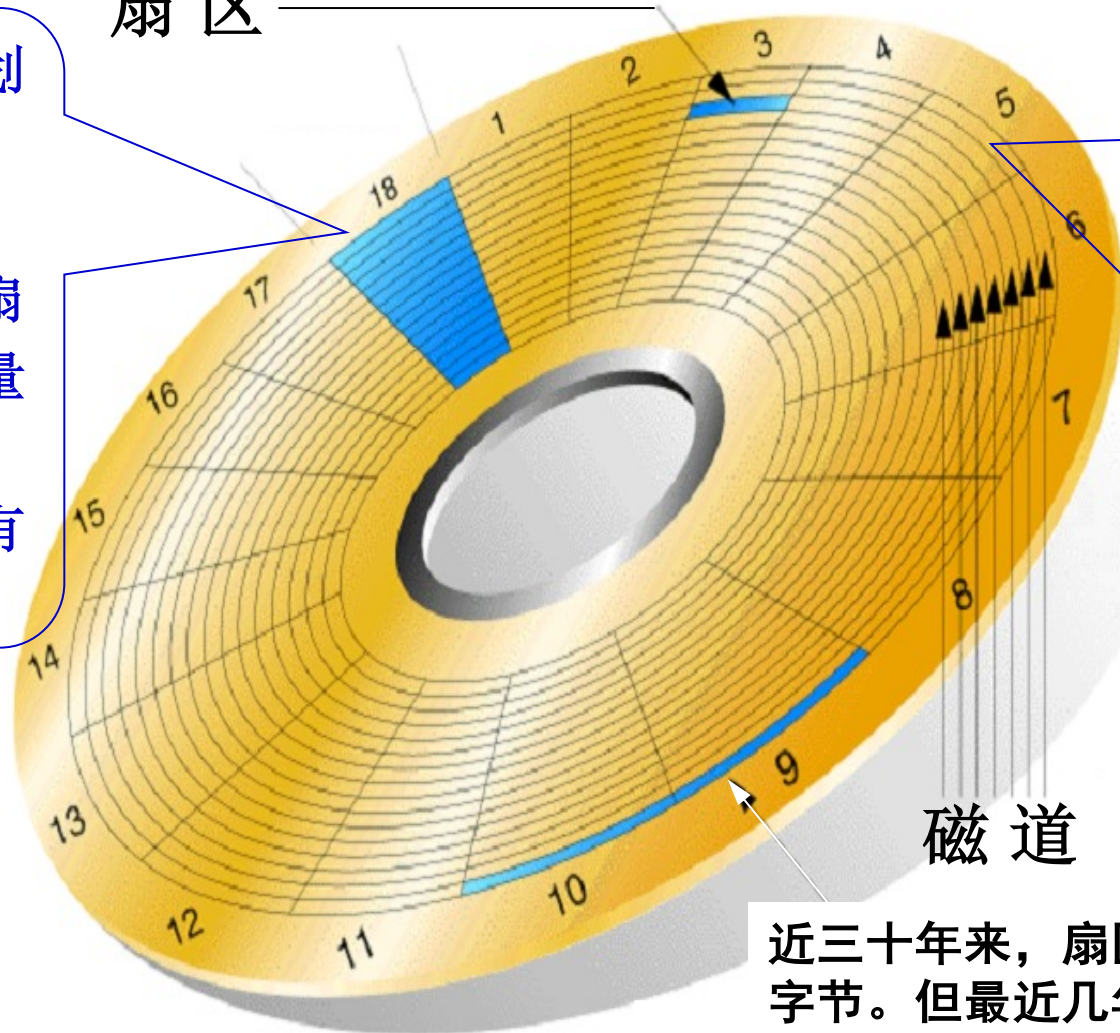
由硬质铝合金材料制成

# 磁盘的磁道和扇区

## 4.5

每个磁道被划分为若干段（段又叫扇区），每个扇区的存储容量为**512**字节。每个扇区都有一个编号

扇区



磁盘表面被分为许多同心圆，每个同心圆称为一个磁道。每个磁道都有一个编号，最外面的是**0**磁道

磁道

近三十年来，扇区大小一直是512字节。但最近几年正迁移到更大、更高效的4096字节扇区，通常称为4K扇区。国际硬盘设备与材料协会（IDEMA）将之称为高级格式化。

## 2. 磁表面存储器的技术指标

(1) 记录密度          道密度  $D_t$     位密度  $D_b$

(2) 存储容量           $C = n \times k \times s$

(3) 平均寻址时间    寻道时间 + 等待时间

辅存的速度 { 寻址时间  
                  磁头读写时间

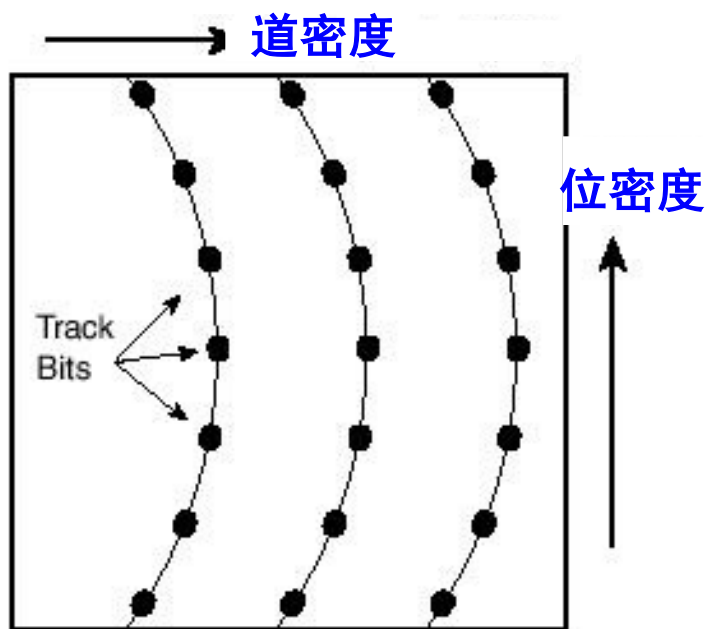
(4) 数据传输率       $D_r = D_b \times V$

(5) 误码率          出错信息位数与读出信息的总位数之比

# 如何增大磁盘片的容量？

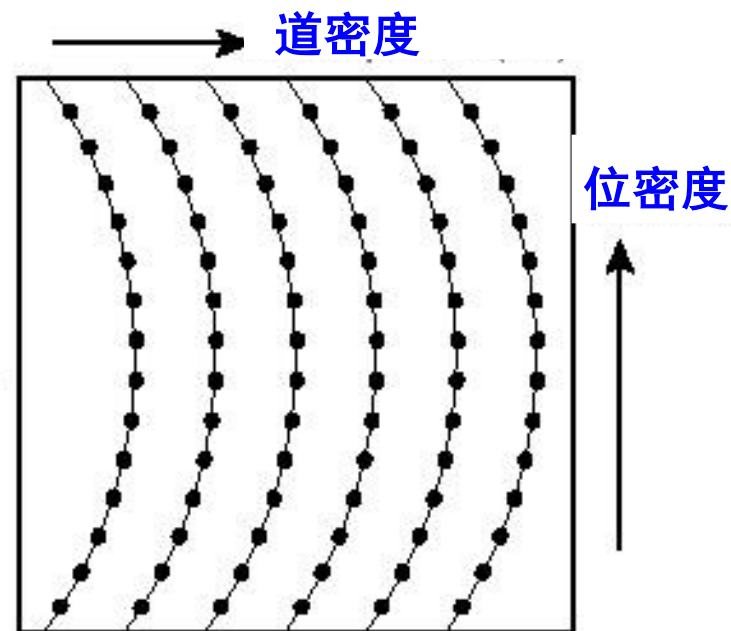
4.5

- 提高盘片上的信息记录密度！
  - 增加磁道数目——提高磁道密度
  - 增加扇区数目——提高位密度，并采用可变扇区数



低密度存储示意图

早期磁盘所有磁道上的扇区数相同，所以位数相同，内道上的位密度比外道位密度高

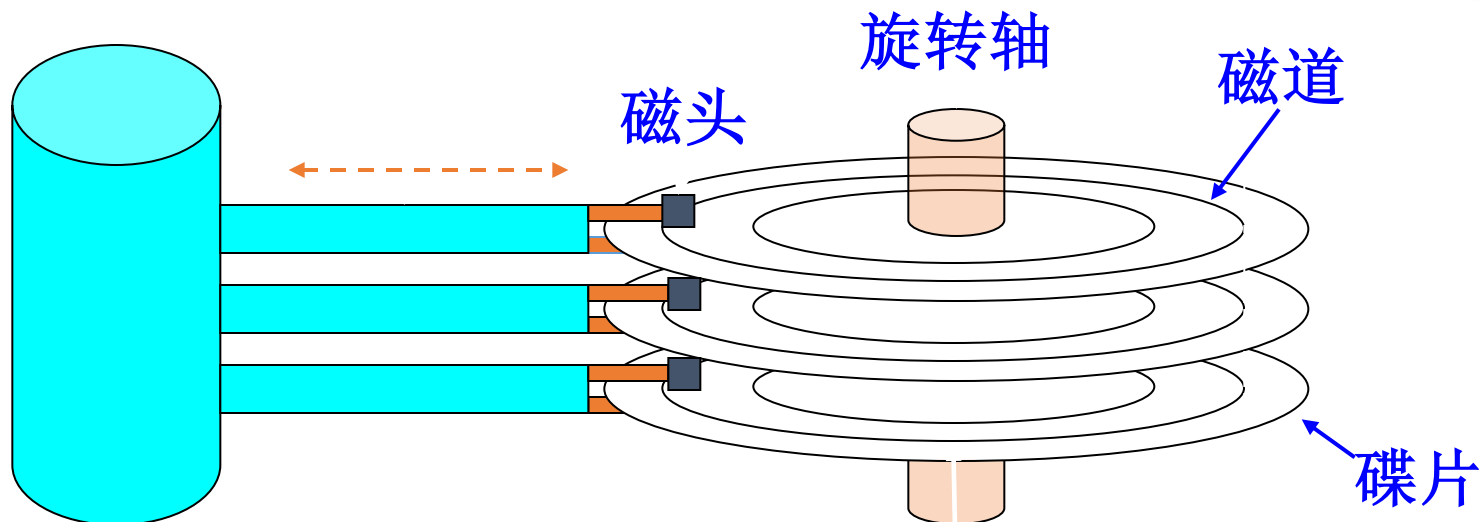


高密度存储示意图

现代磁盘磁道上的位密度相同，所以，外道上的扇区数比内道上扇区数多，使整个磁盘的容量提高

# 平均存取时间

## 4.5



硬盘的操作流程如下：

所有磁头同步寻道（由柱面号控制）→ 选择磁头（由磁头号控制）→  
被选中磁头等待扇区到达磁头下方（由扇区号控制）→ 读写该扇区中数据

- 磁盘上的信息以扇区为单位进行读写，平均存取时间为：

$T = \text{平均寻道时间} + \text{平均旋转等待时间} + \text{数据传输时间（忽略不计）}$

- **平均寻道时间**——磁头寻找到指定磁道所需平均时间 (大约5ms)
- **平均旋转等待时间**——指定扇区旋转到磁头下方所需平均时间 (大约4~6ms) ( 转速：  
4200 / 5400 / 7200 / 10000rpm )
- **数据传输时间**——( 大约0.01ms / 扇区 )

# 磁盘响应时间计算举例

4.5

- 假定每个扇区512字节，磁盘转速为5400 RPM，声称寻道时间（最大寻道时间的一半）为12 ms, 数据传输率为4 MB/s, 磁盘控制器开销为1 ms, 不考虑排队时间，则磁盘响应时间为多少？

$$\begin{aligned}\text{Disk Response Time} &= \text{Seek time} + \text{Rotational Latency} + \text{Transfer time} \\ &\quad + \text{Controller Time} + \text{Queuing Delay} \\ &= 12 \text{ ms} + 0.5 / 5400 \text{ RPM} + 0.5 \text{ KB} / 4 \text{ MB/s} + 1 \text{ ms} + 0 \\ &= 12 \text{ ms} + 0.5 / 90 \text{ RPS} + 0.125 / 1024 \text{ s} + 1 \text{ ms} + 0 \\ &= 12 \text{ ms} + 5.5 \text{ ms} + 0.1 \text{ ms} + 1 \text{ ms} + 0 \text{ ms} \\ &= 18.6 \text{ ms}\end{aligned}$$

如果实际的寻道时间只有1/3的话，则总时间变为10.6ms，这样旋转等待时间就占了近50%！

$12/3 + 5.5 + 0.1 + 1 = 10.6 \text{ ms}$  所以，磁盘转速非常重要！

为什么实际的寻道时间可能只有1/3？

访问局部性使得每次磁盘访问大多在局部几个磁道，实际寻道时间变少！



# 第 7 章 指令系统

## 7.1 机器指令

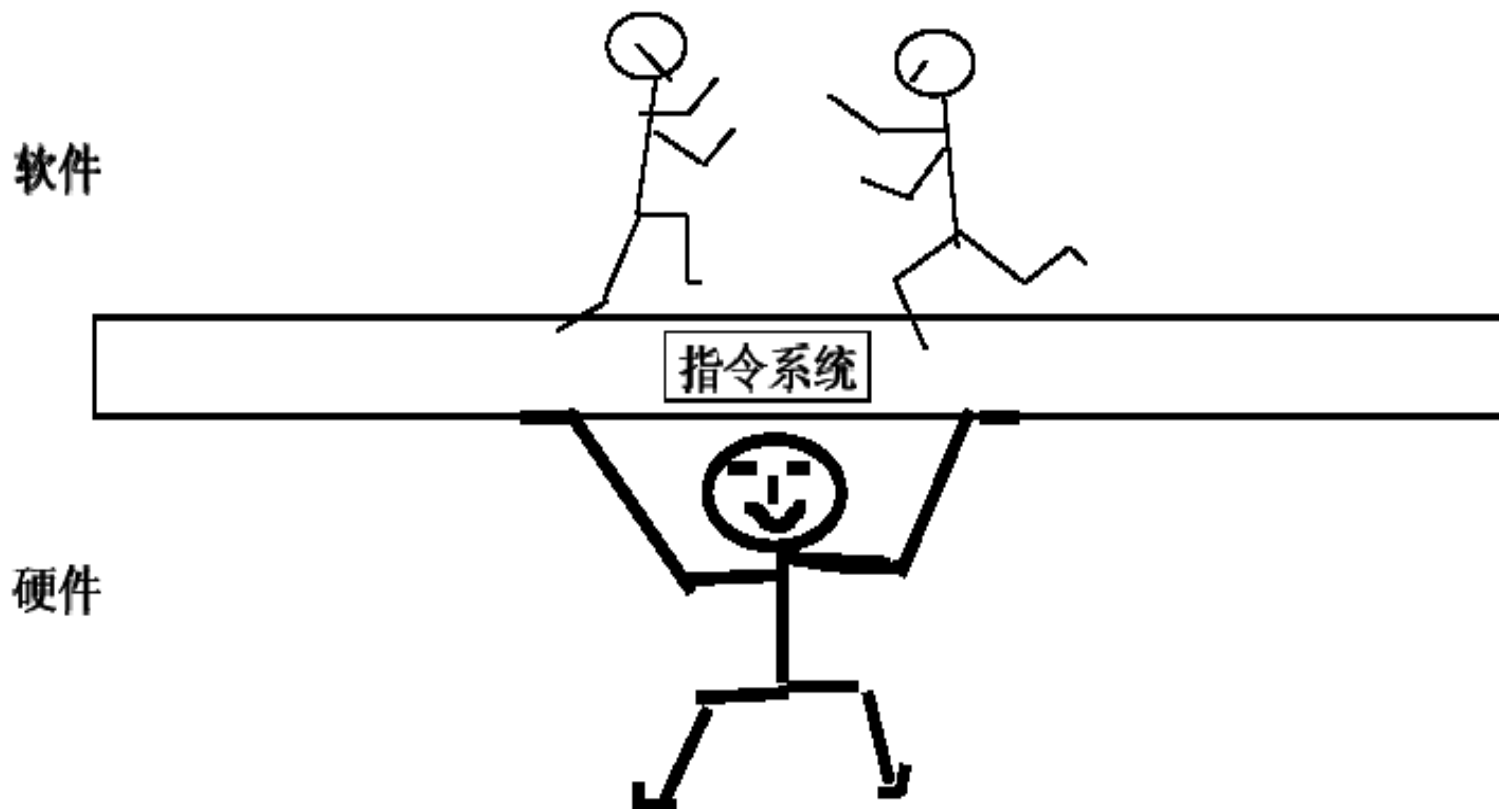
## 7.2 操作数类型和操作类型

## 7.3 寻址方式

## 7.4 指令格式举例








## 7.5 RISC 技术

# 指令系统在计算机中的地位



# 指令集体系结构 Instruction Set Architecture (ISA)

◆不同类型的CPU执行不同指令集，是设计CPU的依据

-  1970 DEC **PDP-11** 1992 **ALPHA(64位)**
-  1978 **x86**, 2001 **IA64**
-  1980 **PowerPC**
-  1981 **MIPS**
-  1985 **SPARC**
-  1991 **arm**
-  2016 **RISC-V**

2020  
LoongArch



## ◆指令集优劣

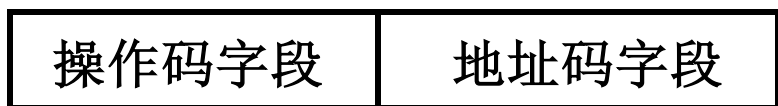
- 方便硬件实现（高性能、低功耗）
- 方便编译优化
- OS，虚拟机开发

# 硬件设计四原则

- 简单性来自规则性
  - ✓ Simplicity favors regularity
  - ✓ 指令越规整设计越简单
- 越小越快
  - ✓ Smaller is faster
  - ✓ 面积小，传播路径小，门延迟少
- 加快经常性事件
  - Make the common case fast
- 好的设计需要适度的折衷
  - Good design demands good compromises

# 7.1 机器指令

## 一、指令的一般格式



### 1. 操作码 反映机器做什么操作

#### (1) 长度固定

用于指令字长较长的情况，**RISC**

如 **IBM 370** 操作码 8 位

#### (2) 长度可变

操作码分散在指令字的不同字段中

# (3) 扩展操作码技术

7.1

操作码的位数随地址数的减少而增加

	OP	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	
4 位操作码	0000 0001 ⋮ 1110	A <sub>1</sub> A <sub>1</sub> ⋮ A <sub>1</sub>	A <sub>2</sub> A <sub>2</sub> ⋮ A <sub>2</sub>	A <sub>3</sub> A <sub>3</sub> ⋮ A <sub>3</sub>	最多15条三地址指令
8 位操作码	1111 1111 ⋮ 1111	0000 0001 ⋮ 1110	A <sub>2</sub> A <sub>2</sub> ⋮ A <sub>2</sub>	A <sub>3</sub> A <sub>3</sub> ⋮ A <sub>3</sub>	最多15条二地址指令
12 位操作码	1111 1111 ⋮ 1111	1111 1111 ⋮ 1111	0000 0001 ⋮ 1110	A <sub>3</sub> A <sub>3</sub> ⋮ A <sub>3</sub>	最多15条一地址指令
16 位操作码	1111 1111 ⋮ 1111	1111 1111 ⋮ 1111	1111 1111 ⋮ 1111	0000 0001 ⋮ 1111	16条零地址指令

# (3) 扩展操作码技术

7.1

操作码的位数随地址数的减少而增加

4 位操作码

OP	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>
0000	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>
0001	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>
⋮	⋮	⋮	⋮
1110	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>

三地址指令操作码  
每减少一种最多可多构成  
2<sup>4</sup> 种二地址指令

8 位操作码

1111	0000	A <sub>2</sub>	A <sub>3</sub>
1111	0001	A <sub>2</sub>	A <sub>3</sub>
⋮	⋮	⋮	⋮
1111	1110	A <sub>2</sub>	A <sub>3</sub>

二地址指令操作码  
每减少一种最多可多  
构成2<sup>4</sup> 种一地址指令

12 位操作码

1111	1111	0000	A <sub>3</sub>
1111	1111	0001	A <sub>3</sub>
⋮	⋮	⋮	⋮
1111	1111	1110	A <sub>3</sub>

16 位操作码

1111	1111	1111	0000
1111	1111	1111	0001
⋮	⋮	⋮	⋮
1111	1111	1111	1111

# 扩展指令举例

- 设某指令系统指令字长16位，每个地址码为6位。若要求设计二地址指令15条、一地址指令34条，问最多还可设计多少条零地址指令？

双操作数15



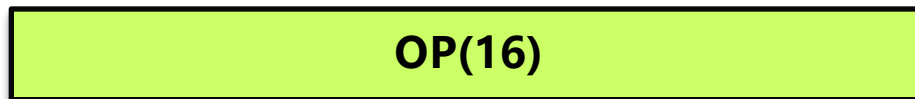
$$2^4$$

单操作数34



$$(2^4 - 15) * 2^6$$

无操作数？



$$((2^4 - 15) * 2^6 - 34) * 2^6$$



# 指令地址码

三地址指令



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

二地址指令



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

单地址指令



$(AC) \text{ OP } (A1) \rightarrow AC$

零地址指令



如停机、空操作、  
开关中断等

## 二、指令字长

7.1

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

字长与机器字的长度有关：单字长，双字长，半字长

- 指令字越长，地址码长度越长，可直接寻址空间越大
- 指令字越长，占用空间越大，取指令越慢
- 定长指令：结构简单，控制线路简单，MIPS指令
- 变长指令：结构灵活，充分利用指令长度，控制复杂，X86指令

## 7.2 操作数类型和操作种类

### 一、操作数类型

地址	无符号整数
数字	定点数、浮点数、十进制数
字符	ASCII
逻辑数	逻辑运算

# 二、操作类型

## 7.2

### (1) x86指令分类

#### ■ 数据传送类

- 取数 `MOV AX, TEMP`
- 存数 `MOV TEMP, AX`
- 传送 `MOV AX, CX`

#### ■ 算术运算类

- 定点`+`, `-`, `×`, `÷` 等
- 浮点`+`, `-`, `×`, `÷` 求反, 求补等

#### ■ 逻辑运算类

- `NOT`, `AND`, `OR`, `XOR`, `TEST`

#### ■ 程序控制类

- 无条件转移 `JMP` 条件转移 `C`, `Z`, `N`, `P`, `V`
- 转子程序 `JSR` 子程序返回 `RET` 中断返回 `IRET`

#### ■ 输入/输出类

- `IN AX, n` `OUT n, AX`

#### ■ 其他类

- 标志操作: `CLC` (clear carry flag)
- `CLI` (clear interrupt enable flag)
- `HLT`, `WAIT`

## (2) MIPS指令分类

### ■ 运算指令

- 算术: add,addi,addu,addiu, sub,subu, mult,multu,div,divu, slt,slti,sltiu
- 逻辑: and,andi,or, ori,xor,xori,nor
- 移位指令: sll,sllv,srl, srlv,sra,srav

### ■ 分支指令

- beq, bne, blez( $\leq 0$ ), bgez( $\geq 0$ ), bltz( $< 0$ ), bgtz( $> 0$ ), jal, j, jr

### ■ 访存指令

- lw,lh,lb,sw,sh,sb

### ■ 系统指令

- syscall, break, sync, cache

## 7.3 寻址方式

寻址方式 确定 本条指令 的 操作数地址  
下一条 欲执行 指令 的 指令地址

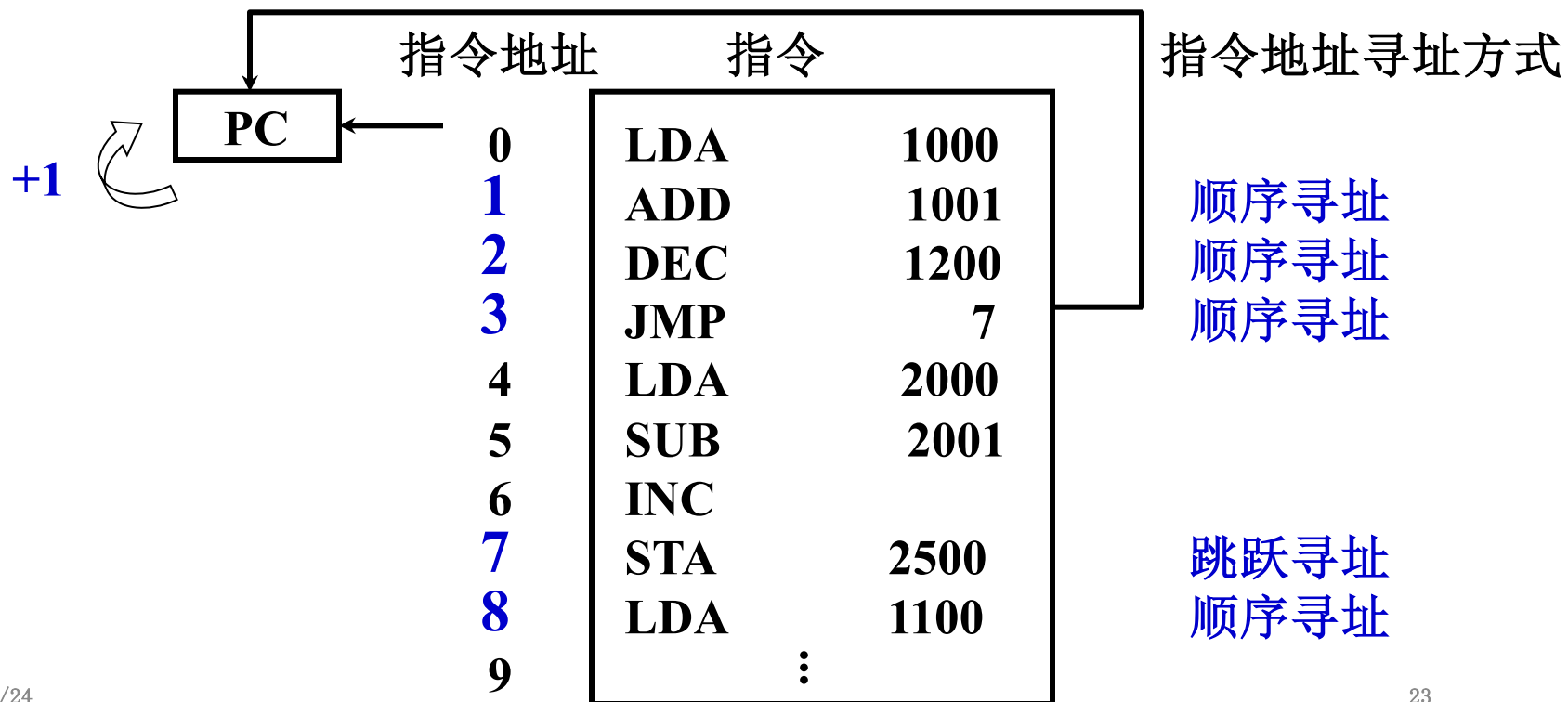
寻址方式 { 指令寻址  
数据寻址

## 7.3 寻址方式

### 一、指令寻址

顺序  $(PC) + 1 \longrightarrow PC$

跳跃 由转移指令指出



## 二、数据寻址

## 7.3

操作码	寻址特征	形式地址 A
-----	------	--------

形式地址          指令字中的地址

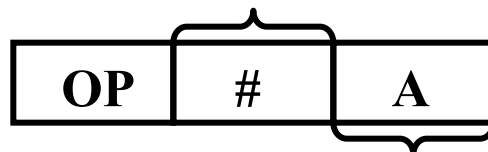
有效地址          操作数的真实地址

约定    指令字长 = 存储字长 = 机器字长

### 1. 立即寻址

形式地址 A 就是操作数

立即寻址特征



立即数    可正可负    补码

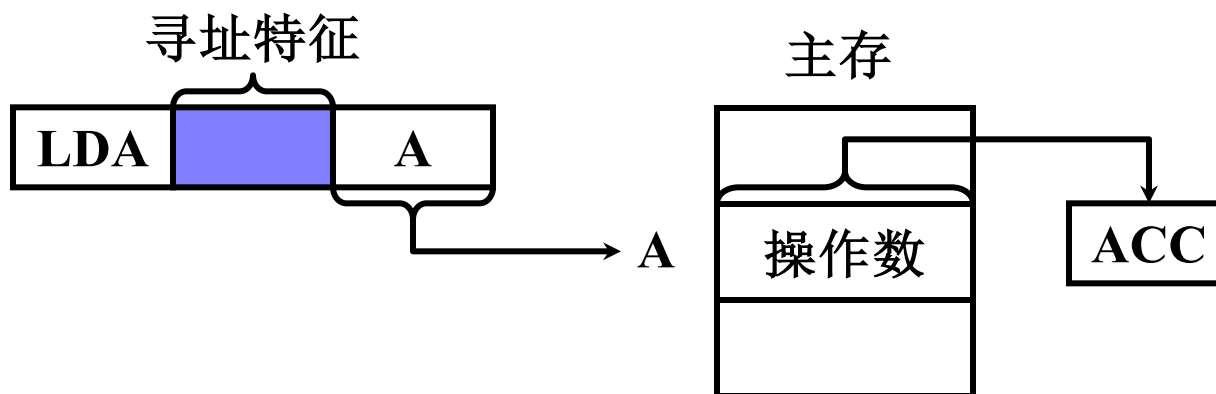
- 指令执行阶段不访存
- A 的位数限制了立即数的范围



## 2. 直接寻址

## 7.3

$EA = A$       有效地址由形式地址直接给出

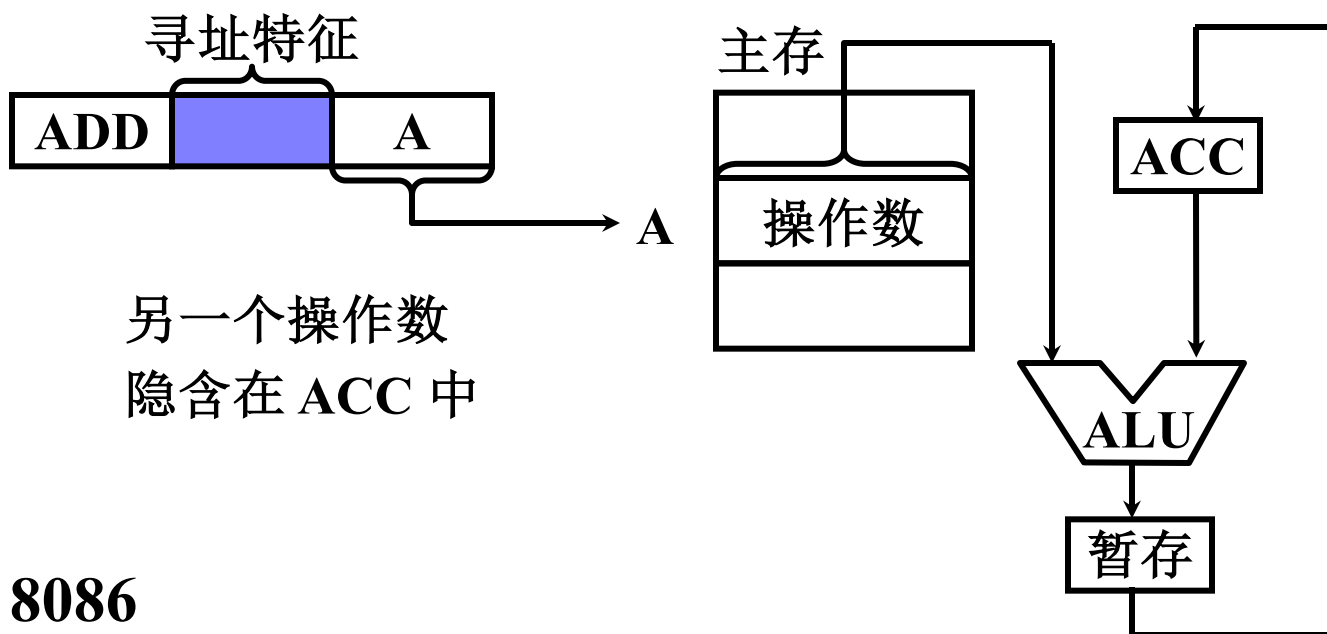


- 执行阶段访问一次存储器
- A 的位数决定了该指令操作数的寻址范围
- 操作数的地址不易修改（必须修改A）

### 3. 隐含寻址

## 7.3

操作数地址隐含在操作码中



如 8086

**MUL 指令** 被乘数隐含在 **AX**（16位）或 **AL**（8位）中

**MOVS 指令** 源操作数的地址隐含在 **SI** 中

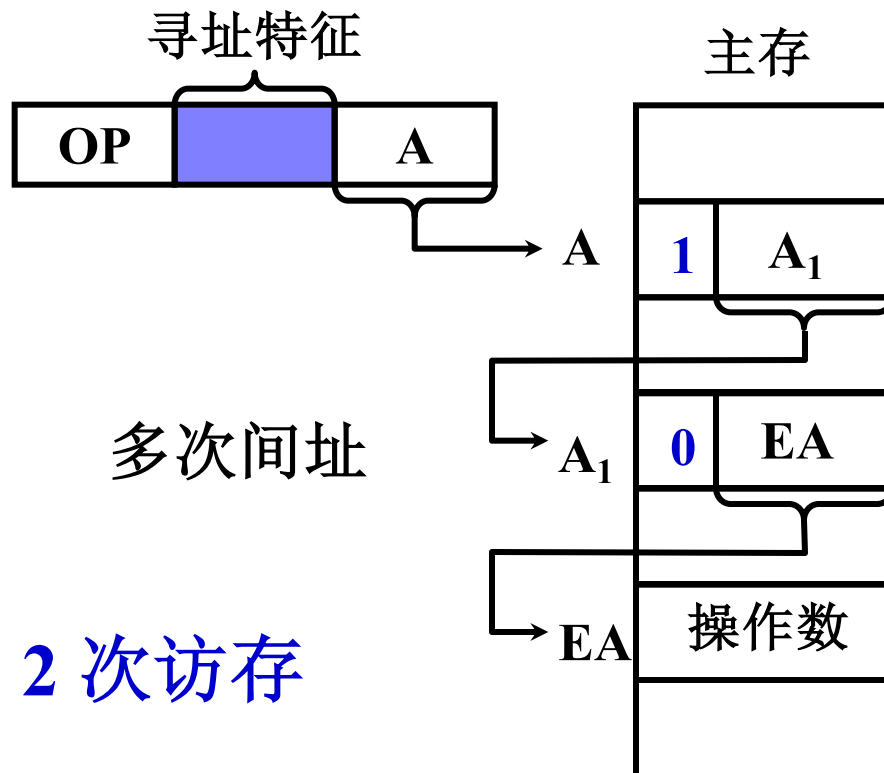
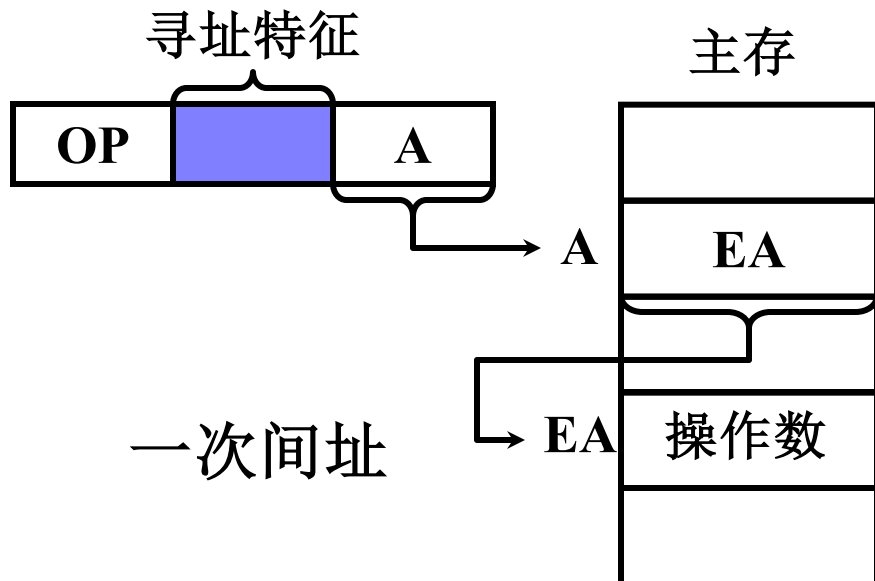
目的操作数的地址隐含在 **DI** 中

- 指令字中少了一个地址字段，可缩短指令字长

## 4. 间接寻址

7.3

$EA = (A)$  有效地址由形式地址间接提供

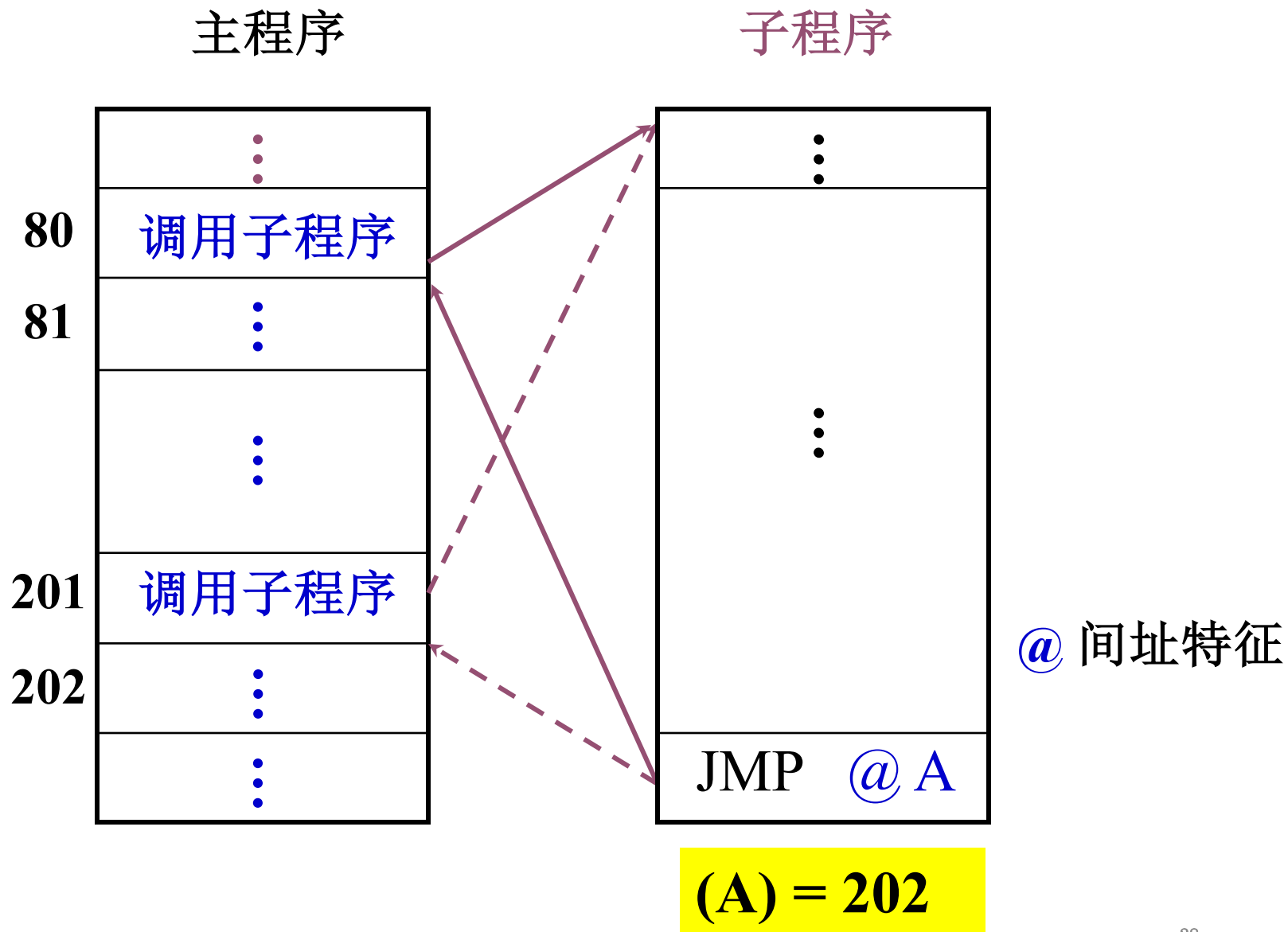


- 执行指令阶段 2 次访存
- 可扩大寻址范围
- 便于编制程序

多次访存

# 间接寻址编程举例

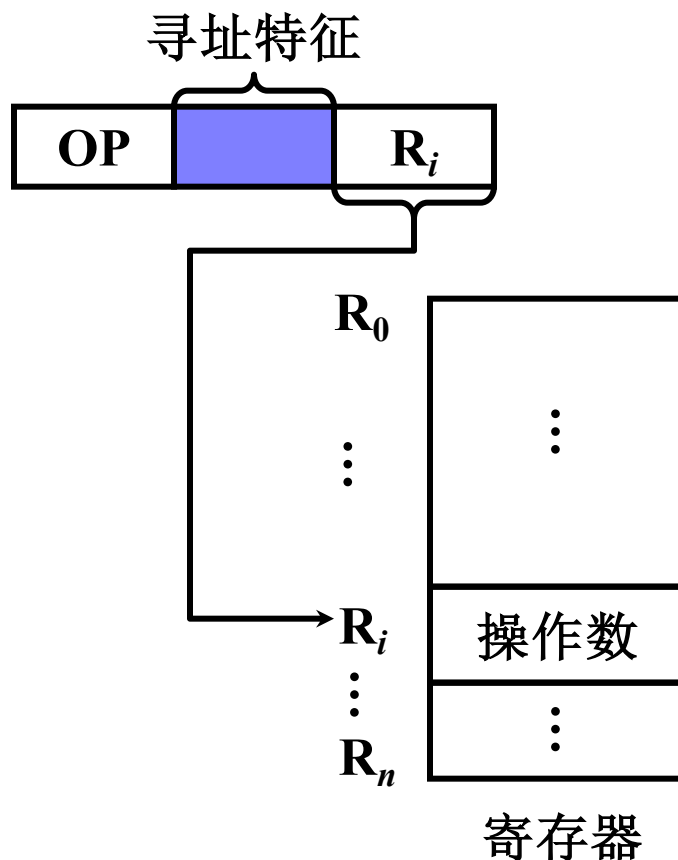
## 7.3



## 5. 寄存器寻址

7.3

$EA = R_i$       有效地址即为寄存器编号



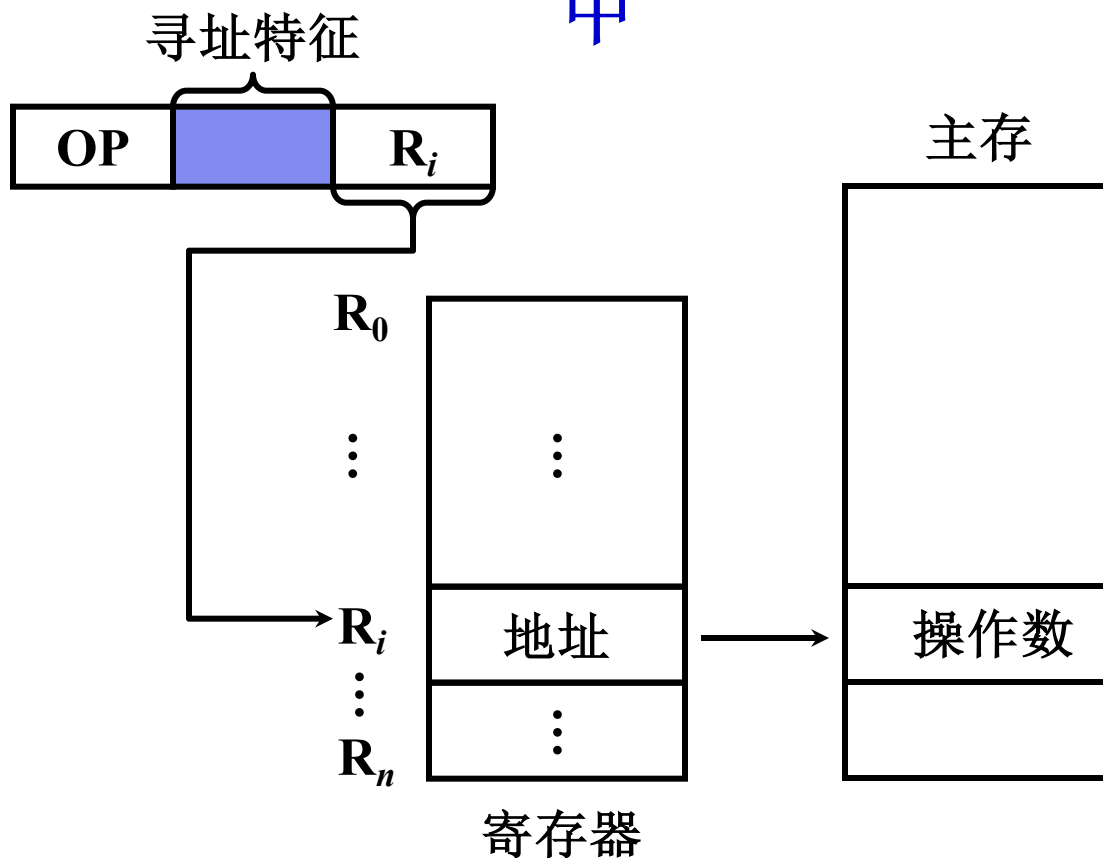
- 执行阶段不访存，只访问寄存器，执行速度快
- 寄存器个数有限，可缩短指令字长

## 6. 寄存器间接寻址

7.3

$$EA = (R_i)$$

有效地址在寄存器中



- 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

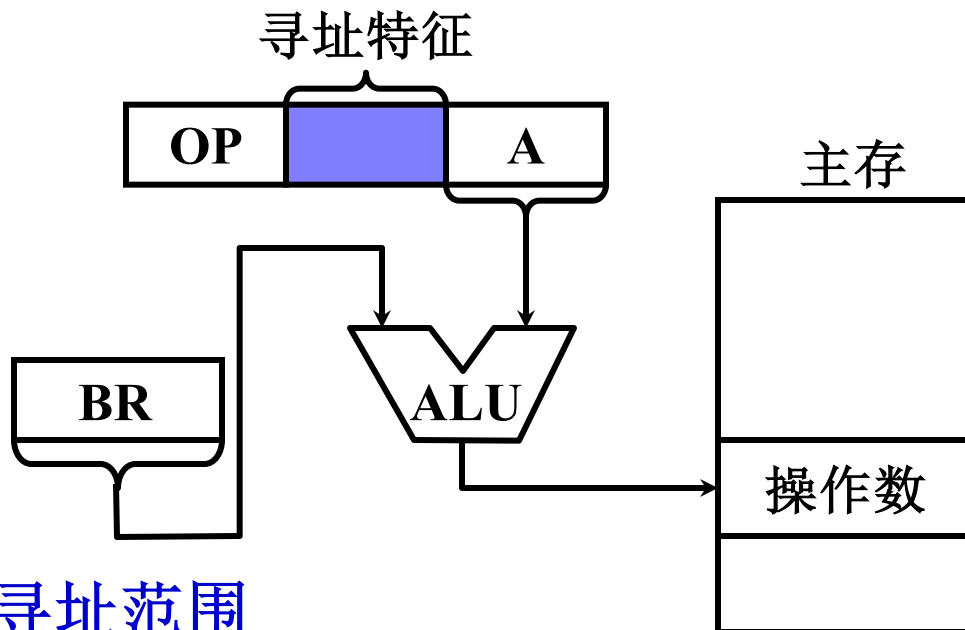
# 7. 基址寻址

## 7.3

### (1) 采用专用寄存器作基址寄存器

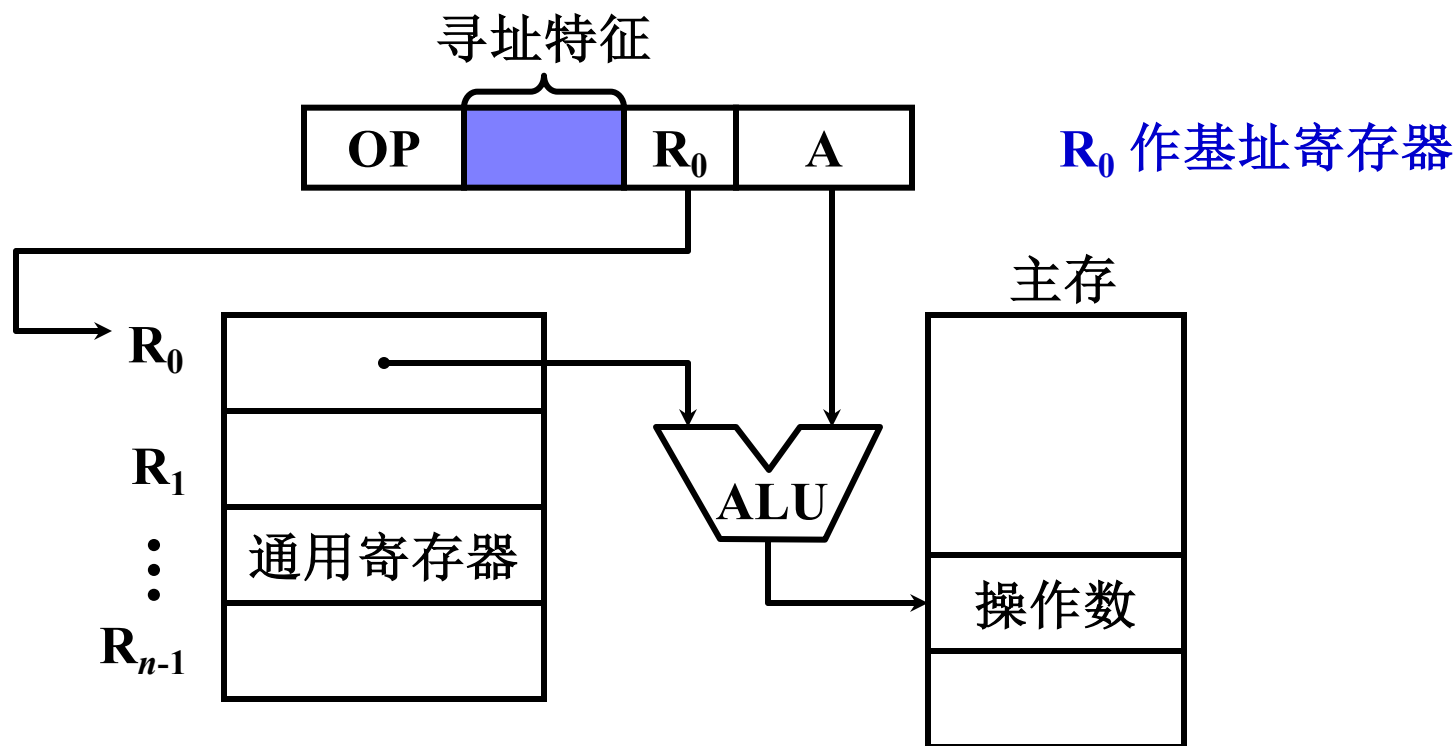
$$EA = (BR) + A$$

**BR** 为基址寄存器



- 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

## (2) 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

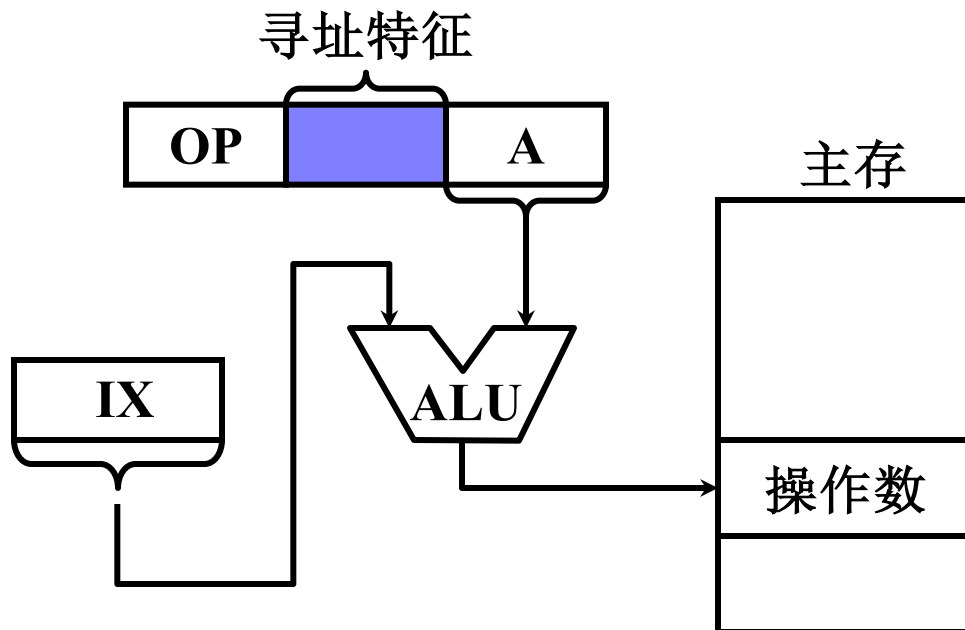


## 8. 变址寻址

7.3

$EA = (IX) + A$      $IX$  为变址寄存器（专用）

通用寄存器也可以作为变址寄存器



- 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

# 例 设数据块首地址为 $D$ ，求 $N$ 个数的平均值 7.3

## 直接寻址

**LDA**  $D$

**ADD**  $D + 1$

**ADD**  $D + 2$

$\vdots$

**ADD**  $D + (N - 1)$

**DIV**  $\# N$

**STA**  $ANS$

共  $N + 2$  条指令

## 变址寻址

**LDA**  $\# 0$

**LDX**  $\# 0$   $X$  为变址寄存器

**ADD**  $X, D$   $D$  为形式地址

**INX**  $(X) + 1 \rightarrow X$

**CPX**  $\# N$   $(X)$  和  $\# N$  比较

**BNE**  $M$  结果不为零则转

**DIV**  $\# N$

**STA**  $ANS$

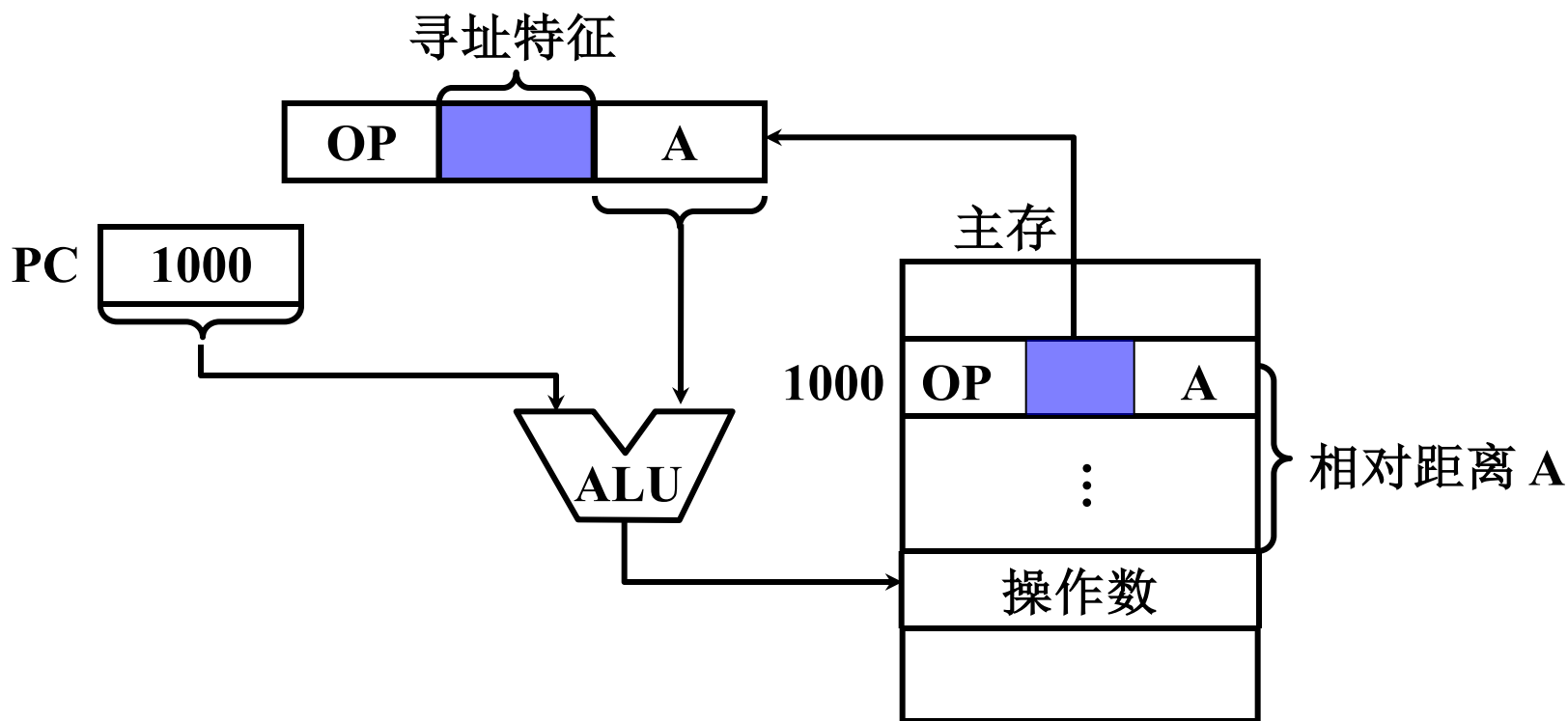
共 8 条指令

## 9. 相对寻址

7.3

$$EA = (PC) + A$$

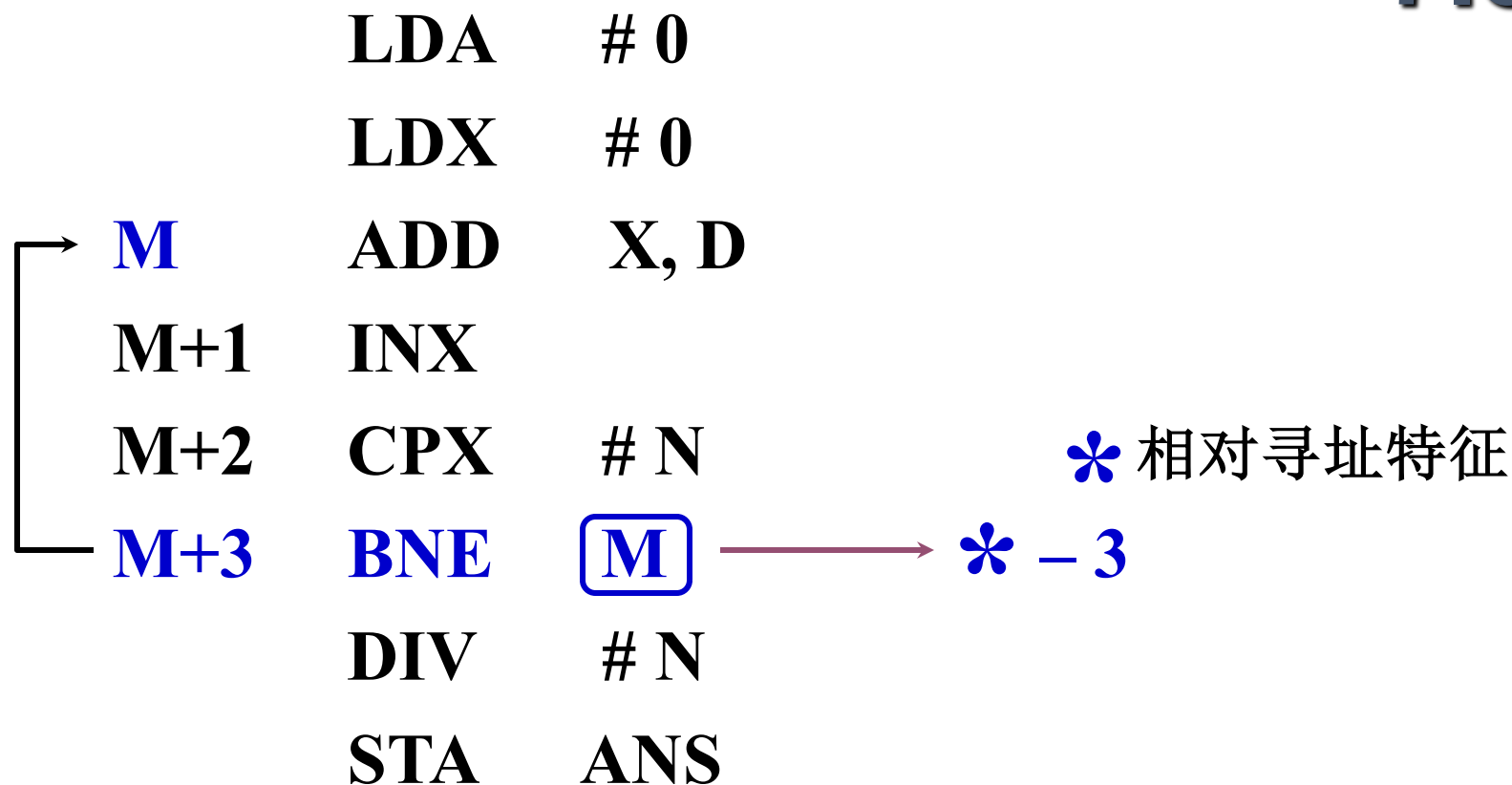
A 是相对于当前指令的位移量（可正可负，补码）



- A 的位数决定操作数的寻址范围
- 程序浮动
- 广泛用于转移指令

# (1) 相对寻址举例

7.3



**M** 随程序所在存储空间的位置不同而不同

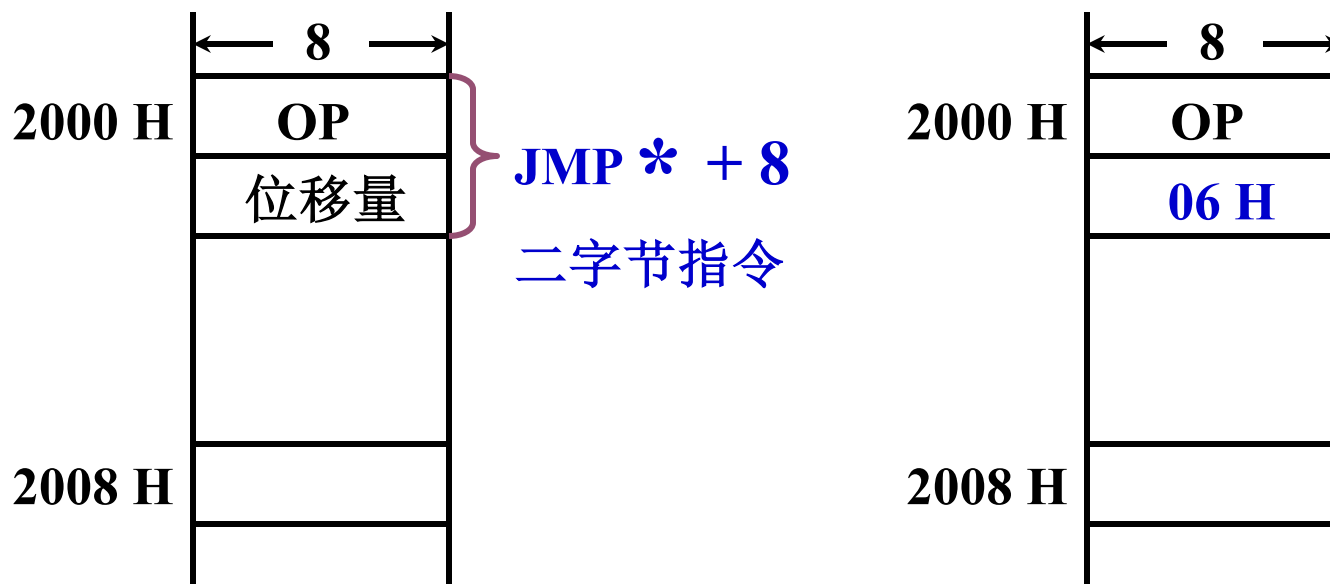
而指令 **BNE \* - 3** 与 指令 **ADD X, D** 相对位移量不变

指令 **BNE \* - 3** 操作数的有效地址为

$$EA = (M+3) - 3 = M$$

## (2) 按字节寻址的相对寻址举例

# 7.3



设 当前指令地址 **PC = 2000H**

转移后的目的地址为 **2008H**

因为 取出 **JMP \* + 8** 后 **PC = 2002H**

故 **JMP \* + 8** 指令 的第二字节为 **2008H - 2002H = 06H**

# 10. 堆栈寻址

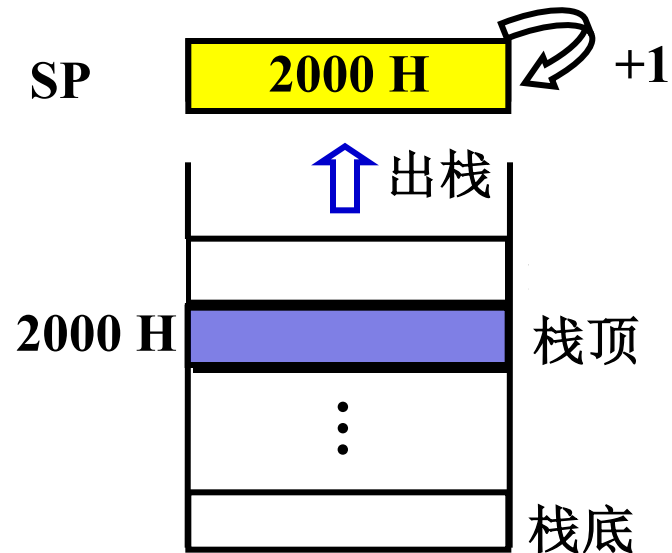
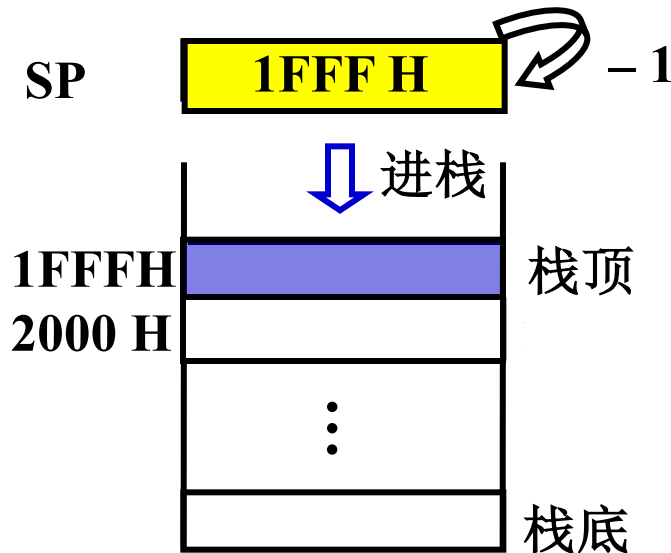
7.3

## (1) 堆栈的特点

堆栈 { 硬堆栈      多个寄存器  
         软堆栈      指定的存储空间

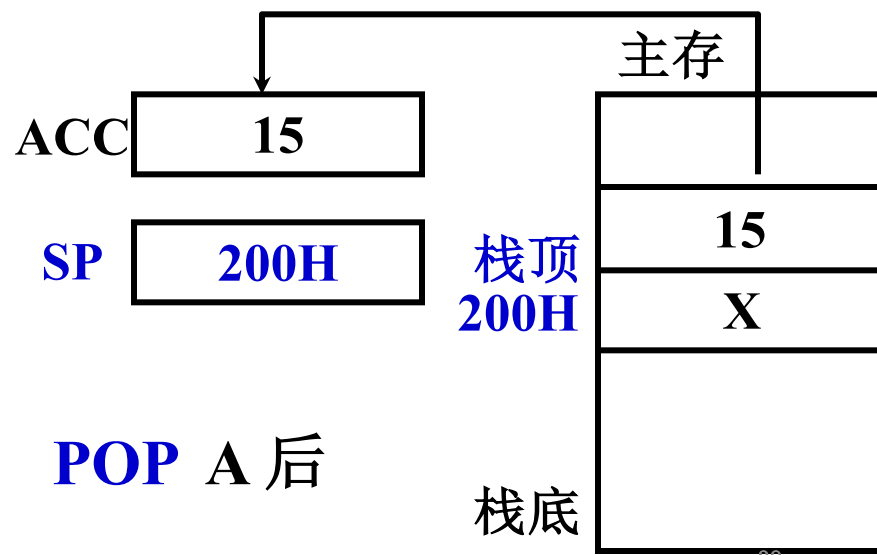
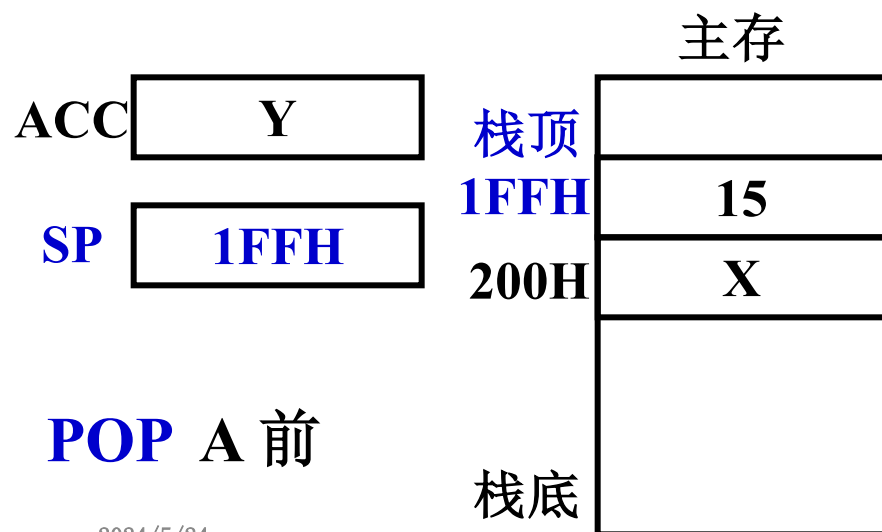
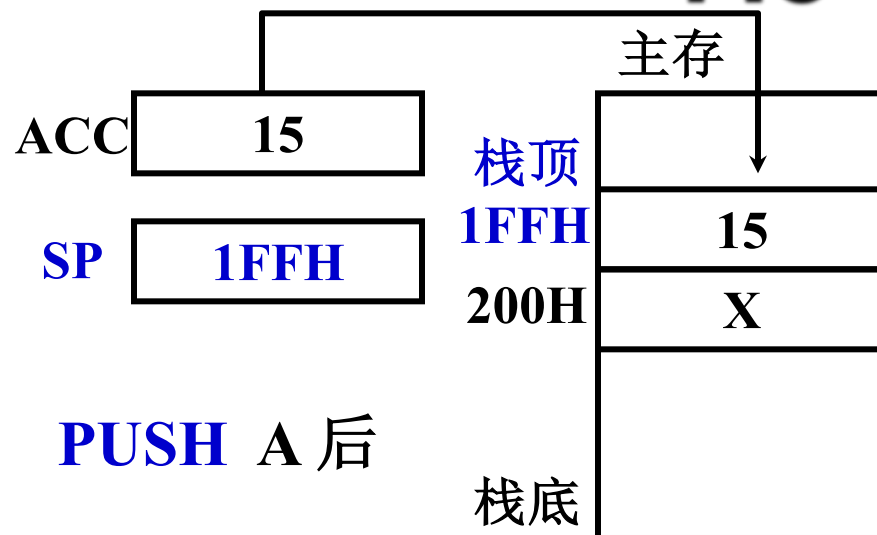
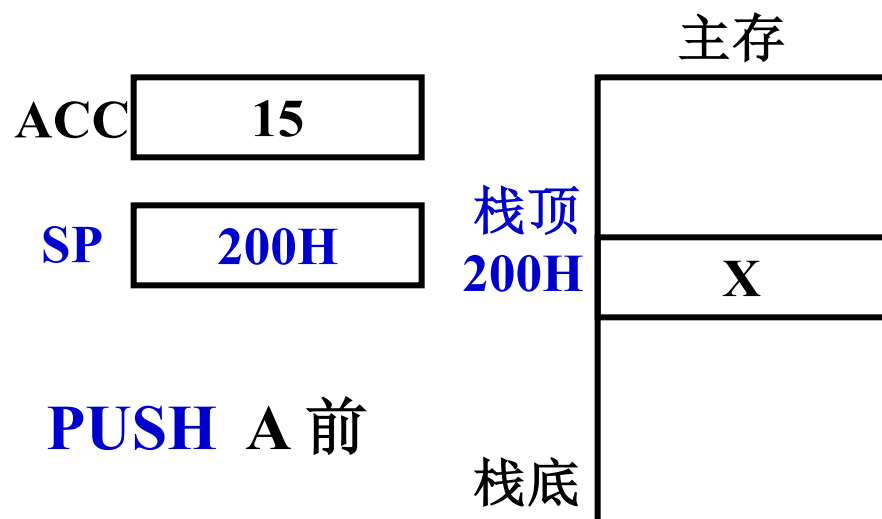
先进后出（一个入出口） 栈顶地址 由 **SP** 指出

进栈  $(SP) - 1 \rightarrow SP$       出栈  $(SP) + 1 \rightarrow SP$



## (2) 堆栈寻址举例

7.3



### (3) SP 的修改与主存编址方法有关 7.3

#### ① 按 字 编址

进栈  $(SP) - 1 \longrightarrow SP$

出栈  $(SP) + 1 \longrightarrow SP$

#### ② 按 字节 编址

存储字长 16 位 进栈  $(SP) - 2 \longrightarrow SP$

出栈  $(SP) + 2 \longrightarrow SP$

存储字长 32 位 进栈  $(SP) - 4 \longrightarrow SP$

出栈  $(SP) + 4 \longrightarrow SP$



# 基本寻址方式的算法和优缺点

## 7.3

假设：A=地址字段值，R=寄存器编号，  
EA=有效地址，(X)=X中的内容



方式	算法	主要优点	主要缺点
立即	操作数=A	指令执行速度快	操作数幅值有限
直接	EA=A	有效地址计算简单	地址范围有限
间接	EA=(A)	有效地址范围大	多次存储器访问
寄存器	操作数=(R)	指令执行快，指令短	地址范围有限
寄间接	EA=(R)	地址范围大	额外存储器访问
偏移	EA=A+(R)	灵活	复杂
堆栈	EA=栈顶	指令短	应用有限

偏移方式：将直接方式和寄存器间接方式结合起来。

有：相对 / 基址 / 变址三种

## 7.4 指令格式举例

### 一、设计指令格式时应考虑的各种因素

1. 指令系统的 **兼容性** （向上兼容）

2. 其他因素

**操作类型**

包括指令个数及操作的难易程度

**数据类型**

确定哪些数据类型可参与操作

**指令格式**

指令字长是否固定

操作码位数、是否采用扩展操作码技术，

地址码位数、地址个数、寻址方式类型

**寻址方式**

指令寻址、操作数寻址

**寄存器个数**

寄存器的多少直接影响指令的执行时间

### 3. 指令系统设计原则

## 7.4

- 完备性：指令丰富，功能齐全，使用方便
- 有效性：程序占空间小，执行速度快
- 规整性：
  - ✓ 对称性 （对不同寻址方式的支持）
  - ✓ 匀齐性 （对不同数据类型的支持）
  - ✓ 一致性 （指令长度和数据长度的一致性）
- 兼容性：系列机软件向上兼容
- 难以兼得

# 二、指令格式举例

## 7.4

### 1. MIPS指令格式

#### 32个MIPS寄存器

寄存器#	助记符	释义
0	\$zero	常量寄存器，值固定为0，硬件实现
1	\$at	临时变量，保留给汇编器使用
2~3	\$v0~\$v1	函数调用返回值
4~7	\$a0~\$a3	4个函数调用参数
8~15	\$t0~\$t7	暂存寄存器，子程序内可直接使用
16~23	\$s0~\$s7	变量寄存器，子程序返回应复原内容
24~25	\$t8~\$t9	暂存寄存器，子程序内可直接使用
26~27	\$k0~\$k1	操作系统保留，中断异常处理
28	\$gp	全局指针 (Global Pointer)
29	\$sp	堆栈指针 (Stack Pointer)
30	\$fp	帧指针 (Frame Pointer)
31	\$ra	函数返回地址 (Return Address)

- 32个32位通用寄存器\$0~\$31
- 32个32位单精度浮点寄存器f0-f31
- 2个32位乘、商寄存器H<sub>i</sub>和L<sub>0</sub>
- 程序寄存器PC是单独的寄存器
- 无程序状态寄存器
- RISC-V也有类似的32个寄存器设置

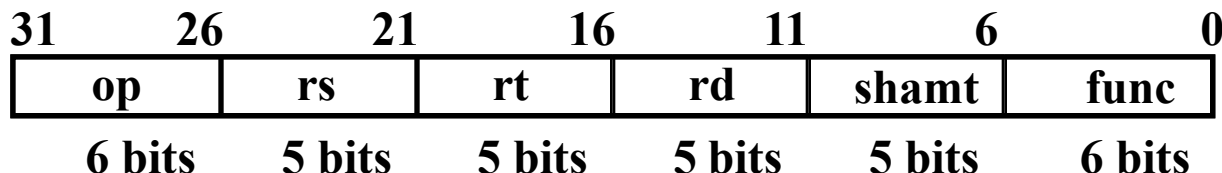
# 1. MIPS指令格式

## 7.4

- ◆ 所有指令都是32位宽，须按字地址对齐

R-Type指令

字地址为4的倍数！



- ◆ 有三种指令格式

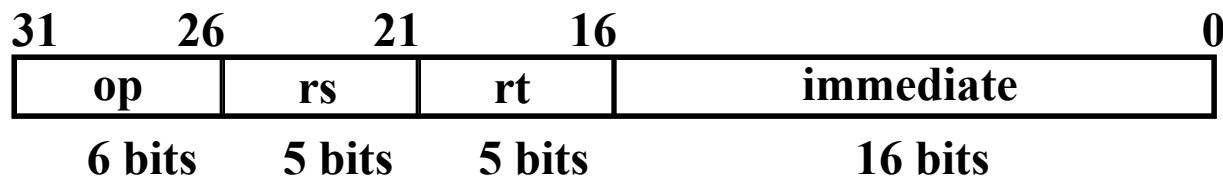
– R-Type

两个操作数和结果都在寄存器的运算指令。如： `sub rd, rs, rt`

– I-Type

- 运算指令：一个寄存器、一个立即数。如： `ori rt, rs, imm16`
- LOAD和STORE指令。如： `lw rt, rs, imm16`
- 条件分支指令。如： `beq rs, rt, imm16`

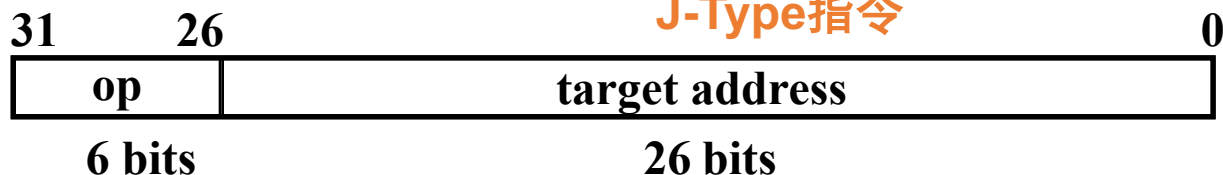
I-Type指令



– J-Type

无条件跳转指令。如： `j target`

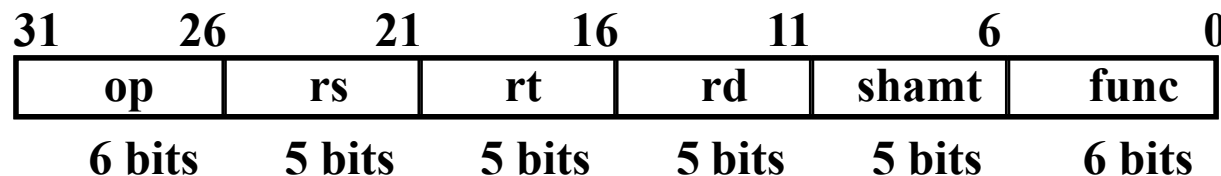
J-Type指令



# MIPS指令字段含义

## 7.4

### R-Type指令



op: 操作码

rs: 第一个源操作数寄存器

rt: 第二个源操作数寄存器

rd: 结果寄存器

shamt: 移位指令的位移量

func: R-Type指令的op字段是固定的“000000”，具体操作由func字段给定。

例如 func=“100000”时，表示“加法”运算。

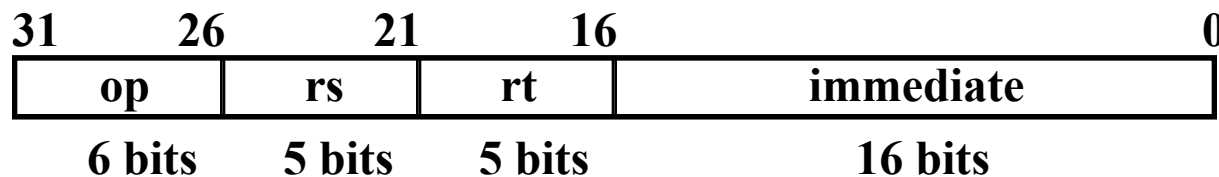
操作码不同，定义不同的含义；

操作码相同，再根据功能码定义不同的含义。

# MIPS指令字段含义

## 7.4

### I-Type指令



op: 操作码

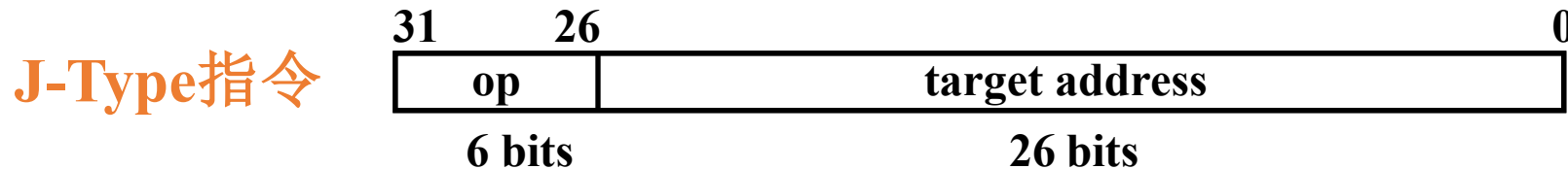
rs: 第一个源操作数寄存器

rt: 第二个源操作数寄存器

immediate: 立即数，或load/store指令和分支指令的偏移地址

# MIPS指令字段含义

## 7.4



op: 操作码

target address: 无条件转移地址的低26位。

将PC高4位拼上26位直接地址，最后添2个“0”就是32位目标地址。  
为何最后两位要添“0”？

指令按字地址对齐，所以每条指令的地址都是4的倍数（最后两位为0）。

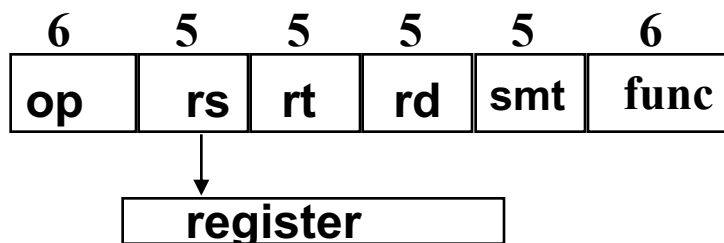


# MIPS Addressing Modes (寻址方式) 7.4

R-format:

OP=000000b

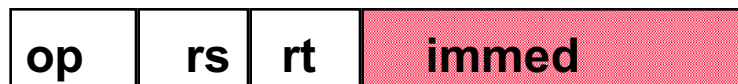
Register



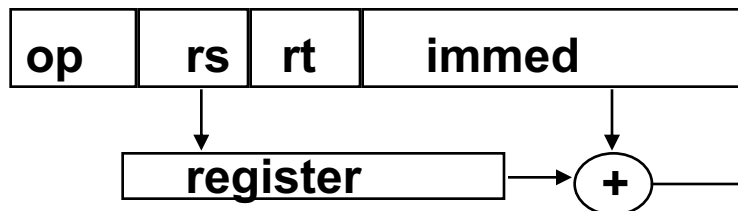
有专门的寻址方式  
字段 (Mod) 吗?

没有! 由指令格式来  
确定, 而指令格式由  
op来确定!

I-format:  
Immediate



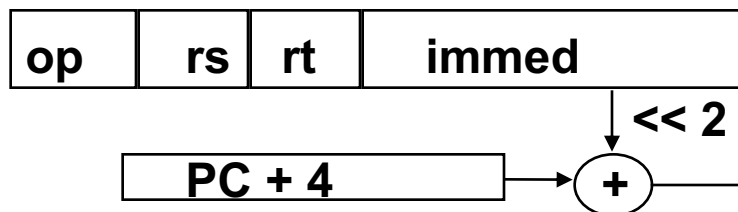
Base或Index  
基址或变址



Byte / Half Word / Word

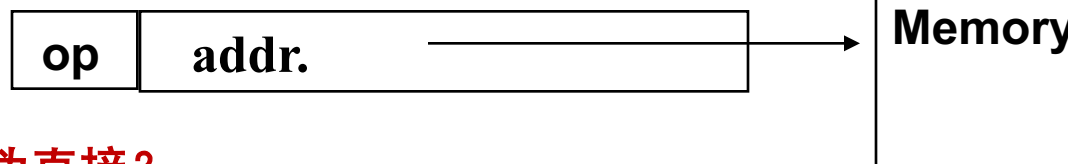


PC-relative  
相对寻址



J-format: OP=000010b or 000011b

Pseudodirect  
伪直接寻址



为什么称伪直接?

最终地址=PC<sub>31~28</sub>||addr.||00 位数: 4+26+2=32

# 2.X86指令系统举例-- IA-32的寄存器组织

## 7.4

%eax	累加器 (32bits)	%ax (16bits)	%ah (8bits)	%al (8bits)
%ecx	计数寄存器	%cx	%ch	%cl
%edx	数据寄存器	%dx	%dh	%dl
%ebx	基址寄存器	%bx	%bh	%bl
%esi	源变址寄存器	%si		
%edi	目标变址寄存器	%di		
%esp	堆栈指针	%sp		
%ebp	基址指针	%bp		
%eip	指令指针	ip		
%eeflags	标志寄存器	flags		

- 8个通用寄存器
- 两个专用寄存器
- 6个段寄存器

CS (代码段) 16bits
SS (堆栈段)
DS (数据段)
ES (附加段)
FS (附加段)
GS (附加段)

# X86指令系统举例：Pentium指令格式

## 7.4

**前缀：**包括指令、段、操作数长度、地址长度四种类型

前缀类型：	指令前缀	段前缀	操作数长度	地址长度
字节数：	0或1	0或1	0或1	0或1

**指令：**含操作码、寻址方式、SIB、位移量和直接数据五部分



**指令格式复杂：**

①位移量和立即数：都可是1/2/4B。

②SIB中基址B和变址I：都可是8个GPR中的任一个，再由SS给出比例因子。

③操作码低2位为d、w位：其中opcode和机器模式（16 / 32位，w位）一起确定寄存器位数（AL / AX / EAX）和操作方向（d位）。

④寻址方式：Mod、Reg/OP、R/M三个字段与操作码w位和机器模式一起确定操作数所在的寄存器编号或有效地址计算方式。Reg/OP字段也可用作扩展操作码。

# X86指令系统举例：Pentium指令格式

## 7.4

**前缀：**包括指令、段、操作数长度、地址长度四种类型

前缀类型：	指令前缀	段前缀	操作数长度	地址长度
字节数：	0或1	0或1	0或1	0或1

**指令：**含操作码、寻址方式、SIB、位移量和直接数据五部分



**指令功能复杂：**

变长指令字：1B~17B

变长操作码：4b / 5b / 6b / 7b / 8b / .....

变长操作数：Byte / Word / DW / QW

变长寄存器：8位 / 16位 / 32位

**call 指令，自动把返回地址压栈**

**专门的push/pop指令，自动修改栈指针**

**ALU指令在Flags中隐含生成条件码**

**ALU指令中的一个操作数可来自存储器**

**提供基址加比例索引寻址**

# Pentium处理器的寻址方式

## 7.4

操作数的来源：

立即数(立即寻址)：直接来自指令

寄存器(寄存器寻址)：来自32位 / 16位 / 8位通用寄存器

存储单元(其他寻址)：需进行地址转换

虚拟地址 => 线性地址LA (=> 内存地址)

分段

分页

指令中的信息：

- (1) 段寄存器SR (隐含或显式给出)
- (2) 8/16/32位偏移量A (显式给出)
- (2) 基址寄存器B (明显给出，任意通用寄存器皆可)
- (3) 变址寄存器I (明显给出，除ESP外的任意通用寄存器皆可。)
  - 有比例变址和非比例变址
  - 比例变址时要乘以比例因子S (1:8位 / 2:16位 / 4:32位 / 8:64位)

# MIPS X86 差异

## 7.4

#	X86	MIPS
1	变长（1-15bytes）	定长指令
2	指令数多 CISC	指令数少 RISC
3	8个通用寄存器	32个通用寄存器
4	寻址方式复杂	寻址方式简单
5	有标志寄存器	无标志寄存器
6	最多两地址指令	三地址指令
7	无限制	只有Load/store能访问存储器
8	有堆栈指令 push, pop	无堆栈指令（访存指令代替）
9	有I/O指令	无I/O指令(设备统一编址)

## 7.5 RISC 技术

### 一、RISC 的产生和发展

**RISC (Reduced Instruction Set Computer)**

**CISC (Complex Instruction Set Computer)**

**80 — 20 规律 — RISC技术**

- 典型程序中 **80%** 的语句仅仅使用处理机中 **20%** 的指令
- 执行频度高的简单指令，因复杂指令的存在，执行速度无法提高
- ？ 能否用 **20%** 的简单指令组合不常用的 **80%** 的指令功能

## 二、RISC 的主要特征

- 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
- 指令 **长度固定、指令格式种类少、寻址方式少**
- 只有 **LOAD / STORE** 指令访存
- CPU 中有 **多个** 通用 **寄存器**
- 采用 **流水技术** 一个时钟周期内完成一条指令
- 采用 **组合逻辑** 实现控制器
- 采用 **优化** 的 **编译** 程序



### 三、CISC 的主要特征

- 系统指令 复杂庞大，各种指令使用频度相差大
- 指令 长度不固定、指令格式种类多、寻址方式多
- 访存 指令 不受限制
- CPU 中设有 专用寄存器
- 大多数指令需要 多个时钟周期 执行完毕
- 采用 微程序 控制器
- 难以 用 优化编译 生成高效的代码

## 四、RISC和CISC 的比较

1. RISC更能 充分利用 VLSI 芯片的面积

2. RISC 更能 提高计算机运算速度

指令数、指令格式、寻址方式少，  
通用 寄存器多，采用 组合逻辑，  
便于实现 指令流水

3. RISC 便于设计，可 降低成本，提高 可靠性

4. RISC 有利于编译程序代码优化

5. RISC 不易 实现 指令系统兼容