

第七章 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

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

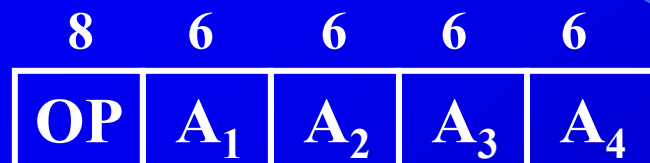
操作码的位数随地址数的减少而增加

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

2. 地址码

7.1

(1) 四地址



A₁ 第一操作数地址

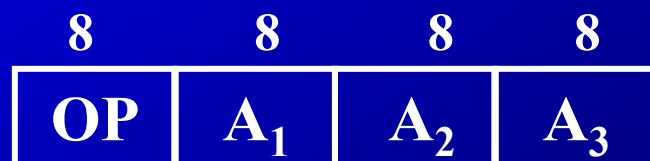
A₂ 第二操作数地址

A₃ 结果的地址

A₄ 下一条指令地址

$(A_1) \text{ OP } (A_2) \longrightarrow A_3$

(2) 三地址



$(A_1) \text{ OP } (A_2) \longrightarrow A_3$

设指令字长为 32 位

操作码固定为 8 位

4 次访存

寻址范围 $2^6 = 64$

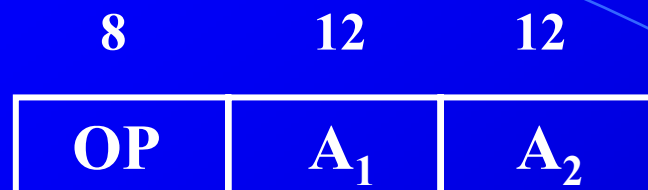
若 PC 代替 A₄

4 次访存

寻址范围 $2^8 = 256$

若 A₃ 用 A₁ 或 A₂ 代替

(3) 二地址



或 (A₁) OP (A₂) → A₁

(A₁) OP (A₂) → A₂

若结果存于 ACC 3次访存

4 次访存

寻址范围 $2^{12} = 4 \text{ K}$

若ACC 代替 A₁ (或A₂)

(4) 一地址



(ACC) OP (A₁) → ACC

2 次访存

寻址范围 $2^{24} = 16 \text{ M}$

(5) 零地址 无地址码

二、指令字长

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

7.1

➤ 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 R₁, R₂, R₃**

二地址 **OP R₁, R₂**

一地址 **OP R₁**

- 可缩短指令字长
- 指令执行阶段不访存

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、数据在存储器中的存放方式

字地址	低字节			
0	3	2	1	0
4	7	6	5	4

字地址 为 低字节 地址

字地址	低字节			
0	0	1	2	3
4	4	5	6	7

字地址 为 高字节 地址

存储器中的数据存放（存储字长为32位） 7.2

边界对准

地址（十进制）

字（地址0）				0
字（地址4）				4
字节（地址11）	字节（地址10）	字节（地址9）	字节（地址8）	8
字节（地址15）	字节（地址14）	字节（地址13）	字节（地址12）	12
半字（地址18）✓		半字（地址16）✓		16
半字（地址22）✓		半字（地址20）✓		20
双字（地址24）▲				24
双字				28
双字（地址32）▲				32
双字				36

边界未对准

地址（十进制）

字(地址2)		半字(地址0)	0
字节(地址7)	字节(地址6)	字(地址4)	4
半字(地址10)		半字(地址8)	8

三、操作类型

7.2

1. 数据传送

源	寄存器	寄存器	存储器	存储器
目的	寄存器	存储器	寄存器	存储器
例如	MOVE	STORE MOVE PUSH	LOAD MOVE POP	MOVE
置“1”，清“0”				

2. 算术逻辑操作

加、减、乘、除、增 1、减 1、求补、浮点运算、十进制运算
与、或、非、异或、位操作、位测试、位清除、位求反

如 8086 ADD SUB MUL DIV INC DEC CMP NEG
 AAA AAS AAM AAD
 AND OR NOT XOR TEST

3. 移位操作

算术移位 逻辑移位

循环移位（带进位和不带进位）

4. 转移

(1) 无条件转移 **JMP**

(2) 条件转移

结果为零转 ($Z = 1$) **JZ**

结果溢出转 ($O = 1$) **JO**

结果有进位转 ($C = 1$) **JC**

跳过一个指令 **SKP**

如

300

⋮

305

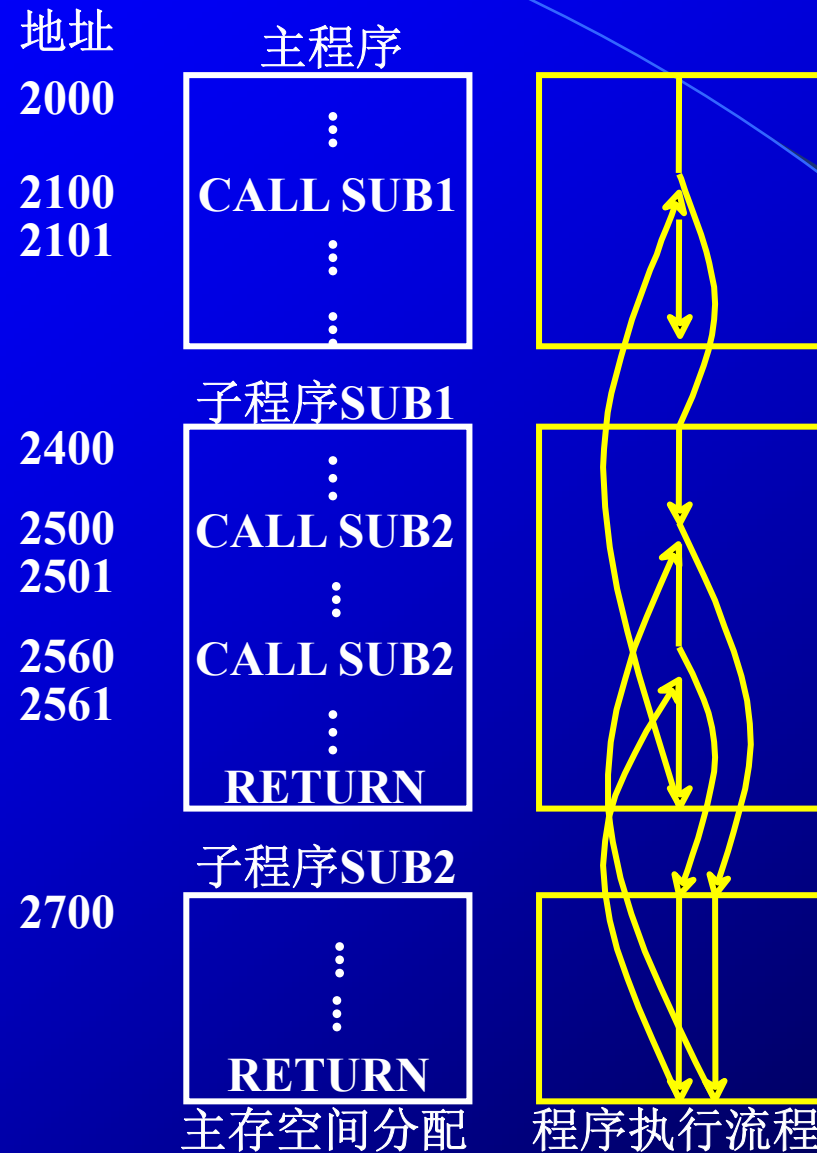
306

307

完成触发器

SKP DZ D = 0 则跳

(3) 调用和返回



(4) 陷阱 (Trap) 与陷阱指令

意外事故的中断

7.2

- 一般不提供给用户直接使用
在出现事故时，由 CPU 自动产生并执行（隐指令）
- 设置供用户使用的陷阱指令

如 8086 INT TYPE 软中断

提供给用户使用的陷阱指令，完成系统调用

5. 输入输出

入 端口地址 → CPU 的寄存器

如 IN AX, m IN AX, DX

出 CPU 的寄存器 → 端口地址

如 OUT n, AX OUT DX, AX

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
-----	------	--------

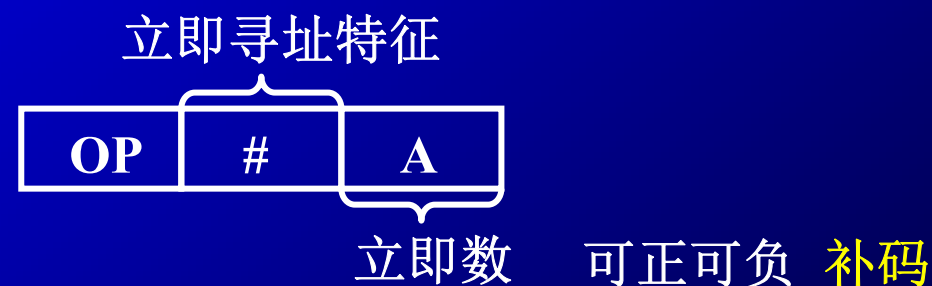
形式地址 指令字中的地址

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

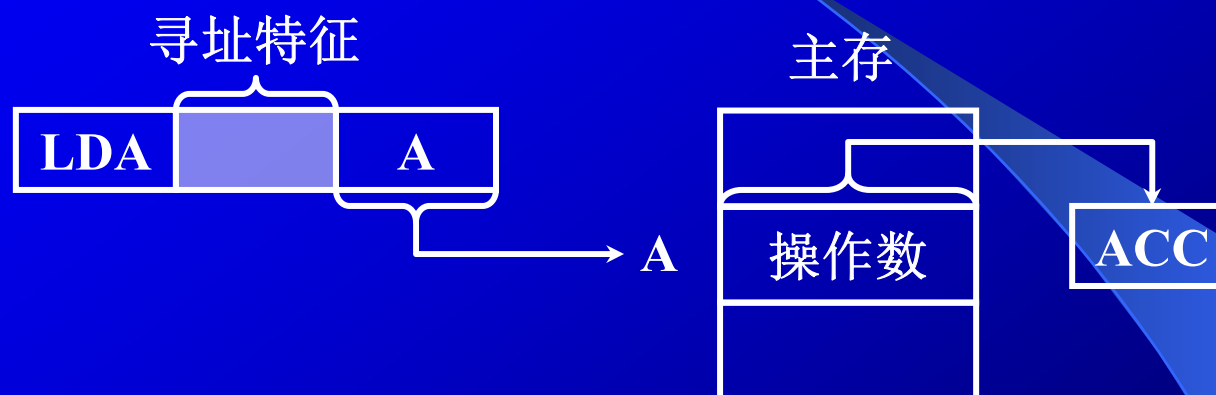


- 指令执行阶段不访存
- A 的位数限制了立即数的范围

2. 直接寻址

7.3

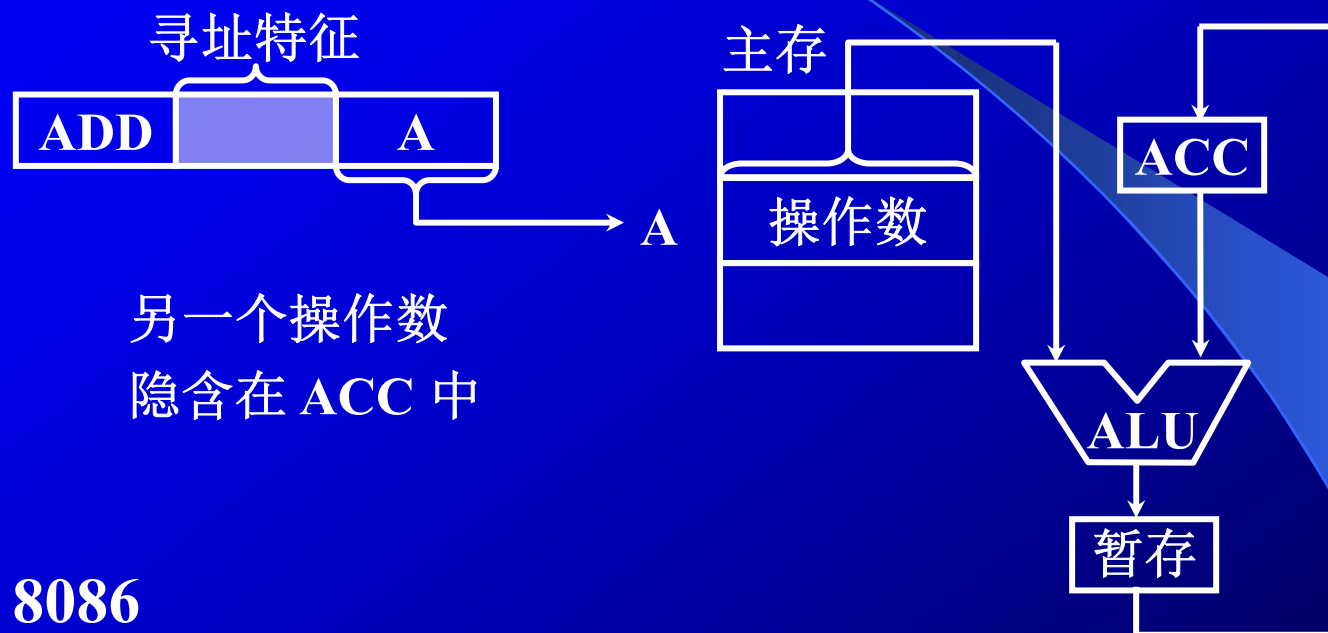
$EA = A$ 有效地址由形式地址直接给出



- 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

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

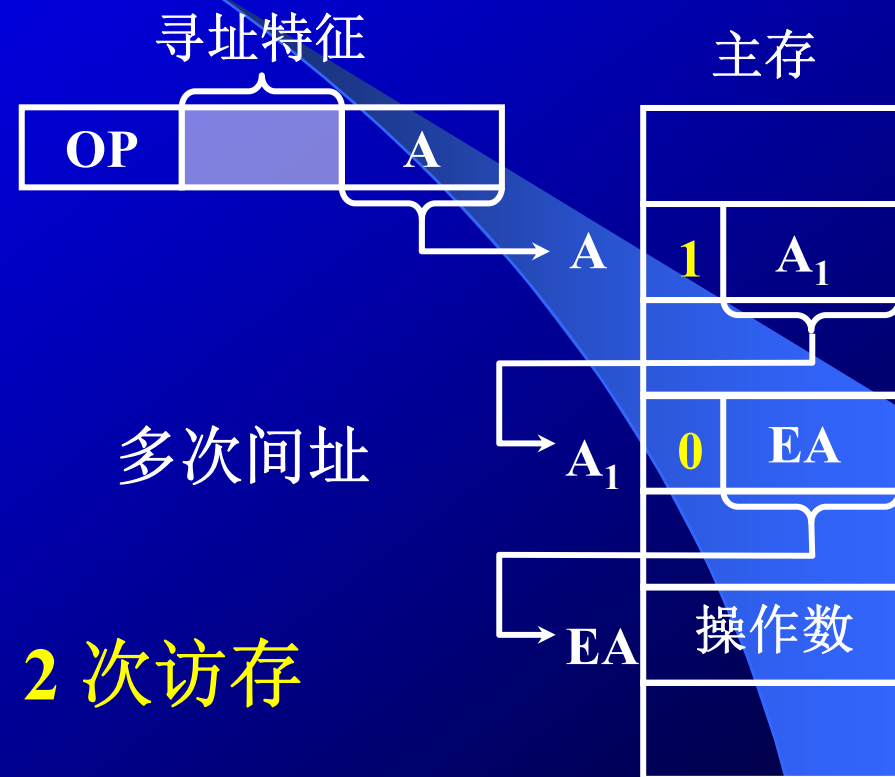
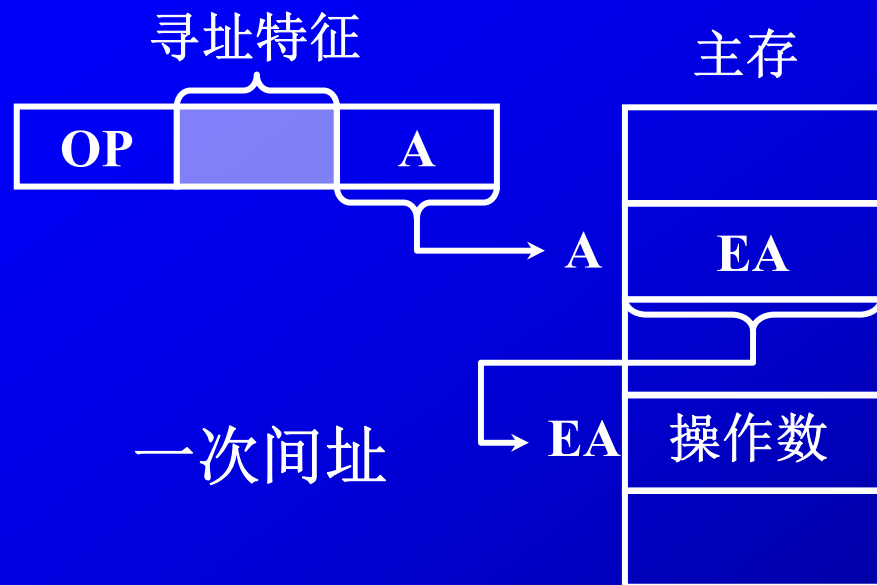
目的操作数的地址隐含在 **DI** 中

- 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

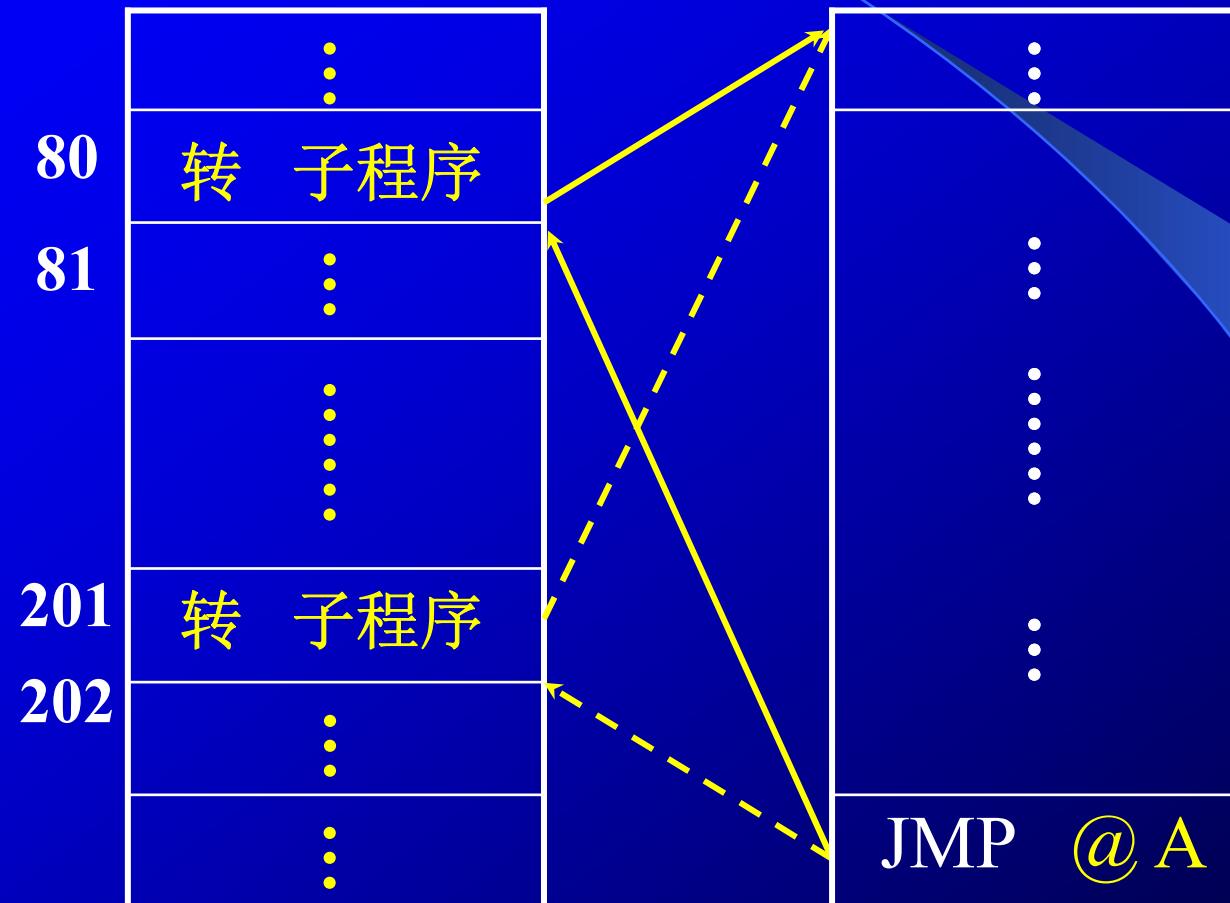
多次访存

间接寻址编程举例

7.3

主程序

子程序



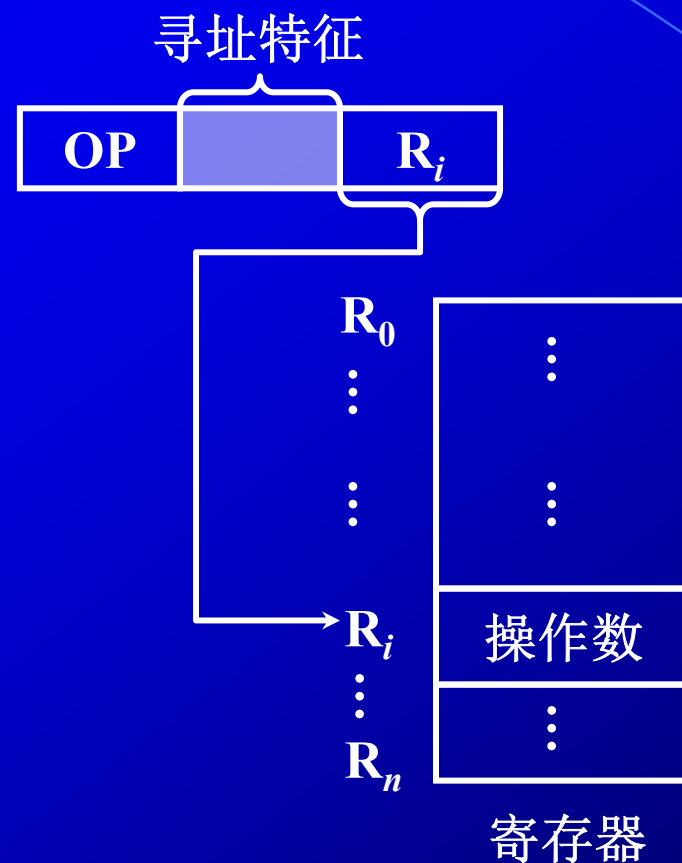
@ 间址特征

(A) = 202

5. 寄存器寻址

7.3

$EA = R_i$ 有效地址即为寄存器编号



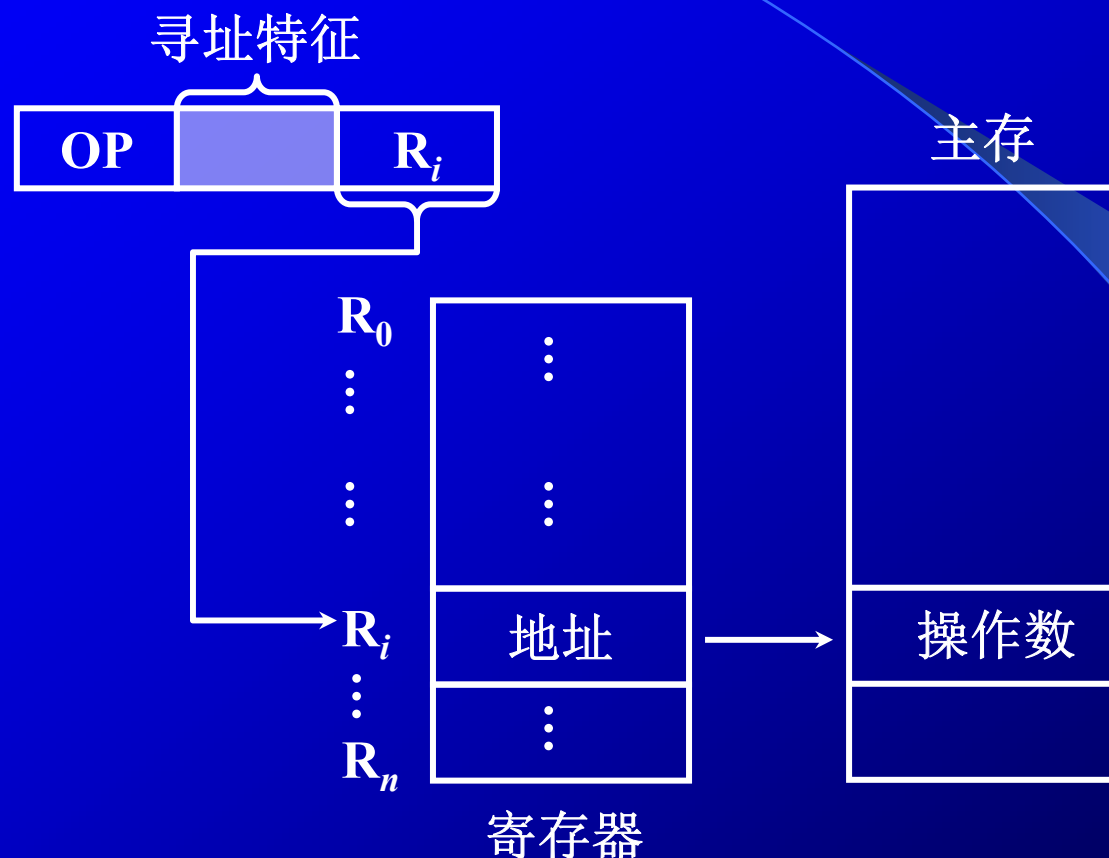
- 执行阶段不访存，只访问寄存器，执行速度快
- 寄存器个数有限，可缩短指令字长

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

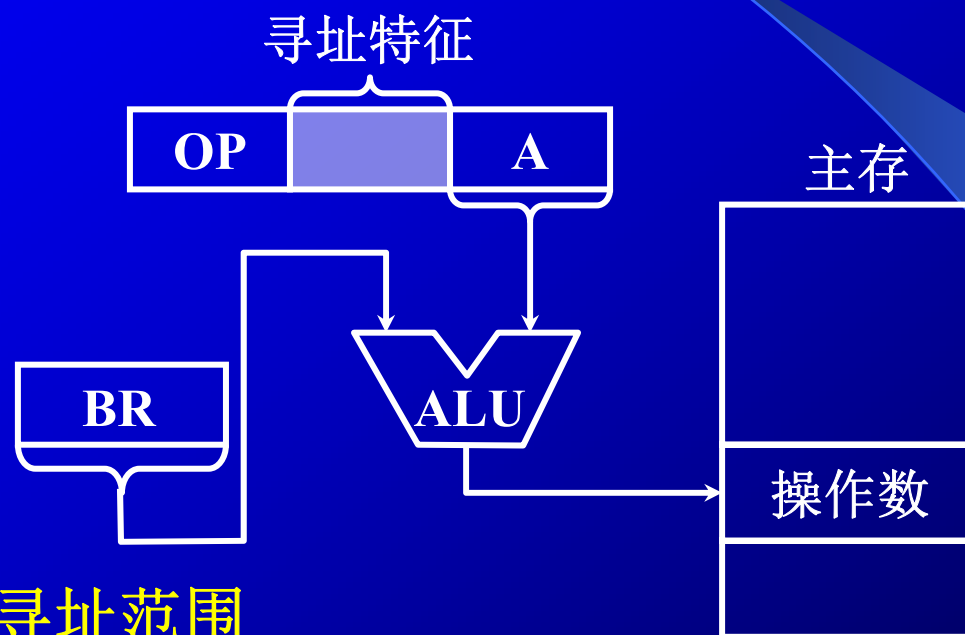
7. 基址寻址

7.3

(1) 采用专用寄存器作基址寄存器

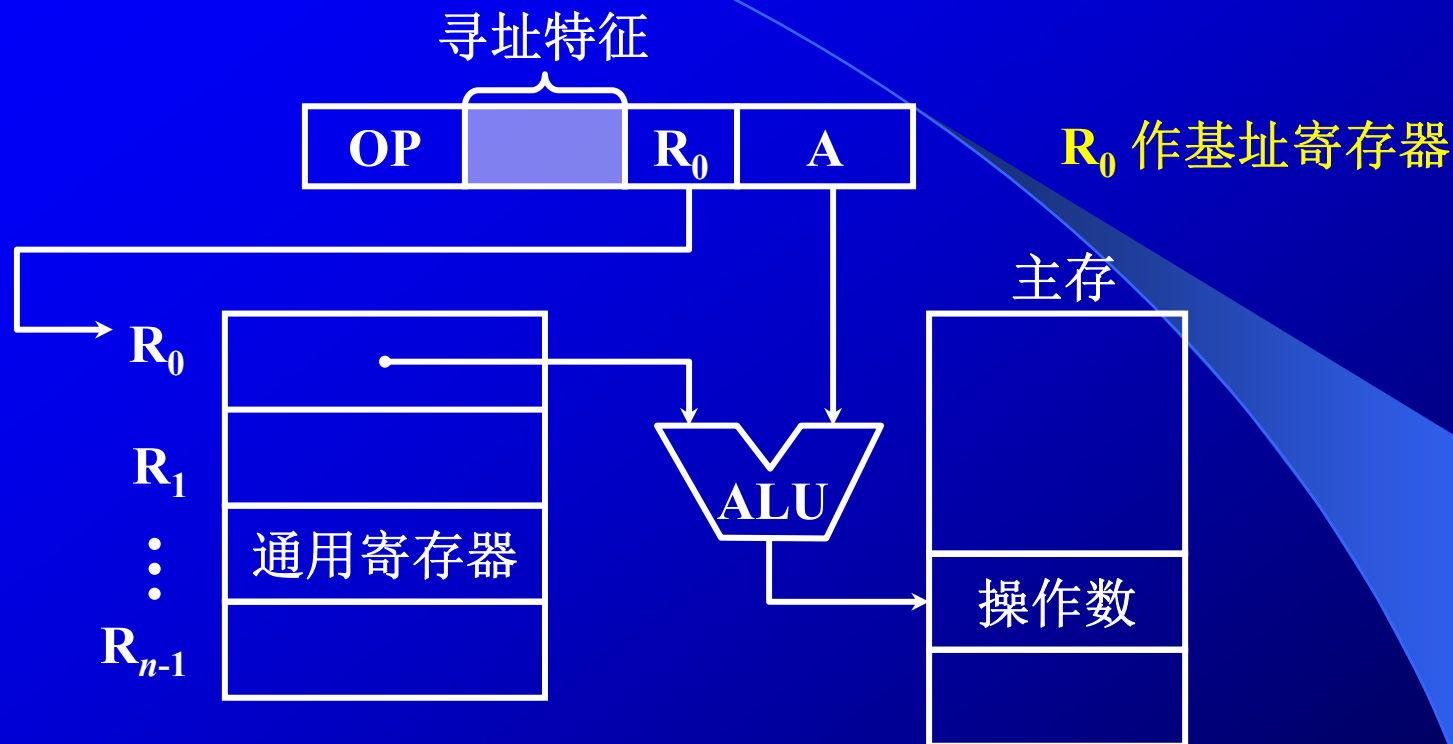
$$EA = (BR) + A$$

BR 为基址寄存器



- 可扩大寻址范围
- 便于程序搬家
- **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



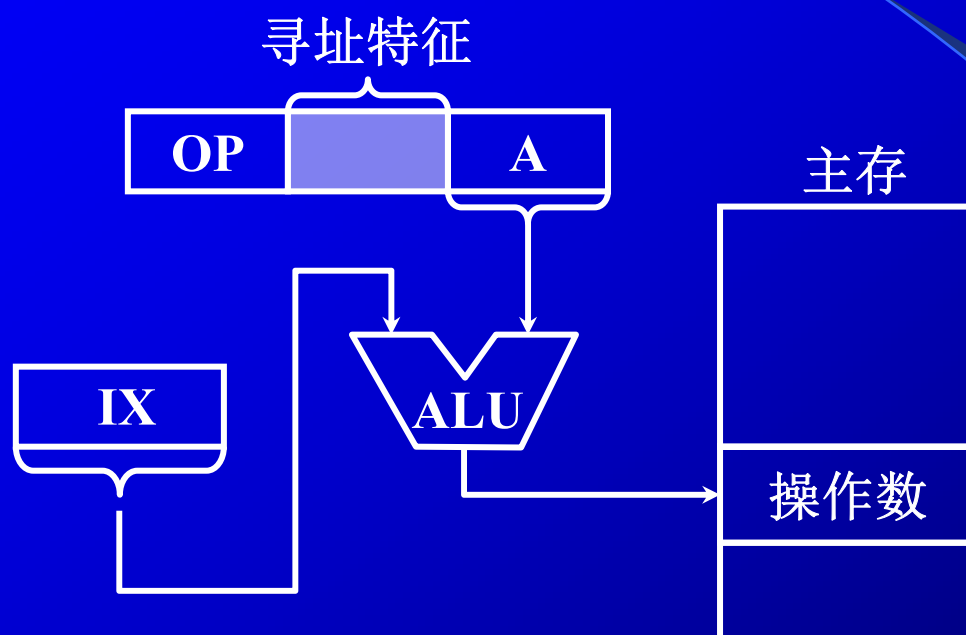
- 由用户指定哪个通用寄存器作为基址寄存器
- 基址寄存器的内容由操作系统确定
- 在程序的执行过程中 R_0 内容不变，形式地址 A 可变

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 $D + 2$

⋮

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

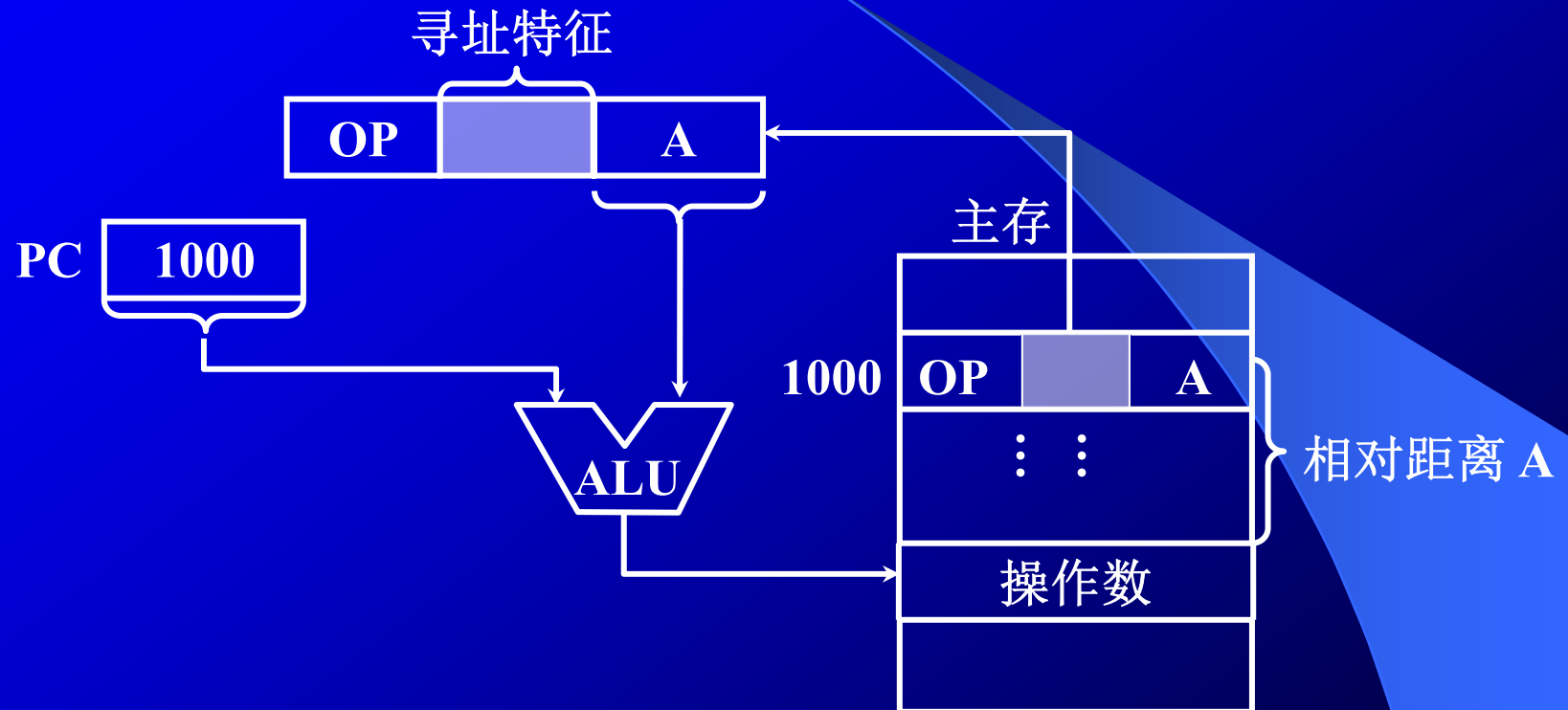
共 8 条指令

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

	LDA	# 0	
	LDX	# 0	
→ M	ADD	X, D	
M+1	INX		
M+2	CPX	# N	✱ 相对寻址特征
— M+3	BNE	M → ✱ - 3	
	DIV	# N	
	STA	ANS	

M 随程序所在存储空间的位置不同而不同

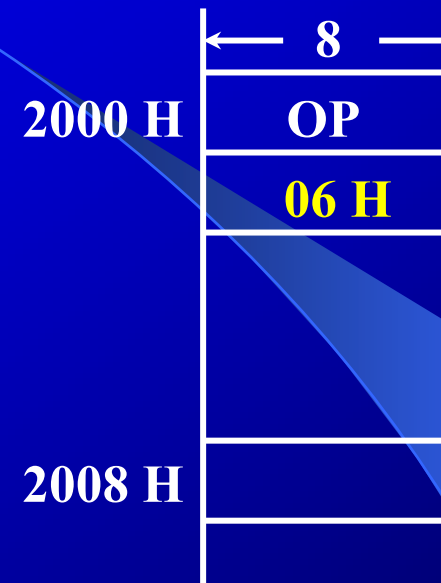
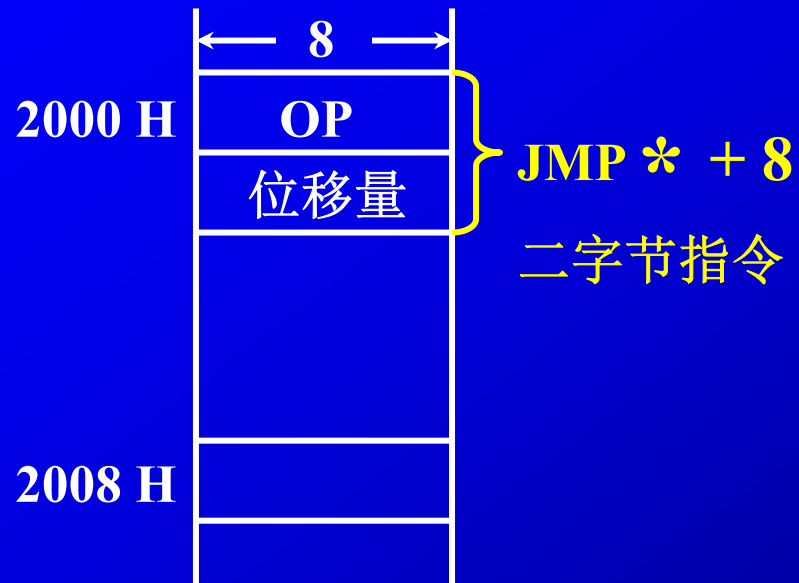
而指令 **BNE ✱-3** 与 指令 **ADD X, D** 相对位移量不变

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 = 6H**

10. 堆栈寻址

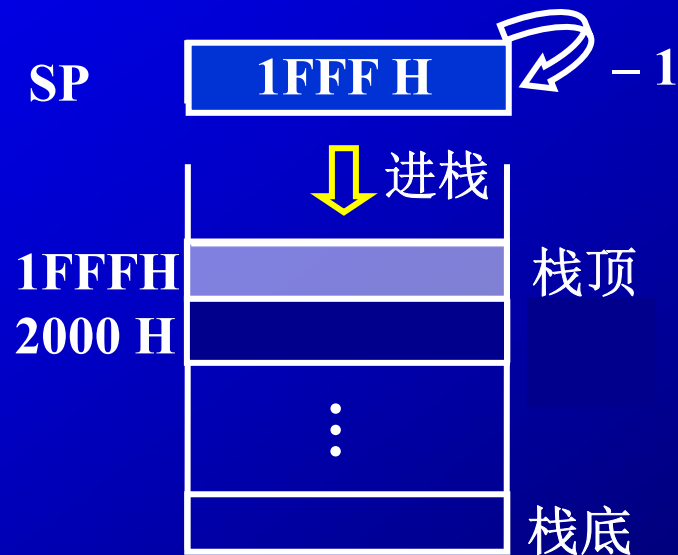
7.3

(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

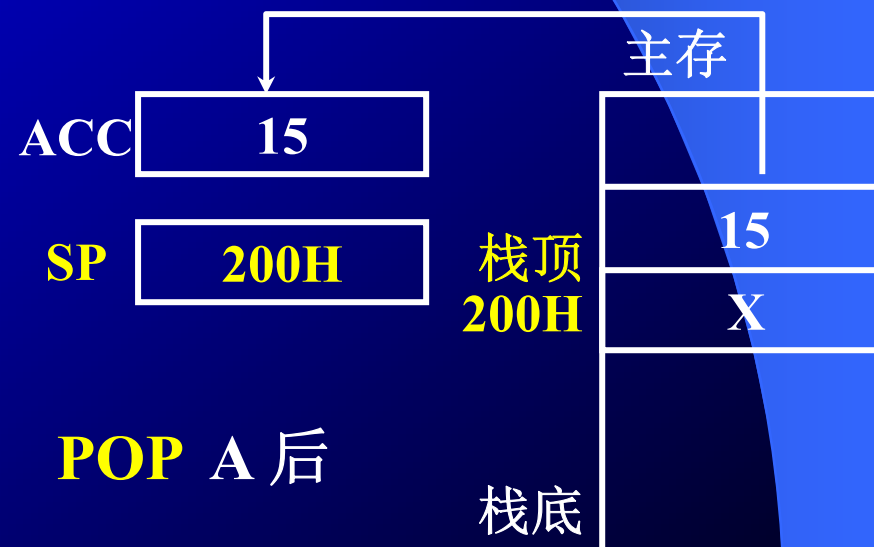
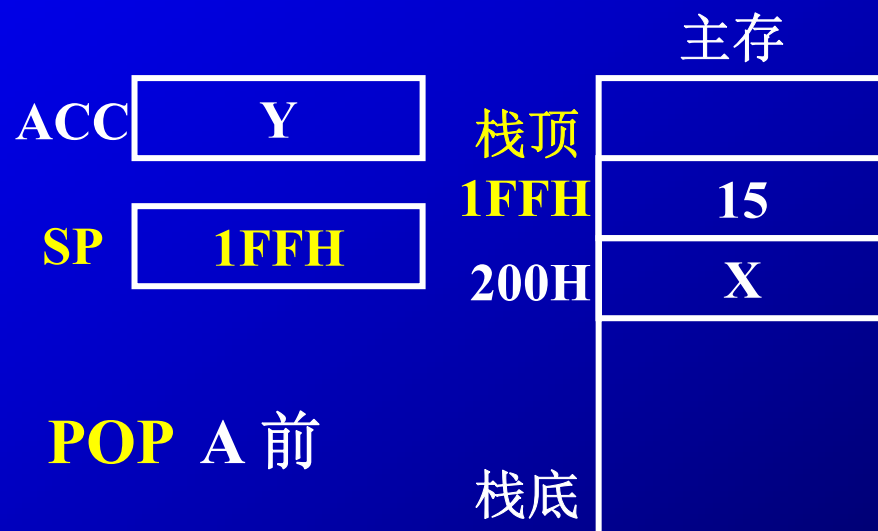
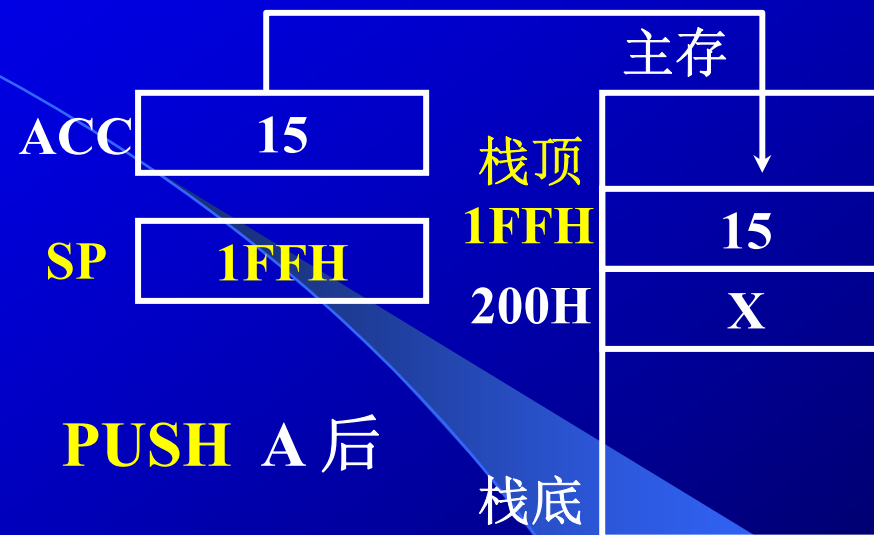
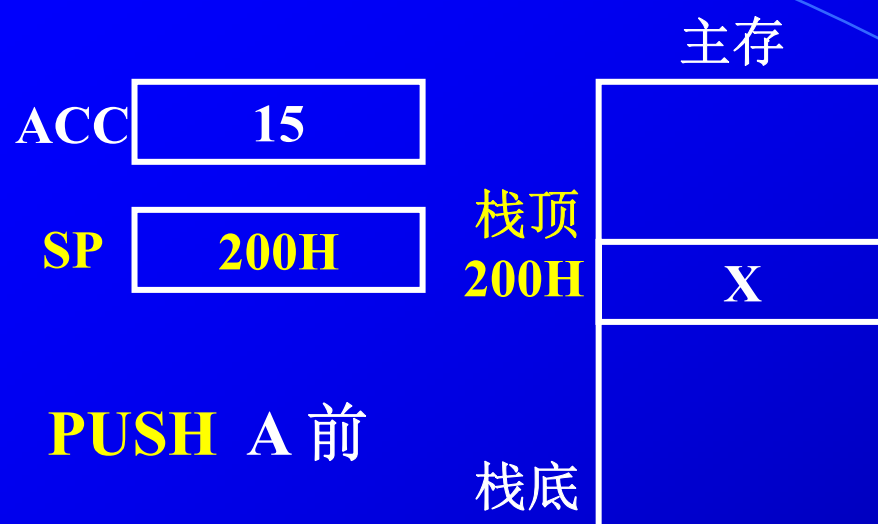
先进后出（一个入出口） 栈顶地址 由 **SP** 指出

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 $(SP) + 4 \longrightarrow SP$

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

指令格式

指令字长、操作码位数

寻址方式、是否采用扩展操作码

地址码位数、地址个数

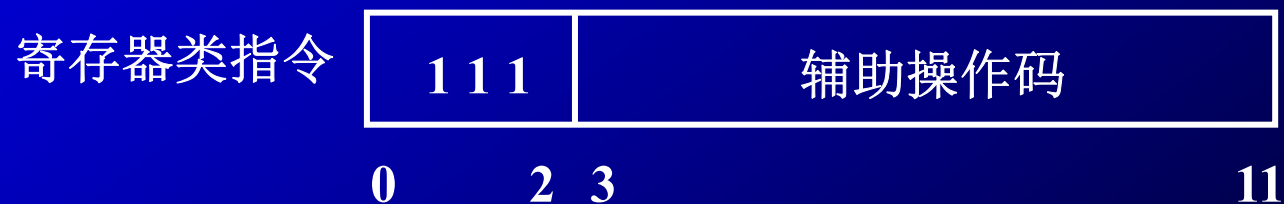
寻址方式

寄存器个数

二、指令格式举例

7.4

1. PDP-8 指令字长固定 12 位



2. PDP-11

7.4

指令字长有 16 位、32 位、48 位三种



16

零地址 (16 位)

扩展操作码技术



10

6

一地址 (16 位)



4

6

6

二地址 $R - R$ (16 位)



10

6

16

二地址 $R - M$ (32 位)



4

6

6

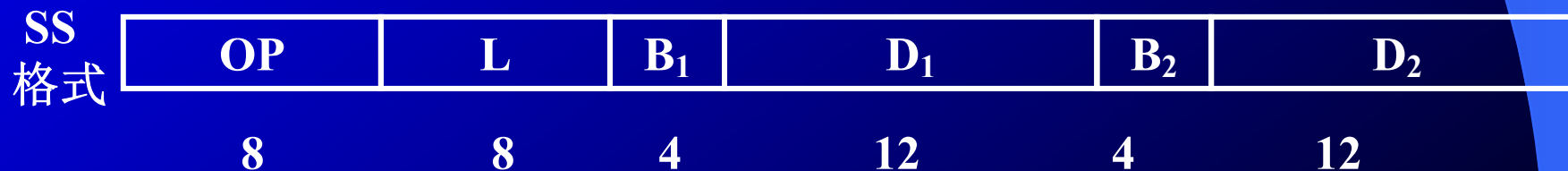
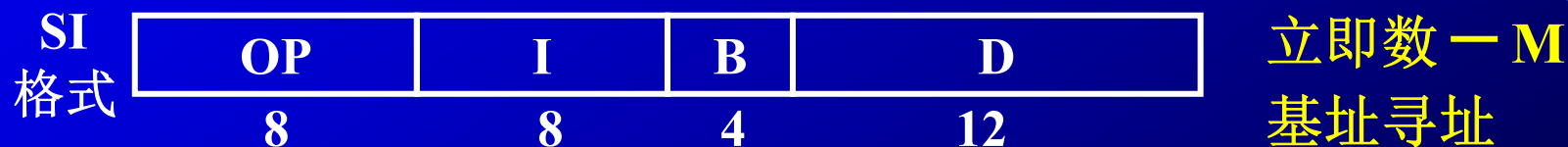
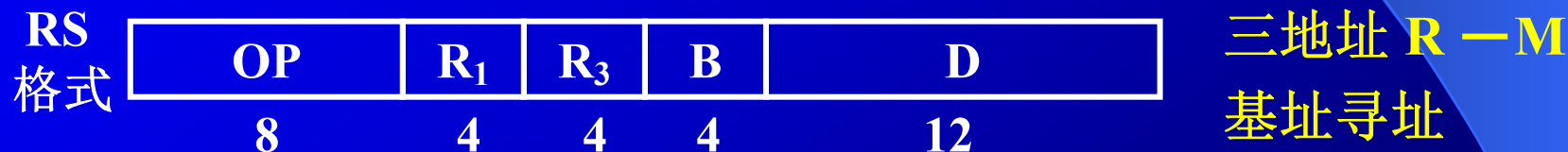
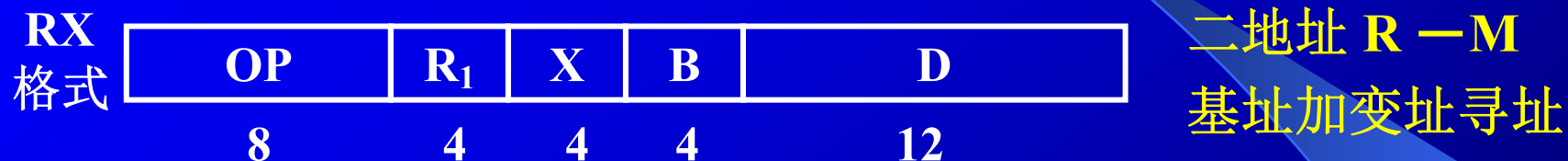
16

16

二地址 $M - M$ (48 位)

3. IBM 360

7.4



二地址 **M—M** 基址寻址

4. Intel 8086

7.4

(1) 指令字长 1~6 个字节

INC AX 1 字节

MOV WORD PTR[0204], 0138H 6 字节

(2) 地址格式

零地址	NOP	1 字节	
一地址	CALL 段间调用	5 字节	
	CALL 段内调用	3 字节	
二地址	ADD AX, BX	2 字节	寄存器 — 寄存器
	ADD AX, 3048H	3 字节	寄存器 — 立即数
	ADD AX, [3048H]	4 字节	寄存器 — 存储器

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

- 选用使用频率较高的一些 **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**
- 只有 **LOAD / STORE** 指令访存
- 流水技术 一个**时钟周期** 内完成一条指令
- **组合逻辑** 实现控制器
- 多个 通用 **寄存器**
- 采用 **优化** 的 **编译** 程序