

计算机组成原理 (第七讲)



厦门大学信息学院软件工程系 曾文华
2021年5月19日



第3篇 中央处理器

第 6 章 计算机的运算方法

第 7 章 指令系统

第 8 章 CPU 的结构和功能



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



7.1 机器指令

一、指令的一般格式

二、指令字长



一、指令的一般格式

操作码字段	地址码字段
-------	-------

1. 操作码 反映机器做什么操作

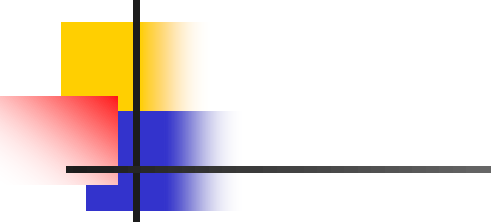
(1) 长度固定

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
PDP-11、Intel 8086、80386等

80x86汇编语言程序



```
CODE SEGMENT USE16
    ASSUME CS:CODE,DS:DATA
START:
    MOV AX,DATA
    MOV DS,AX

    MOV BX,16

    MUL BX

    ADD AX,OFFSET GDT
    ADC DX,0
    MOV WORD PTR VGDTR.BASE,AX
    MOV WORD PTR VGDTR.BASE+2,DX

    MOV AX,CS
    MUL BX

    MOV CODE_DES.BASEL,AX
    MOV CODE_DES.BASEM,DL
    MOV CODE_DES.BASEH,DH
```

80x86指令系统（指令集）

一、数据传输指令

1. 通用数据传送指令

1	MOV	传送字或字节。
2	MOVSX	先符号扩展,再传送。
3	MOVZX	先零扩展,再传送。
4	PUSH	把字压入堆栈。
5	POP	把字弹出堆栈。
6	PUSHA	把AX,CX,DX,BX,SP,BP,SI,DI依次压入堆栈。
7	POPA	把DI,SI,BP,SP,BX,DX,CX,AX依次弹出堆栈。
8	PUSHAD	把EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI依次压入堆栈。
9	POPAD	把EDI,ESI,EBP,ESP,EBX,EDX,ECX,EAX依次弹出堆栈。
10	BSWAP	交换32位寄存器里字节的顺序
11	XCHG	交换字或字节。(至少有一个操作数为寄存器,段寄存器不可作为操作数)
12	CMPXCHG	比较并交换操作数。(第二个操作数必须为累加器AL/AX/EAX)
13	XADD	先交换再累加。(结果在第一个操作数里)
14	XLAT	字节查表转换。----BX指向一张256字节的表的起点,AL为表的索引值(0-255,即0-FFH);返回AL为查表结果。(I

2. 输入输出端口传送指令.

- | | | |
|---|-----|---|
| 1 | IN | I/O端口输入. (语法: IN 累加器, {端口号 DX}) |
| 2 | OUT | I/O端口输出. (语法: OUT {端口号 DX},累加器) 输入输出端口由立即方式指定时, 其范围是 0-255; |

3. 目的地址传送指令.

- | | | |
|---|-----|--|
| 1 | LEA | 装入有效地址.例: LEA DX,string ;把偏移地址存到DX. |
| 2 | LDS | 传送目标指针,把指针内容装入DS.例: LDS SI,string ;把段地址:偏移地址存到DS:SI. |
| 3 | LES | 传送目标指针,把指针内容装入ES.例: LES DI,string ;把段地址:偏移地址存到ES:DI. |
| 4 | LFS | 传送目标指针,把指针内容装入FS.例: LFS DI,string ;把段地址:偏移地址存到FS:DI. |
| 5 | LGS | 传送目标指针,把指针内容装入GS.例: LGS DI,string ;把段地址:偏移地址存到GS:DI. |
| 6 | LSS | 传送目标指针,把指针内容装入SS.例: LSS DI,string ;把段地址:偏移地址存到SS:DI. |

4. 标志传送指令.

- | | | |
|---|-------|-----------------------|
| 1 | LAHF | 标志寄存器传送,把标志装入AH. |
| 2 | SAHF | 标志寄存器传送,把AH内容装入标志寄存器. |
| 3 | PUSHF | 标志入栈. |
| 4 | POPF | 标志出栈. |
| 5 | PUSHD | 32位标志入栈. |
| 6 | POPD | 32位标志出栈. |

二、算术运算指令

1	ADC	带进位加法。
2	INC	加 1。
3	AAA	加法的ASCII码调整。
4	DAA	加法的十进制调整。
5	SUB	减法。
6	SBB	带借位减法。
7	DEC	减 1。
8	NEG	求反(以 0 减之)。
9	CMP	比较。(两操作数作减法,仅修改标志位,不回送结果)。
10	AAS	减法的ASCII码调整。
11	DAS	减法的十进制调整。
12	MUL	无符号乘法。结果回送AH和AL(字节运算),或DX和AX(字运算),
13	IMUL	整数乘法。结果回送AH和AL(字节运算),或DX和AX(字运算),
14	AAM	乘法的ASCII码调整。
15	DIV	无符号除法。结果回送:商回送AL,余数回送AH,(字节运算);或 商回送AX,余数回送DX,(字运算)。
16	IDIV	整数除法。结果回送:商回送AL,余数回送AH,(字节运算);或 商回送AX,余数回送DX,(字运算)。
17	AAD	除法的ASCII码调整。
18	CBW	字节转换为字。(把AL中字节的符号扩展到AH中去)
19	CWD	字转换为双字。(把AX中的字的符号扩展到DX中去)
20	CWDE	字转换为双字。(把AX中的字符符号扩展到EAX中去)
21	CDQ	双字扩展。(把EAX中的字的符号扩展到EDX中去)

三、逻辑运算指令

1	AND	与运算。
2	OR	或运算。
3	XOR	异或运算。
4	NOT	取反。
5	TEST	测试。(两操作数作与运算,仅修改标志位,不回送结果)。
6	SHL	逻辑左移。
7	SAL	算术左移。(=SHL)
8	SHR	逻辑右移。
9	SAR	算术右移。(=SHR)
10	ROL	循环左移。
11	ROR	循环右移。
12	RCL	通过进位的循环左移。
13	RCR	通过进位的循环右移。
14	以上八种移位指令,其移位次数可达255次。	
15	移位一次时,可直接用操作码。如 SHL AX,1。	
16	移位>1次时,则由寄存器CL给出移位次数。	
17	如 MOV CL,04 SHL AX,CL	

四、串指令

1	DS:SI	源串段寄存器 :源串变址.
2	ES:DI	目标串段寄存器:目标串变址.
3	CX	重复次数计数器.
4	AL/AX	扫描值.
5	D标志	0表示重复操作中SI和DI应自动增量; 1表示应自动减量.
6	Z标志	用来控制扫描或比较操作的结束.
7	MOVS	串传送.(MOVSB 传送字符. MOVSW 传送字. MOVSD 传送双字.)
8	CMPS	串比较.(CMPSB 比较字符. CMPSW 比较字.)
9	SCAS	串扫描.把AL或AX的内容与目标串作比较,比较结果反映在标志位.
10	LODS	装入串.把源串中的元素(字或字节)逐一装入AL或AX中.(LODSB 传送字符. LODSW 传送字. LODSD 传送双字.)
11	STOS	保存串.是LODS的逆过程.
12	REP	当CX/ECX<>0时重复.
13	REPE/REPZ	当ZF=1或比较结果相等,且CX/ECX<>0时重复.
14	REPNE/REPNZ	当ZF=0或比较结果不相等,且CX/ECX<>0时重复.
15	REPC	当CF=1且CX/ECX<>0时重复.
16	REPNC	当CF=0且CX/ECX<>0时重复.

五、程序转移指令

1. 无条件转移指令 (长转移)

1	JMP	无条件转移指令
2	CALL	过程调用
3	RET/RETF	过程返回。

2. 条件转移指令 (短转移, -128到+127的距离内)(当且仅当 $(SF \oplus OF)=1$ 时, $OP1 < OP2$)

1	JA/JNBE	不小于或不等于时转移。
2	JAE/JNB	大于或等于转移。
3	JB/JNAE	小于转移。
4	JBE/JNA	小于或等于转移。
5	以上四条,测试无符号整数运算的结果(标志C和Z)。	
6	JG/JNLE	大于转移。
7	JGE/JNL	大于或等于转移。
8	JL/JNGE	小于转移。
9	JLE/JNG	小于或等于转移。
10	以上四条,测试带符号整数运算的结果(标志S, O和Z)。	
11	JE/JZ	等于转移。
12	JNE/JNZ	不等于时转移。
13	JC	有进位时转移。
14	JNC	无进位时转移。
15	JNO	不溢出时转移。
16	JNP/JPO	奇偶性为奇数时转移。
17	JNS	符号位为 "0" 时转移。
18	JO	溢出转移。
19	JP/JPE	奇偶性为偶数时转移。
20	JS	符号位为 "1" 时转移。

3. 循环控制指令(短转移)

1	LOOP	CX不为零时循环。
2	LOOPE/LOOPZ	CX不为零且标志Z=1时循环。
3	LOOPNE/LOOPNZ	CX不为零且标志Z=0时循环。
4	JCXZ	CX为零时转移。
5	JECXZ	ECX为零时转移。

4. 中断指令

1	INT	中断指令
2	INTO	溢出中断
3	IRET	中断返回

5. 处理器控制指令

1	HLT	处理器暂停，直到出现中断或复位信号才继续。
2	WAIT	当芯片引线TEST为高电平时使CPU进入等待状态。
3	ESC	转换到外处理器。
4	LOCK	封锁总线。
5	NOP	空操作。
6	STC	置进位标志位。
7	CLC	清进位标志位。
8	CMC	进位标志取反。
9	STD	置方向标志位。
10	CLD	清方向标志位。
11	STI	置中断允许位。
12	CLI	清中断允许位。

六、伪指令

1	DW	定义字(2字节)。
2	PROC	定义过程。
3	ENDP	过程结束。
4	SEGMENT	定义段。
5	ASSUME	建立段寄存器寻址。
6	ENDS	段结束。
7	END	程序结束。

七、处理机控制指令：标志处理指令

1	CLC	进位位置0指令
2	CMC	进位位求反指令
3	STC	进位位置为1指令
4	CLD	方向标志置1指令
5	STD	方向标志位置1指令
6	CLI	中断标志置0指令
7	STI	中断标志置1指令
8	NOP	无操作
9	HLT	停机
10	WAIT	等待
11	ESC	换码
12	LOCK	封锁

ARM汇编语言程序

- 程序代码如下：

伪操作

```
.global _start                @声明全局变量 “_start”
.text                        @代码段
_start:
    MOV R8, #20                @低32位初始化为20
    MOV R9, #0                @高32位初始化为0    [R9:R8]=20
    SUB R0, R8, #1            @初始化计数器    R0=19
Loop:
    MOV R1, R9                @暂存高位值
    UMULL R8, R9, R0, R8      @[R9: R8]=R0*R8
    MLA R9, R1, R0, R9        @R9=R1*R0+R9
    SUBS R0, R0, #1           @计数器递减
    BNE Loop                  @计数器不为0，继续循环
Stop:
    B Stop
.end                          @源文件结束（程序结束）
```

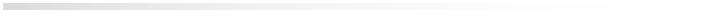
伪操作

ARM指令系统（指令集）

ARM 指令列表

ADC	带进位的32位数加法
ADD	32位数相加
AND	32位数的逻辑与
B	在32M空间内的相对跳转指令
BEQ	相等则跳转（Branch if Equal）
BNE	不相等则跳转（Branch if Not Equal）
BGE	大于或等于跳转（Branch if Greater than or Equal）
BGT	大于跳转（Branch if Greater Than）
BIC	32位数的逻辑位清零
BKPT	断点指令
BL	带链接的相对跳转指令
BLE	小于或等于跳转（Branch if Less than or Equal）
BLEQ	带链接等于跳转（Branch with Link if Equal）
BLLT	带链接小于跳转（Branch with Link if Less Than）
BLT	小于跳转（Branch if Less Than）
BLX	带链接的切换跳转
BX	切换跳转
CDP CDP2	协处理器数据处理操作
CLZ	零计数
CMN	比较两个数的相反数
CMP	32位数比较

EOR	32位逻辑异或
LDC LDC2	从协处理器取一个或多个32位值
LDM	从内存送多个32位字到ARM寄存器
LDR	从虚拟地址取一个单个的32位值
MCR MCR2 MCRR	从寄存器送数据到协处理器
MLA	32位乘累加
MOV	传送一个32位数到寄存器
MRC MRC2 MRRC	从协处理器传送数据到寄存器
MRS	把状态寄存器的值送到通用寄存器
MSR	把通用寄存器的值传送到状态寄存器
MUL	32位乘
MVN	把一个32位数的逻辑“非”送到寄存器
ORR	32位逻辑或
PLD	预装载提示指令
QADD	有符号32位饱和加
QDADD	有符号双32位饱和加
QSUB	有符号32位饱和减
QDSUB	有符号双32位饱和减
RSB	逆向32位减法
RSC	带进位的逆向32法减法
SBC	带进位的32位减法
SMLAxy	有符号乘累加(16位*16位)+32位=32位



SMLAL	64位有符号乘累加((32位*32位)+64位=64位)
SMALxy	64位有符号乘累加((32位*32位)+64位=64位)
SMLAWy	号乘累加((32位*16位)>>16位)+32位=32位
SMULL	64位有符号乘累加(32位*32位)=64位
SMULxy	有符号乘(16位*16位=32位)
SMULWy	有符号乘(32位*16位>>16位=32位)
STC STC2	从协处理器中把一个或多个32位值存到内存
STM	把多个32位的寄存器值存放到内存
STR	把寄存器的值存到一个内存的虚地址内
SUB	32位减法
SWI	软中断
SWP	把一个字或者一个字节和一个寄存器值交换
TEQ	等值测试
TST	位测试
UMLAL	64位无符号乘累加((32位*32位)+64位=64位)
UMULL	64位无符号乘累加(32位*32位)=64位

(3) 扩展操作码技术

操作码的位数随地址数的减少而增加

	OP	A ₁	A ₂	A ₃	指令的总长度固定（16位）
4 位操作码	0000 0001 ⋮ 1110	A ₁ A ₁ ⋮ A ₁	A ₂ A ₂ ⋮ A ₂	A ₃ A ₃ ⋮ A ₃	最多15条三地址指令
8 位操作码	1111 1111 ⋮ 1111	0000 0001 ⋮ 1110	A ₂ A ₂ ⋮ A ₂	A ₃ A ₃ ⋮ A ₃	最多15条二地址指令
12 位操作码	1111 1111 ⋮ 1111	1111 1111 ⋮ 1111	0000 0001 ⋮ 1110	A ₃ A ₃ ⋮ A ₃	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

- **例7.1:** 假设指令字长为**16**位, 操作数的地址码为**6**位, 指令有零地址、一地址、二地址等**3**种格式。

(1) 设操作码固定, 若零地址指令有**P**种、一地址指令有**Q**种, 则二地址指令有多少种?

(2) 采用扩展操作码技术, 若二地址指令有**X**种, 零地址指令有**Y**种, 则一地址指令最多有几种?

- **例7.1:** 假设指令字长为**16**位，操作数的地址码为**6**位，指令有零地址、一地址、二地址等**3**种格式。

(1) 设操作码固定，若零地址指令有**P**种、一地址指令有**Q**种，则二地址指令有多少种？

(2) 采用扩展操作码技术，若二地址指令有**X**种，零地址指令有**Y**种，则一地址指令最多有几种？

- **解:** (1) 操作码固定，且只有**4**位 (**16**位指令字长=**4**位操作码+**6**位地址码+**6**位地址码)，故共有 **$2^4=16$** 种指令。

则，剩余二地址指令最多为: **$16-P-Q$**

零地址指令:	XXXX	XXXXXX	XXXXXX	P种
一地址指令:	XXXX	XXXXXX	A1-A6	Q种
二地址指令:	XXXX	A1-A6	A1-A6	$16-P-Q$种



(2) 采用扩展操作码技术，则：

二地址指令： **XXXX** **A1-A6** **A1-A6** **X种**

一地址指令： **1111** **XXXXXX** **A1-A6** **M种?**

零地址指令： **1111** **111111** **XXXXXX** **Y种**

一地址指令最多 $(2^4 - X) * 2^6$ **M种?**

零地址指令最多 $[(2^4 - X) * 2^6 - M] * 2^6$ **Y种**

因为有： $Y = [(2^4 - X) * 2^6 - M] * 2^6$

故求得： $M = (2^4 - X) * 2^6 - Y * 2^{-6}$

例如：二地址指令有**4种**（**X**），零地址指令有**64种**（**Y**），则一地址指令最多有**767种**（**M**）

- 0000 A1-A6 A1-A6
- 0001 A1-A6 A1-A6
- 0010 A1-A6 A1-A6
- 0011 A1-A6 A1-A6

X=4种

- 0100 000000 A1-A6
- A1-A6
- 0100 111111 A1-A6

64

$$64 \times 11 + 63 = 767$$

- A1-A6
- 1110 000000 A1-A6
- A1-A6
- 1110 111111 A1-A6

64

则M=767种

- 1111 000000 A1-A6
- A1-A6
- 1111 111110 A1-A6

63

$$\begin{aligned} M &= (2^4 - X) \times 2^6 - Y \times 2^{-6} \\ &= (16 - 4) \times 64 - 1 \\ &= 767 \end{aligned}$$

- 1111 111111 000000
- 000000
- 1111 111111 111111

Y=64种

64(4)

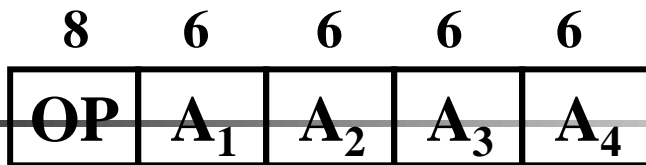
64(14)

63(15)

2. 地址码

地址可以是主存的地址、寄存器的地址、I/O的地址

(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

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

设指令字长为 32 位

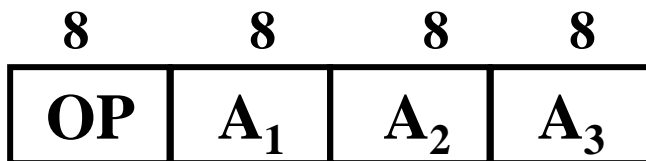
操作码固定为 8 位

4 次访存

寻址范围 $2^6 = 64$

若 PC 代替 A₄ **A4** 则可省去

(2) 三地址



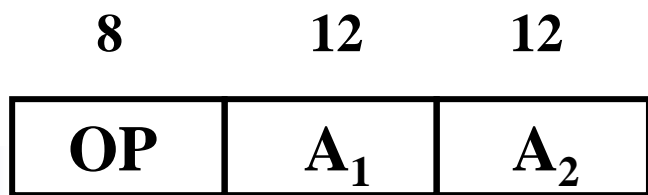
$(A_1) \text{ OP } (A_2) \longrightarrow A_3$

4 次访存

寻址范围 $2^8 = 256$

若 A₃ 用 A₁ 或 A₂ 代替 **A3** 则可省去

(3) 二地址



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

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

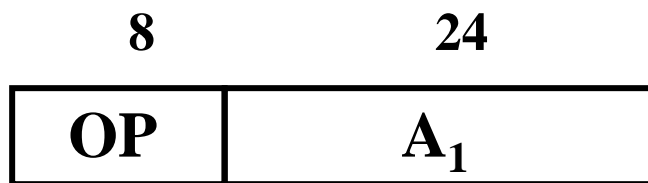
若结果存于 ACC 3次访存

4 次访存

寻址范围 $2^{12} = 4 \text{ K}$

若ACC 代替 A_1 (或 A_2)

(4) 一地址



$(\text{ACC}) \text{ OP } (A_1) \longrightarrow \text{ACC}$

2 次访存

寻址范围 $2^{24} = 16 \text{ M}$

(5) 零地址

NOP 指令

无地址码

小结

如PC（程序计数器）代替A4

ACC代替A1或A2

➤ 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

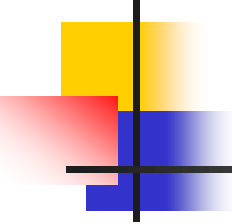
三地址 **OP R_1 , R_2 , R_3**

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

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

存储字长：存储单元的位数

机器字长：CPU的运算器的位数

指令字长 = 存储字长 = 机器字长

2. 指令字长 可变

按字节的倍数变化，8位、16位、....



7.2 操作数类型和操作类型

一、操作数类型

二、数据在存储器中的存放方式

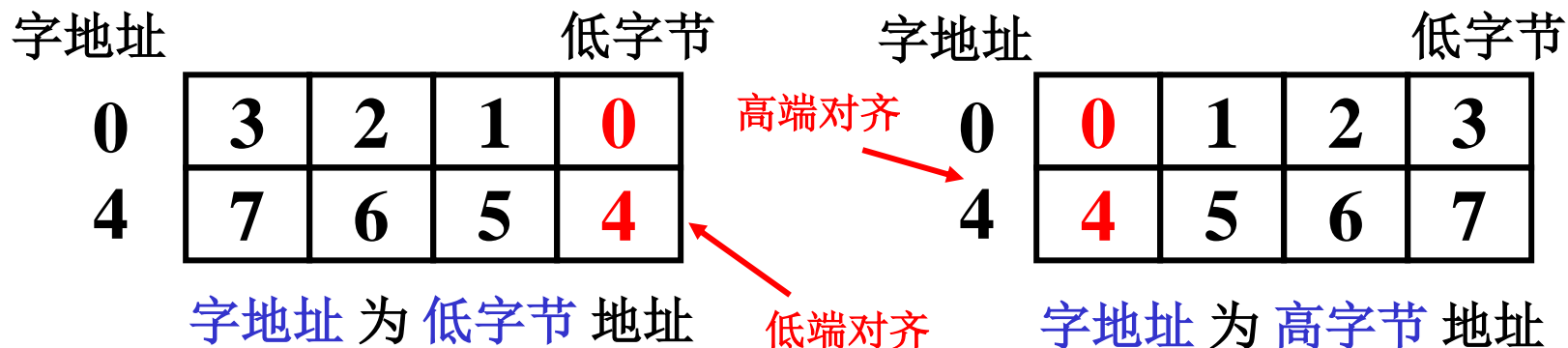
三、操作类型

一、操作数类型

指令 = 操作码 + 地址码 (操作数)

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(第四章P73)



存储器中的数据存放（存储字长为32位）

边界对准

地址（十进制）

字（地址 0）				32位
字（地址 4）				
字节（地址11）	字节（地址10）	字节（地址 9）	字节（地址 8）	
字节（地址15）	字节（地址14）	字节（地址13）	字节（地址12）	
半字（地址18）✓		半字（地址16）✓ 16位		
半字（地址22）✓		半字（地址20）✓		
双字（地址24）▲				
双字				64位
双字（地址32）▲				
双字				

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

存储器中的数据存放（存储字长为32位）

边界对准

地址（十进制）

字（地址 0）				32位
字（地址 4）				
字节（地址11）	字节（地址10）	字节（地址 9）	字节（地址 8）	
字节（地址15）	字节（地址14）	字节（地址13）	字节（地址12）	
半字（地址18）✓		半字（地址16）✓ 16位		
半字（地址22）✓		半字（地址20）✓		
边界未对准时，从存储器中读写1个字的时间，可能会比边界对准时要增加1倍				
双字（地址32）				
双字				

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

操作码的类型（指令的类型）

1. 数据传送

源

目的

例如

寄存器

寄存器

MOVE

寄存器

存储器

STORE

MOVE

PUSH

存储器

寄存器

LOAD

MOVE

POP

存储器

存储器

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**

结果为负转 (**N** = 1) **JN**

结果为奇偶转 (**P** = 1) **JP**

跳过一条指令 **SKP**

Zero
Overflow
Carry
Negative
Parity

完成触发器D

300

⋮

305

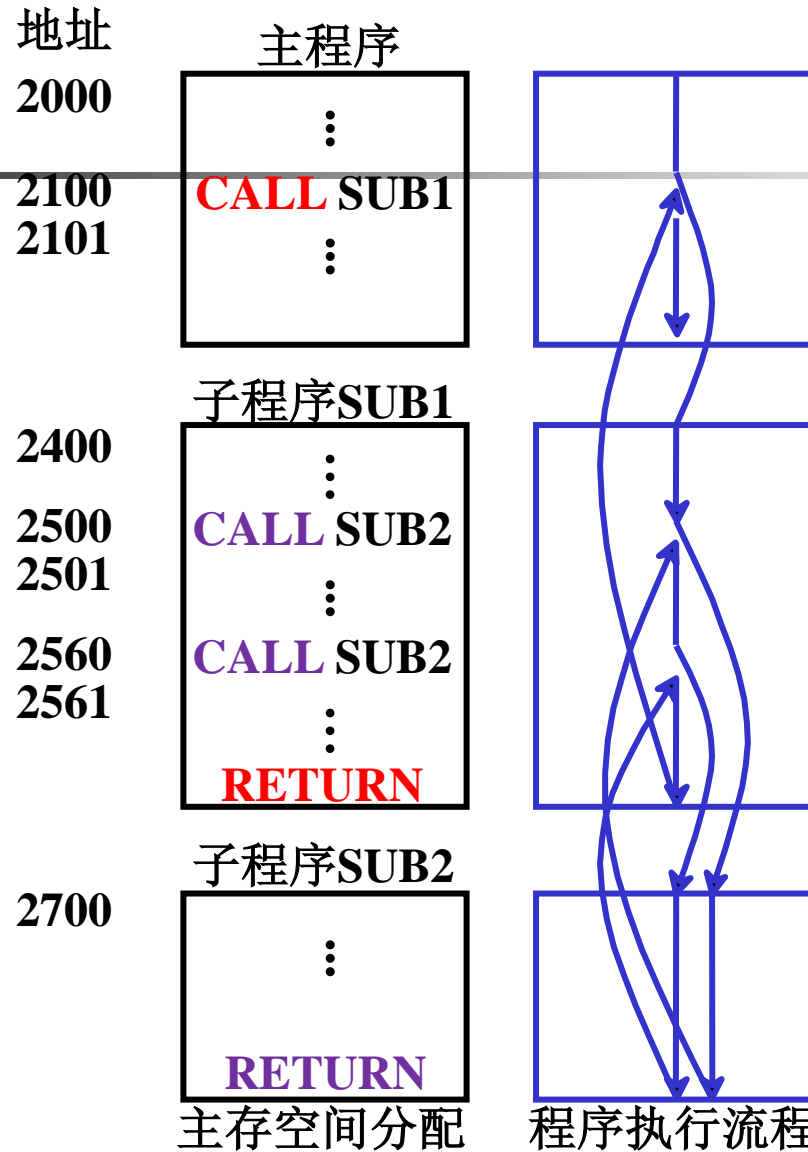
306

307

SKP DZ D = 0 则跳

如果D=0，则跳过
一条指令，转到307

(3) 调用和返回



(4) 陷阱 (Trap) 与陷阱指令

陷阱：意外事故的中断
(如除数为0)

有二类陷阱指令：

- **第一类：**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**)

- **第二类：**设置供用户使用的陷阱指令 (“访管” 指令)

如 8086 INT TYPE 软中断

INT 21H

提供给用户使用的陷阱指令，完成系统调用

5. 输入输出

输入 端口地址 \longrightarrow CPU 的寄存器

如 IN AX, n IN AL, DX

输出 CPU 的寄存器 \longrightarrow 端口地址

如 OUT n, AL OUT DX, AL



6.其他指令

等待指令、停机指令、空操作指令(**NOP**)、开中断指令(**EI**)、关中断指令(**DI**)、置条件码指令等

80x86指令系统（指令集）

一、数据传输指令

数据传送

1. 通用数据传送指令

1	MOV	传送字或字节。
2	MOVSX	先符号扩展,再传送。
3	MOVZX	先零扩展,再传送。
4	PUSH	把字压入堆栈。
5	POP	把字弹出堆栈。
6	PUSHA	把AX,CX,DX,BX,SP,BP,SI,DI依次压入堆栈。
7	POPA	把DI,SI,BP,SP,BX,DX,CX,AX依次弹出堆栈。
8	PUSHAD	把EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI依次压入堆栈。
9	POPAD	把EDI,ESI,EBP,ESP,EBX,EDX,ECX,EAX依次弹出堆栈。
10	BSWAP	交换32位寄存器里字节的顺序
11	XCHG	交换字或字节。(至少有一个操作数为寄存器,段寄存器不可作为操作数)
12	CMPXCHG	比较并交换操作数。(第二个操作数必须为累加器AL/AX/EAX)
13	XADD	先交换再累加。(结果在第一个操作数里)
14	XLAT	字节查表转换。----BX指向一张256字节的表的起点,AL为表的索引值(0-255,即0-FFH);返回AL为查表结果。(I

数据传送

2. 输入输出端口传送指令.

- | | | |
|---|-----|---|
| 1 | IN | I/O端口输入. (语法: IN 累加器, {端口号 DX}) |
| 2 | OUT | I/O端口输出. (语法: OUT {端口号 DX},累加器)输入输出端口由立即方式指定时, 其范围是 0-255; |

输入输出

3. 目的地址传送指令.

- | | | |
|---|-----|--|
| 1 | LEA | 装入有效地址.例: LEA DX,string ;把偏移地址存到DX. |
| 2 | LDS | 传送目标指针,把指针内容装入DS.例: LDS SI,string ;把段地址:偏移地址存到DS:SI. |
| 3 | LES | 传送目标指针,把指针内容装入ES.例: LES DI,string ;把段地址:偏移地址存到ES:DI. |
| 4 | LFS | 传送目标指针,把指针内容装入FS.例: LFS DI,string ;把段地址:偏移地址存到FS:DI. |
| 5 | LGS | 传送目标指针,把指针内容装入GS.例: LGS DI,string ;把段地址:偏移地址存到GS:DI. |
| 6 | LSS | 传送目标指针,把指针内容装入SS.例: LSS DI,string ;把段地址:偏移地址存到SS:DI. |

4. 标志传送指令.

- | | | |
|---|-------|-----------------------|
| 1 | LAHF | 标志寄存器传送,把标志装入AH. |
| 2 | SAHF | 标志寄存器传送,把AH内容装入标志寄存器. |
| 3 | PUSHF | 标志入栈. |
| 4 | POPF | 标志出栈. |
| 5 | PUSHD | 32位标志入栈. |
| 6 | POPD | 32位标志出栈. |

二、算术运算指令

算术逻辑操作

1	ADC	带进位加法。
2	INC	加 1。
3	AAA	加法的ASCII码调整。
4	DAA	加法的十进制调整。
5	SUB	减法。
6	SBB	带借位减法。
7	DEC	减 1。
8	NEG	求反(以 0 减之)。
9	CMP	比较。(两操作数作减法,仅修改标志位,不回送结果)。
10	AAS	减法的ASCII码调整。
11	DAS	减法的十进制调整。
12	MUL	无符号乘法。结果回送AH和AL(字节运算),或DX和AX(字运算),
13	IMUL	整数乘法。结果回送AH和AL(字节运算),或DX和AX(字运算),
14	AAM	乘法的ASCII码调整。
15	DIV	无符号除法。结果回送:商回送AL,余数回送AH,(字节运算);或 商回送AX,余数回送DX,(字运算)。
16	IDIV	整数除法。结果回送:商回送AL,余数回送AH,(字节运算);或 商回送AX,余数回送DX,(字运算)。
17	AAD	除法的ASCII码调整。
18	CBW	字节转换为字。(把AL中字节的符号扩展到AH中去)
19	CWD	字转换为双字。(把AX中的字的符号扩展到DX中去)
20	CWDE	字转换为双字。(把AX中的字符符号扩展到EAX中去)
21	CDQ	双字扩展。(把EAX中的字的符号扩展到EDX中去)

算术逻辑操作

三、逻辑运算指令

1	AND	与运算。
2	OR	或运算。
3	XOR	异或运算。
4	NOT	取反。
5	TEST	测试。(两操作数作与运算,仅修改标志位,不回送结果)。
6	SHL	逻辑左移。
7	SAL	算术左移.(=SHL)
8	SHR	逻辑右移。
9	SAR	算术右移.(=SHR)
10	ROL	循环左移。
11	ROR	循环右移。
12	RCL	通过进位的循环左移。
13	RCR	通过进位的循环右移。
14	以上八种移位指令,其移位次数可达255次。	
15	移位一次时,可直接用操作码。如 SHL AX,1。	
16	移位>1次时,则由寄存器CL给出移位次数。	
17	如 MOV CL,04 SHL AX,CL	

移位操作

四、串指令

1	DS:SI	源串段寄存器 :源串变址.
2	ES:DI	目标串段寄存器:目标串变址.
3	CX	重复次数计数器.
4	AL/AX	扫描值.
5	D标志	0表示重复操作中SI和DI应自动增量; 1表示应自动减量.
6	Z标志	用来控制扫描或比较操作的结束.
7	MOVS	串传送.(MOVSB 传送字符. MOVSW 传送字. MOVSD 传送双字.)
8	CMPS	串比较.(CMPSB 比较字符. CMPSW 比较字.)
9	SCAS	串扫描.把AL或AX的内容与目标串作比较,比较结果反映在标志位.
10	LODS	装入串.把源串中的元素(字或字节)逐一装入AL或AX中.(LODSB 传送字符. LODSW 传送字. LODSD 传送双字.)
11	STOS	保存串.是LODS的逆过程.
12	REP	当CX/ECX<>0时重复.
13	REPE/REPZ	当ZF=1或比较结果相等,且CX/ECX<>0时重复.
14	REPNE/REPNZ	当ZF=0或比较结果不相等,且CX/ECX<>0时重复.
15	REPC	当CF=1且CX/ECX<>0时重复.
16	REPNC	当CF=0且CX/ECX<>0时重复.

五、程序转移指令

转移

1. 无条件转移指令 (长转移)

1	JMP	无条件转移指令
2	CALL	过程调用
3	RET/RETF	过程返回。

2. 条件转移指令 (短转移, -128到+127的距离内)(当且仅当 $(SF \oplus OF)=1$ 时, $OP1 < OP2$)

1	JA/JNBE	不小于或不等于时转移。
2	JAE/JNB	大于或等于转移。
3	JB/JNAE	小于转移。
4	JBE/JNA	小于或等于转移。
5	以上四条,测试无符号整数运算的结果(标志C和Z)。	
6	JG/JNLE	大于转移。
7	JGE/JNL	大于或等于转移。
8	JL/JNGE	小于转移。
9	JLE/JNG	小于或等于转移。
10	以上四条,测试带符号整数运算的结果(标志S, O和Z)。	
11	JE/JZ	等于转移。
12	JNE/JNZ	不等于时转移。
13	JC	有进位时转移。
14	JNC	无进位时转移。
15	JNO	不溢出时转移。
16	JNP/JPO	奇偶性为奇数时转移。
17	JNS	符号位为 "0" 时转移。
18	JO	溢出转移。
19	JP/JPE	奇偶性为偶数时转移。
20	JS	符号位为 "1" 时转移。

3. 循环控制指令(短转移)

1	LOOP	CX不为零时循环。
2	LOOPE/LOOPZ	CX不为零且标志Z=1时循环。
3	LOOPNE/LOOPNZ	CX不为零且标志Z=0时循环。
4	JCXZ	CX为零时转移。
5	JECXZ	ECX为零时转移。

4. 中断指令

1	INT	中断指令
2	INTO	溢出中断
3	IRET	中断返回

5. 处理器控制指令

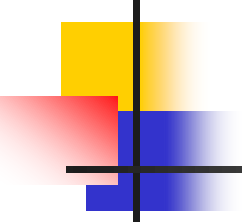
1	HLT	处理器暂停，直到出现中断或复位信号才继续。
2	WAIT	当芯片引线TEST为高电平时使CPU进入等待状态。
3	ESC	转换到外处理器。
4	LOCK	封锁总线。
5	NOP	空操作。
6	STC	置进位标志位。
7	CLC	清进位标志位。
8	CMC	进位标志取反。
9	STD	置方向标志位。
10	CLD	清方向标志位。
11	STI	置中断允许位。
12	CLI	清中断允许位。

六、伪指令

1	DW	定义字(2字节)。
2	PROC	定义过程。
3	ENDP	过程结束。
4	SEGMENT	定义段。
5	ASSUME	建立段寄存器寻址。
6	ENDS	段结束。
7	END	程序结束。

七、处理机控制指令：标志处理指令

1	CLC	进位位置0指令
2	CMC	进位位求反指令
3	STC	进位位置为1指令
4	CLD	方向标志置1指令
5	STD	方向标志位置1指令
6	CLI	中断标志置0指令
7	STI	中断标志置1指令
8	NOP	无操作
9	HLT	停机
10	WAIT	等待
11	ESC	换码
12	LOCK	封锁



7.3 寻址方式

一、指令寻址

二、数据寻址



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

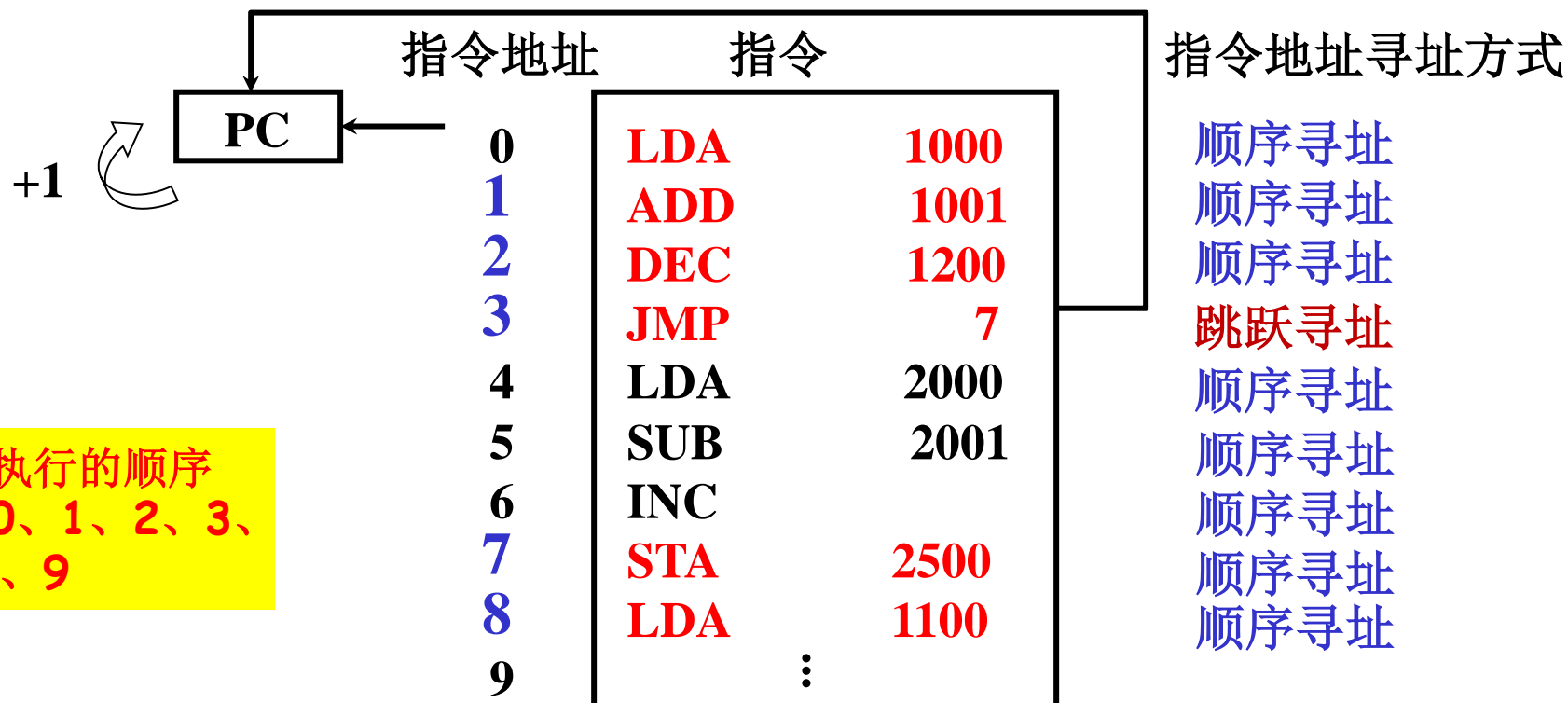
寻址方式 {

- 指令寻址（确定下一条指令的地址）
- 数据寻址（确定操作数的地址）

一、指令寻址

1、顺序 $(PC) + 1 \longrightarrow PC$

2、跳跃 由转移指令指出



二、数据寻址

操作码(OP)	寻址特征	形式地址 A
---------	------	--------

形式地址A 指令字中的地址

有效地址EA 操作数的真实地址

Effective Address

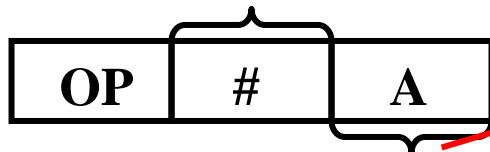
约定 指令字长 = 存储字长 = 机器字长

1. 立即寻址

形式地址 A 就是操作数

MOV AX, 2300H

立即寻址特征



立即数 可正可负 补码

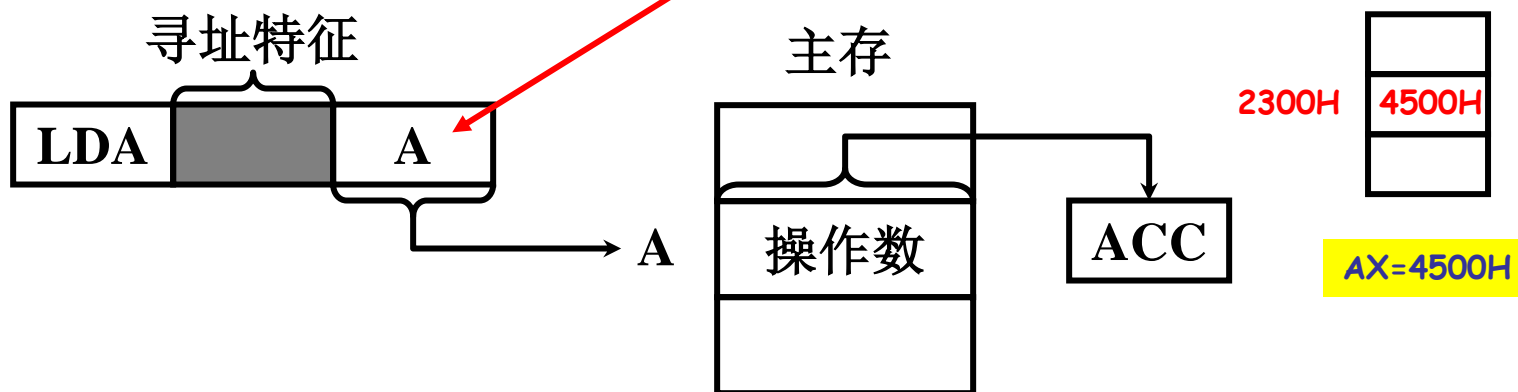
- 指令执行阶段不访存
- A 的位数限制了立即数的范围

2. 直接寻址

MOV AX, [2300H]

EA = A

有效地址由形式地址直接给出



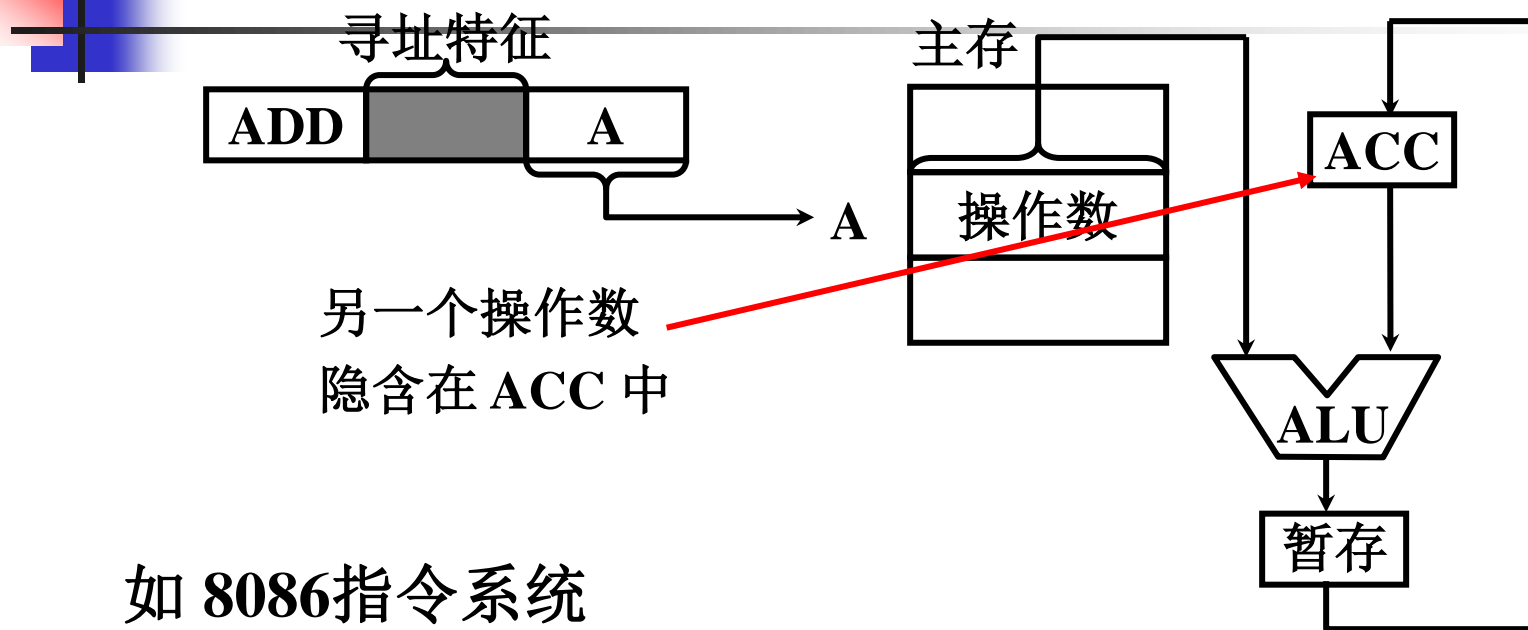
- 执行阶段访问一次存储器
- A 的位数决定了该指令操作数的寻址范围
- 操作数的地址不易修改（必须修改A）

3. 隐含寻址

操作数地址隐含在操作码中

ADD [2300H]

相当于: ADD AX,[2300H]



如 8086指令系统

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

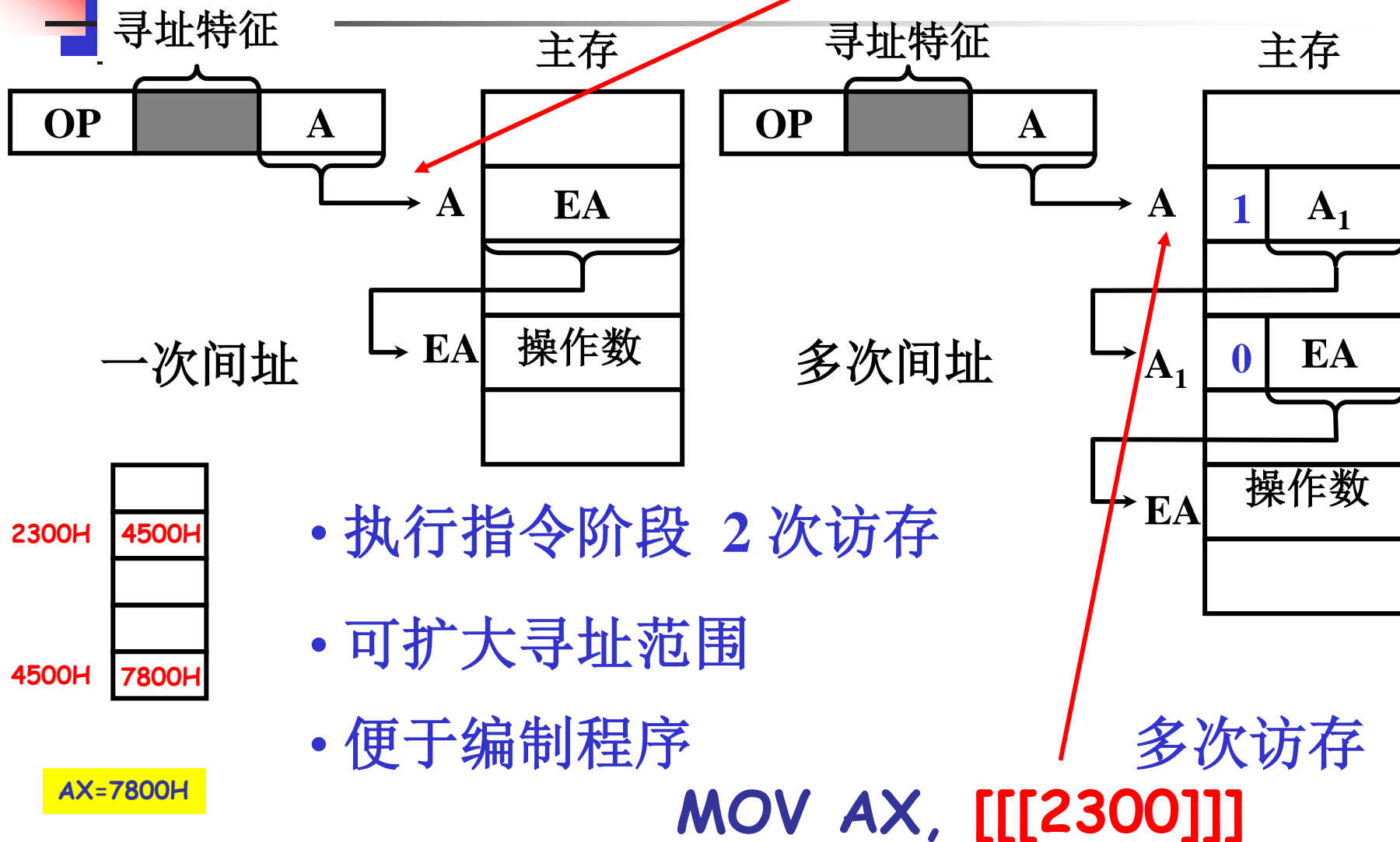
目的操作数的地址隐含在 DI 中

- 指令字中少了一个地址字段, 可缩短指令字长

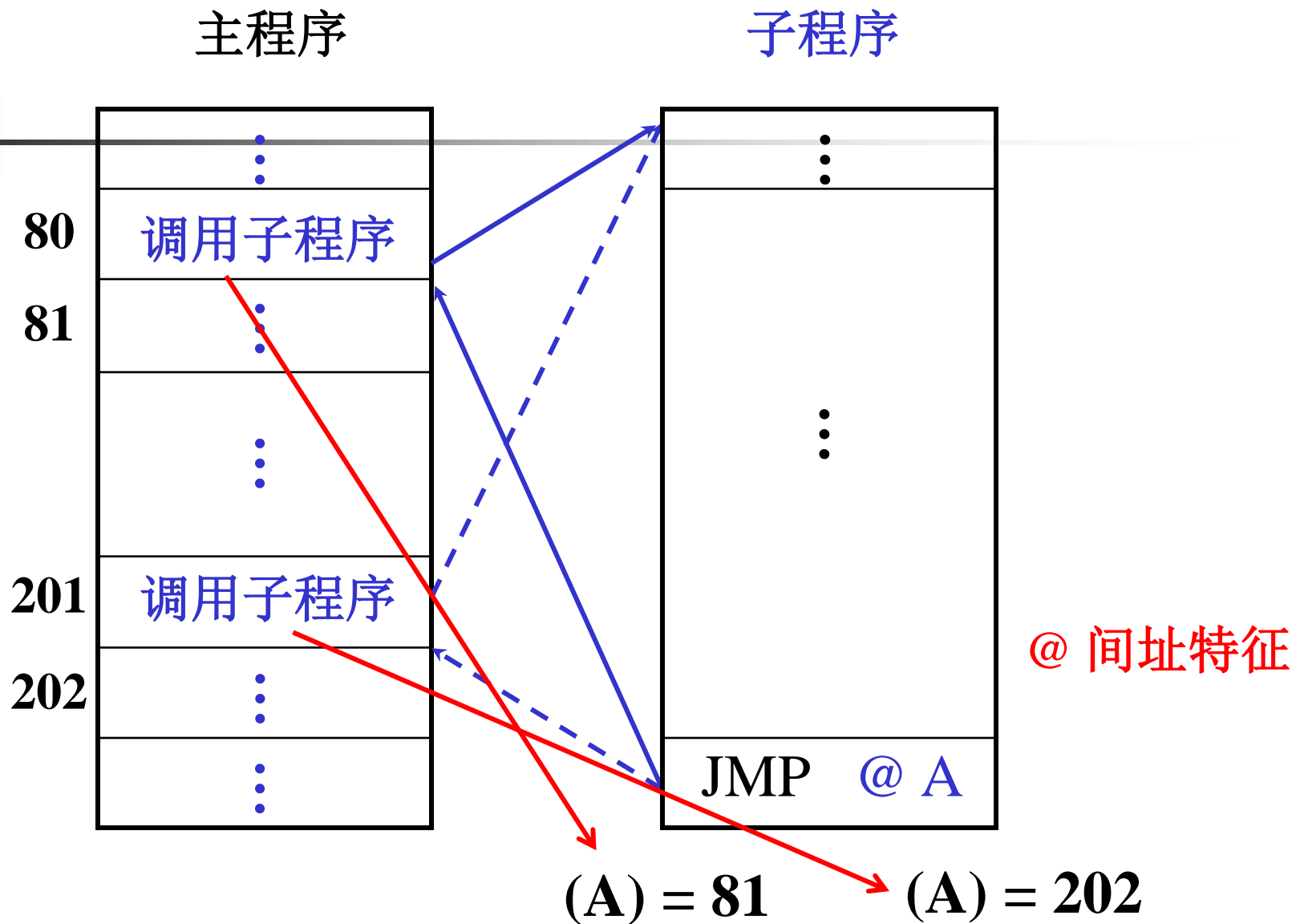
4. 间接寻址

MOV AX, [[2300]]

EA = (A) 有效地址由形式地址间接提供

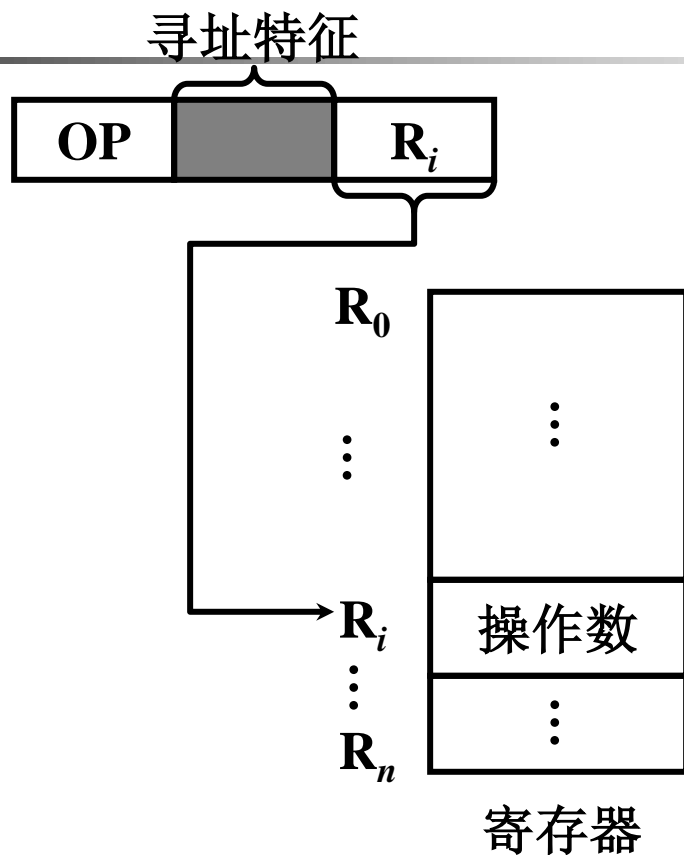


间接寻址编程举例



5. 寄存器寻址

$EA = R_i$ 有效地址即为寄存器编号



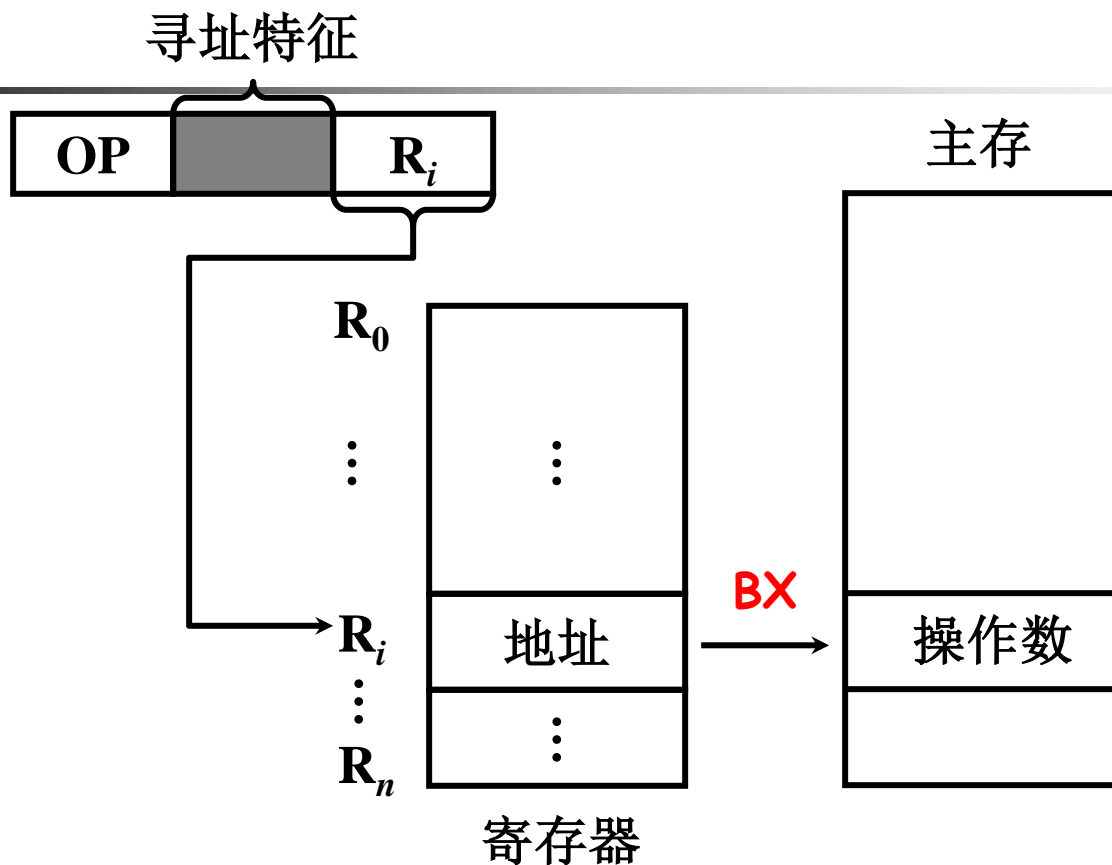
`MOV AX, BX`

- 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

$EA = (R_i)$

有效地址在寄存器中



`MOV AX, [BX]`

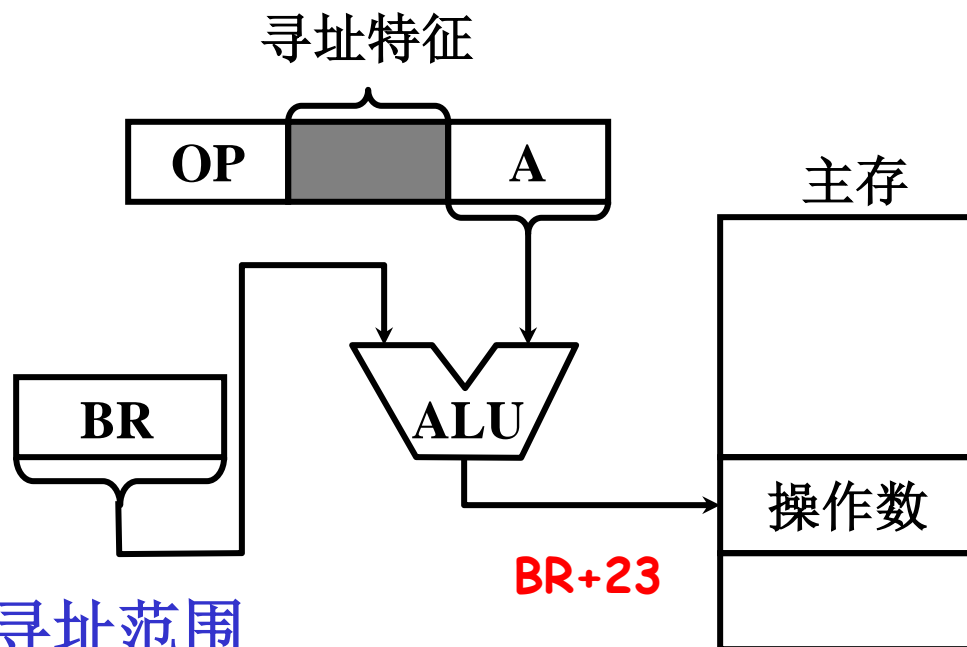
- 有效地址在寄存器中，操作数在存储器中，执行阶段访存
- 便于编制循环程序

7. 基址寻址

MOV AX, [BR+23]

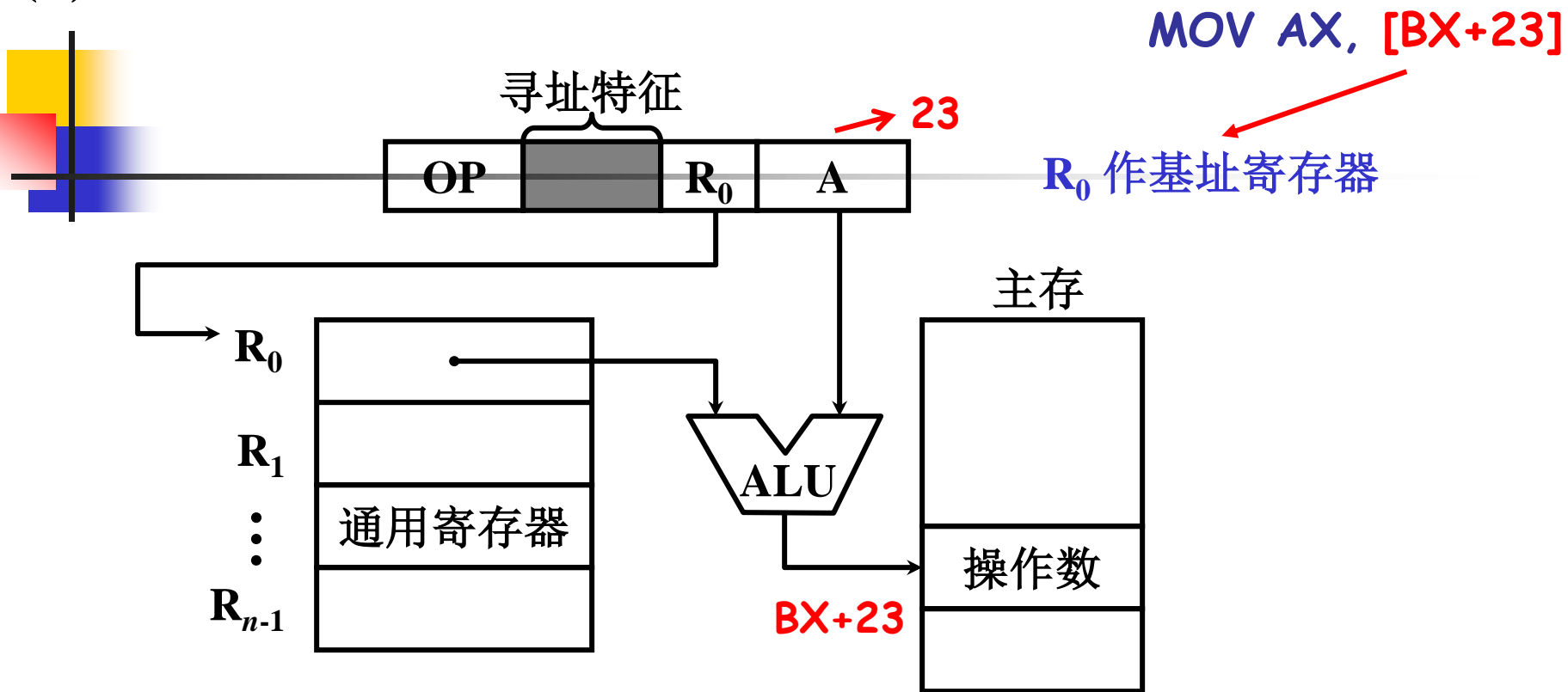
(1) 采用专用寄存器作基址寄存器

$EA = (BR) + A \rightarrow 23$ **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



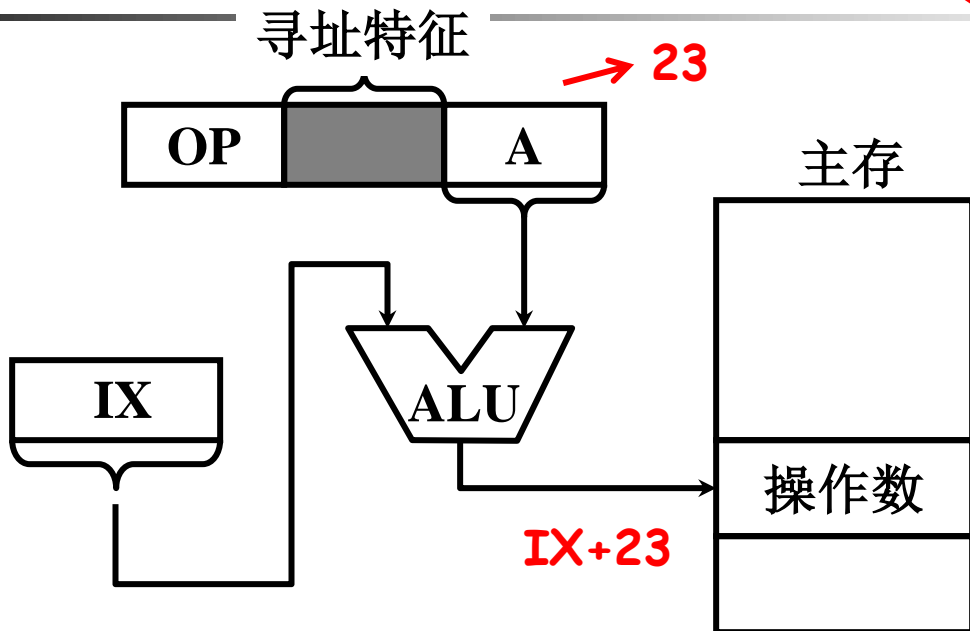
- 由用户指定哪个通用寄存器作为基址寄存器
- 基址寄存器的内容由操作系统确定
- 在程序的执行过程中 **R₀** 内容不变，形式地址 **A** 可变

8. 变址寻址

$$EA = (IX) + A$$

IX 为变址寄存器（专用）

通用寄存器也可以作为变址寄存器



`MOV AX, [IX+23]`

- 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

直接寻址

LDA D

ADD $D + 1$

ADD $D + 2$

⋮

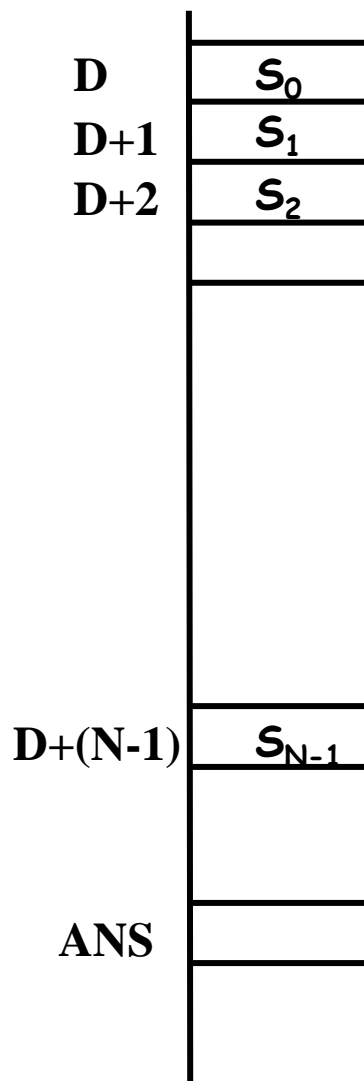
ADD $D + (N - 1)$

DIV $\# N$

STA ANS

共 $N + 2$ 条指令

$N=100$ 时，102条指令



例 设数据块首地址为 **D**，求 **N** 个数的平均值

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

$N=100$ 时，102 条指令

变址寻址

LDA # 0 **A=0**
LDX # 0 **X=0**
 X 为变址寄存器
ADD X, D **D 为形式地址**
INX **(X) + 1 → X**
CPX # N **(X) 和 #N 比较**
BNE M **结果不为零则转**
DIV # N
STA ANS

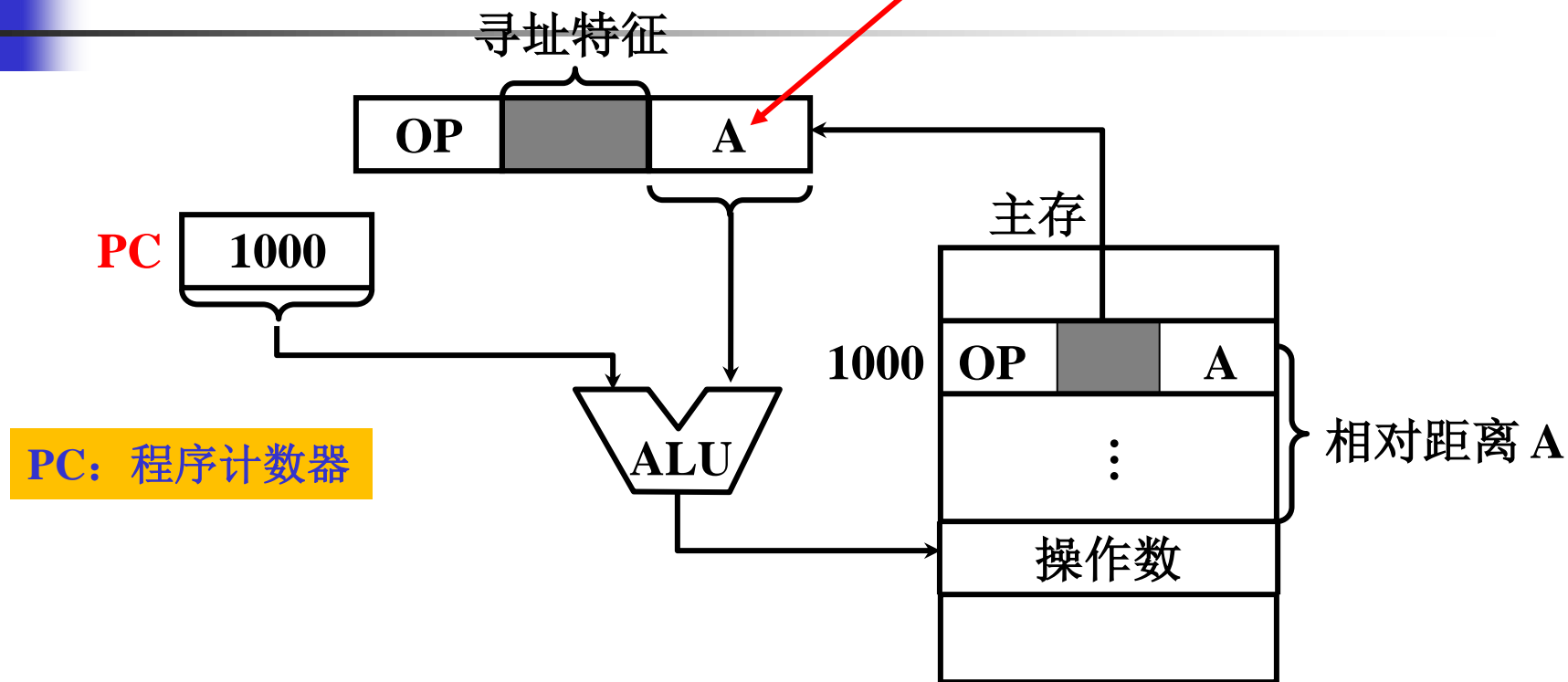
共 **8** 条指令

9. 相对寻址

$$EA = (PC) + A$$

A 是相对于当前指令的位移量（可正可负，补码）

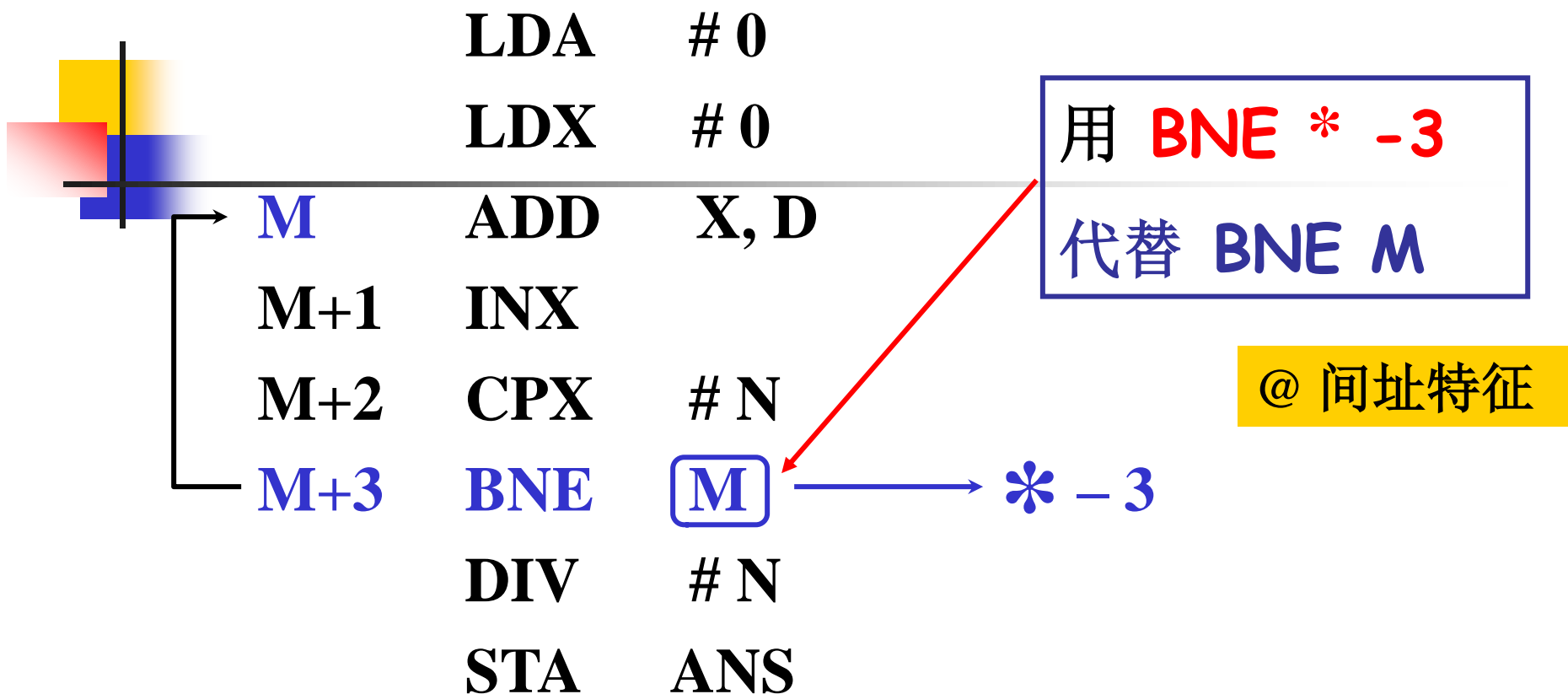
JMP A



- 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

✱ 相对寻址特征



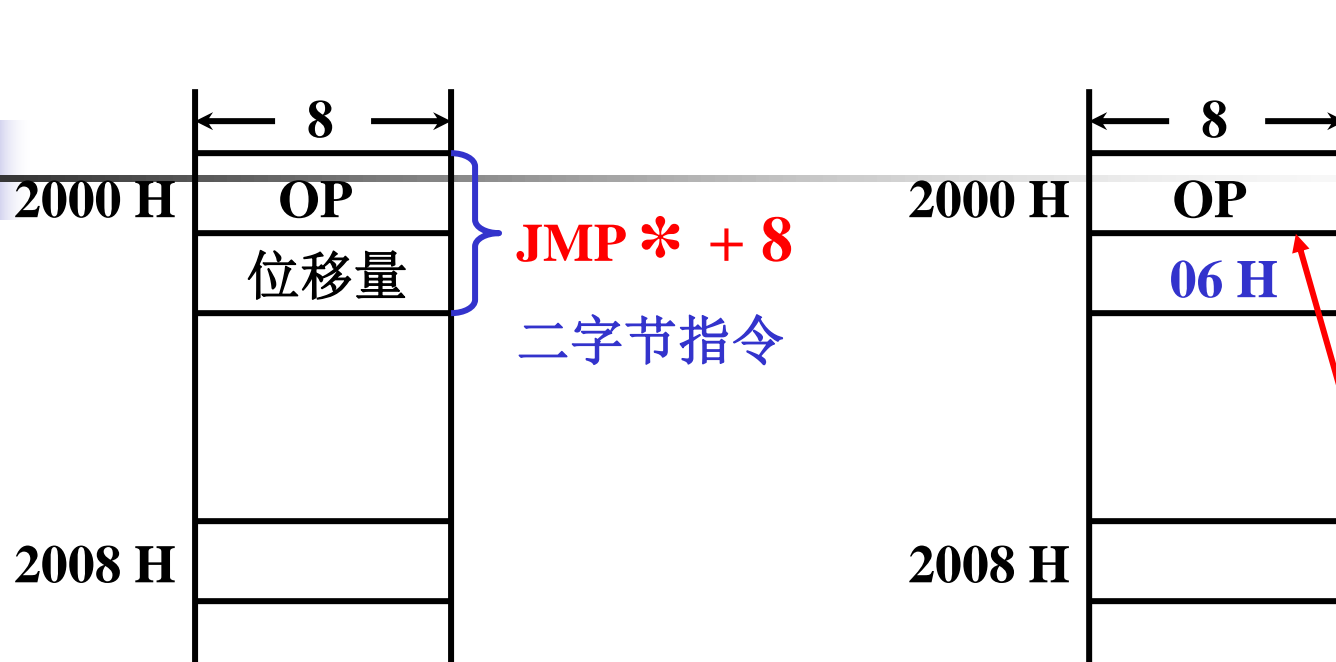
M 随程序所在存储空间的位置不同而不同

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



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**

- **例7.2:** 当前指令地址为**240**, 要求转移到**290**, 则转移指令的第二、三字节的机器码是什么? 当前指令地址为**240**, 要求转移到**200**, 则转移指令的第二、三字节的机器码是什么?

- **例7.2:** 当前指令地址为**240**，要求转移到**290**，则转移指令的第二、三字节的机器码是什么？当前指令地址为**240**，要求转移到**200**，则转移指令的第二、三字节的机器码是什么？

- **解:** (1) $290 - 243 = 47 = 2FH$

第二字节为**2FH**、第三字节为**00H**

(2) $200 - 243 = -43 = D5H$

第二字节为**D5H**、第三字节为**FFH**

240	OP	
241	2FH	D5H
242	00H	FFH
243		

$43 = 0010\ 1011$ (-43) 原码= $1010\ 1011$ (-43) 补码= $1101\ 0100 + 1 = 1101\ 0101 = D5$

取反加1

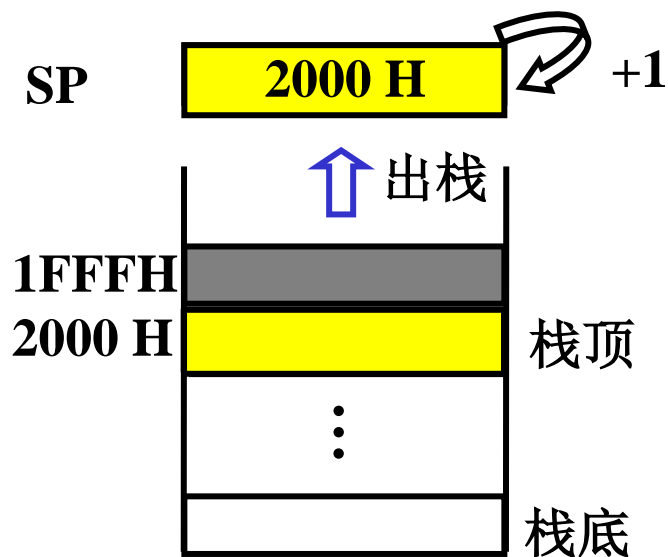
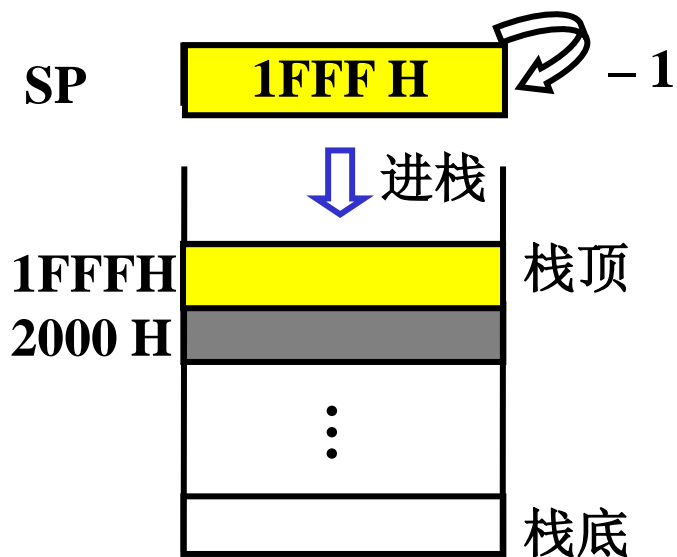
10. 堆栈寻址

(1) 堆栈的特点

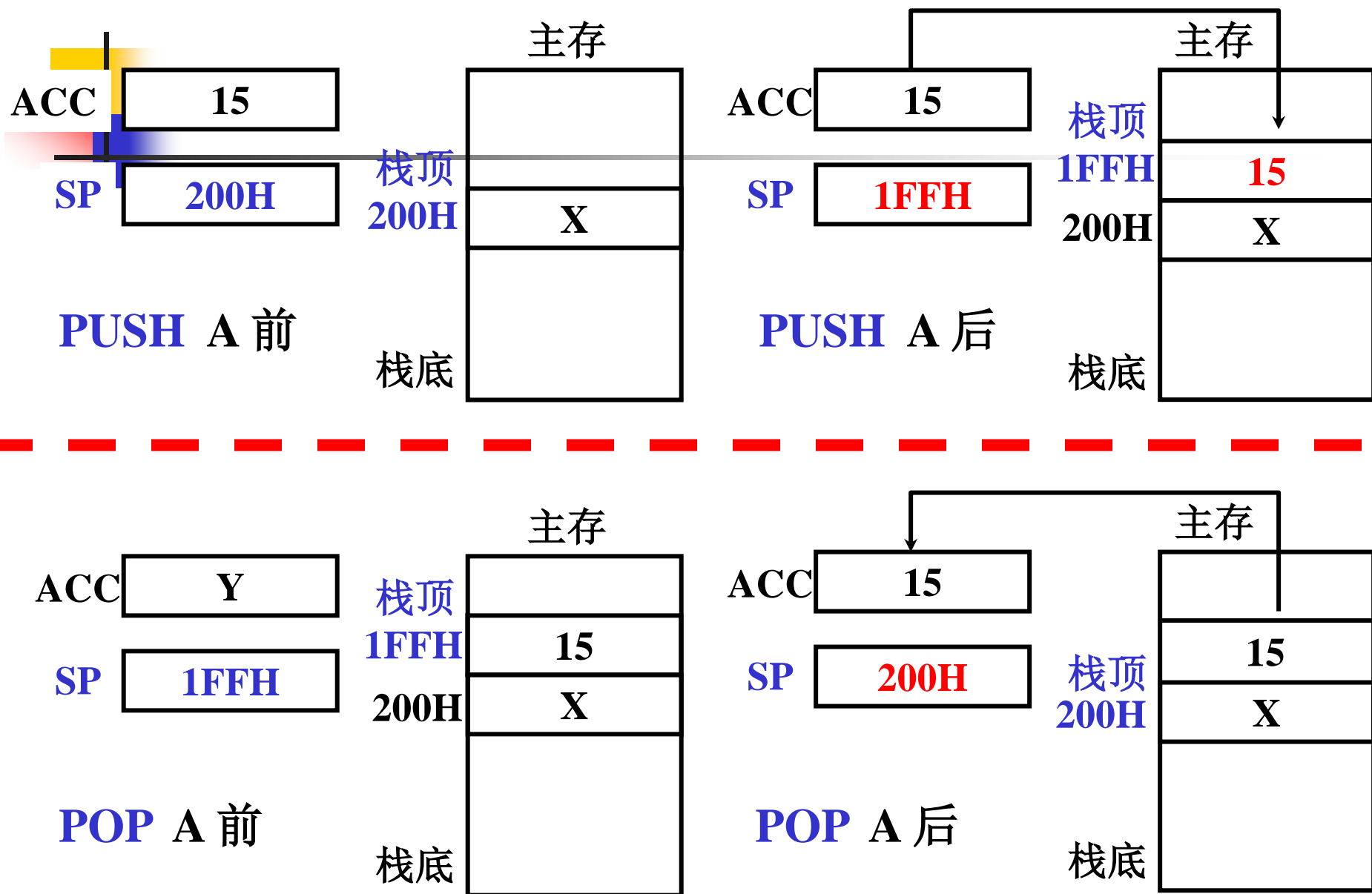
堆栈 { 硬堆栈 多个寄存器
 软堆栈 指定的存储空间

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



(3) SP 的修改与主存编址方法有关

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 $(SP) + 4 \longrightarrow SP$

- **例7.3**：子程序调用指令的**机器码(32位)=OP 5000H**；
PC=2000H；**SP=0100H**；**栈顶内容=2746H**。

CALL 指令

- (1) **CALL**指令被读取前，**PC**、**SP**及栈顶内容各为多少？
- (2) **CALL**指令被执行后，**PC**、**SP**及栈顶内容各为多少？
- (3) 子程序返回后，**PC**、**SP**及栈顶内容各为多少？

- **例7.3**：子程序调用指令的**机器码(32位)=OP 5000H**；
PC=2000H；**SP=0100H**；栈顶内容=**2746H**。

(1) **CALL**指令被读取前，**PC**、**SP**及栈顶内容各为多少？

(2) **CALL**指令被执行后，**PC**、**SP**及栈顶内容各为多少？

(3) 子程序返回后，**PC**、**SP**及栈顶内容各为多少？

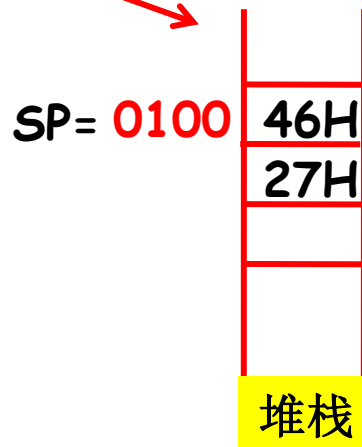
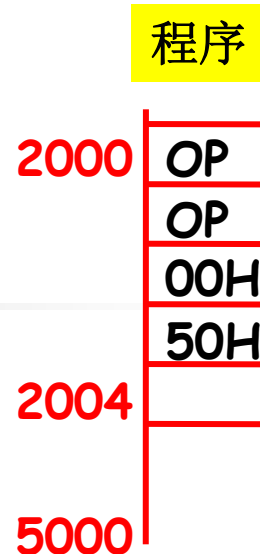


- **例7.3**: 子程序调用指令的**机器码(32位)=OP 5000H**;
PC=2000H; SP=0100H; 栈顶内容=2746H。

- (1) **CALL**指令被读取前, **PC**、**SP**及栈顶内容各为多少?
- (2) **CALL**指令被执行后, **PC**、**SP**及栈顶内容各为多少?
- (3) 子程序返回后, **PC**、**SP**及栈顶内容各为多少?

解:

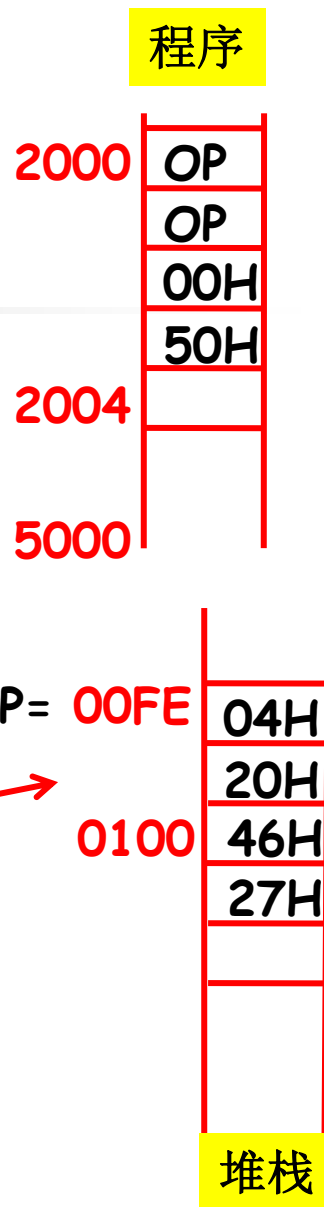
(1) **CALL** 指 令 被 读 取 前 , **PC=2000H** 、
SP=0100H、栈顶内容=**2746H**



- (1) **CALL**指令被读取前, **PC**、**SP**及栈顶内容各为多少?
- (2) **CALL**指令被执行后, **PC**、**SP**及栈顶内容各为多少?
- (3) 子程序返回后, **PC**、**SP**及栈顶内容各为多少?

(1) **CALL** 指令被读取前, **PC=2000H**、**SP=0100H**、**栈顶内容=2746H**

(2) **CALL** 指令被执行后, **PC=5000H**、**SP=0100H-02=00FEH**、栈顶内容=**2004H** (子程序返回的地址)



- **例7.3**: 子程序调用指令的**机器码(32位)=OP 5000H**;
PC=2000H; SP=0100H; 栈顶内容=2746H。

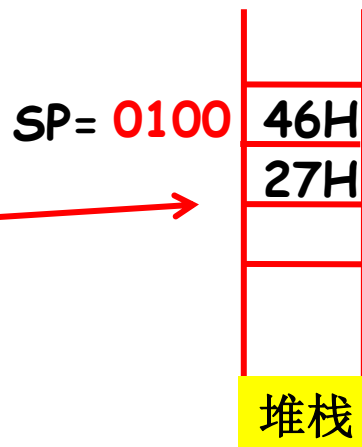
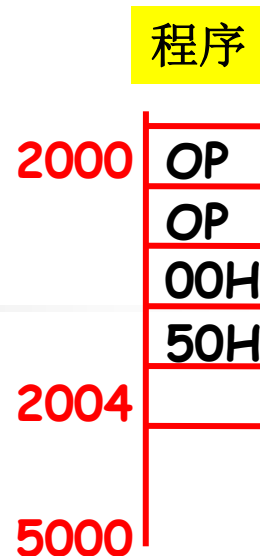
- (1) **CALL**指令被读取前, **PC**、**SP**及栈顶内容各为多少?
- (2) **CALL**指令被执行后, **PC**、**SP**及栈顶内容各为多少?
- (3) 子程序返回后, **PC**、**SP**及栈顶内容各为多少?

解:

(1) **CALL** 指 令 被 读 取 前 , **PC=2000H** 、
SP=0100H、栈顶内容=**2746H**

(2) **CALL** 指 令 被 执 行 后 , **PC=5000H** 、
SP=0100H-02=00FEH、栈顶内容=**2004H** (子
程序返回的地址)

(3) 子程序返回后, **PC=2004H**、**SP=0100H**、
栈顶内容=**2746H**





7.4 指令格式举例

- 一、设计指令格式时应考虑的各种因素
- 二、指令格式举例
- 三、指令格式设计举例

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）

80286的指令应该兼容8086的指令

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

二、指令格式举例

1. PDP-8 指令字长固定 12 位，共35条指令



采用扩展操作码技术

PDP-8 小型计算机



- **PDP**是“**Programmed Data Processor**”（程序数据处理机）的首字母缩写。**1960**年，**DEC**公司推出了首个以阴极管为监视器（**CRT**）的小型机产品**PDP-1**设备。**PDP-1**简化了大型机的功能，而且价格低廉。**1965**年推出了世界上第一台真正意义的小型计算机**PDP-8**。随后，**DEC**又推出了世界上首款采用晶体管的小型机**PDP-8/1**。

2. PDP – 11

指令字长有 16 位、32 位、48 位三种

OP-CODE

16

零地址 (16 位)

扩展操作码技术

OP-CODE	目的地址
---------	------

10

6

一地址 (16 位)

OP	源地址	目的地址
----	-----	------

4

6

6

二地址 R – R (16 位)

R: 寄存器
M: 存储器

OP	目的地址	存储器地址
----	------	-------

10

6

16

二地址 R – M (32 位)

OP	源地址	目的地址	存储器地址1	存储器地址2
----	-----	------	--------	--------

4

6

6

16

16

二地址 M – M (48 位)

2. PDP – 11

指令字长有 16 位、32 位、48 位三种



The diagram illustrates the instruction format of the PDP-11. It features a horizontal line with a vertical tick mark on the left. To the left of this tick mark are three overlapping squares: yellow, red, and blue. A rectangular box labeled 'OP-CODE' is positioned on the line, with the number '16' centered below it. To the right of the box, the text '零地址 (16 位)' is written in blue, with a horizontal line extending from the box to this text.

OP-CODE

16

零地址 (16 位)

例如: **NOP** 指令

2. PDP – 11

指令字长有 16 位、32 位、48 位三种



OP-CODE

16

零地址 (16 位)

OP-CODE

目的地址

10

6

一地址 (16 位)

例如: **INC AX** 指令

2. PDP – 11

指令字长有 16 位、32 位、48 位三种

OP-CODE

16

零地址 (16 位)

OP-CODE

目的地址

10

6

一地址 (16 位)

OP

源地址

目的地址

4

6

6

二地址 R – R (16 位)

例如: **MOV AX,BX** 指令

2. PDP – 11

指令字长有 16 位、32 位、48 位三种

OP-CODE

16

零地址 (16 位)

OP-CODE	目的地址
---------	------

10

6

一地址 (16 位)

OP	源地址	目的地址
----	-----	------

源地址

目的地址

4

6

6

二地址 R – R (16 位)

OP	目的地址	存储器地址
----	------	-------

目的地址

存储器地址

10

6

16

二地址 R – M (32 位)

例如: **MOV AX,[2300H]** 指令

2. PDP – 11

指令字长有 16 位、32 位、48 位三种

OP-CODE

16

零地址 (16 位)

OP-CODE	目的地址
---------	------

10

6

一地址 (16 位)

OP	源地址	目的地址
----	-----	------

源地址

目的地址

4

6

6

二地址 R – R (16 位)

OP	目的地址	存储器地址
----	------	-------

目的地址

存储器地址

10

6

16

二地址 R – M (32 位)

OP	源地址	目的地址	存储器地址1	存储器地址2
----	-----	------	--------	--------

源地址

目的地址

存储器地址1

存储器地址2

4

6

6

16

16

例如: **MOV [DI+2400H],[SI+2300H]** 指令

二地址 M – M (48 位)

PDP-11 小型计算机

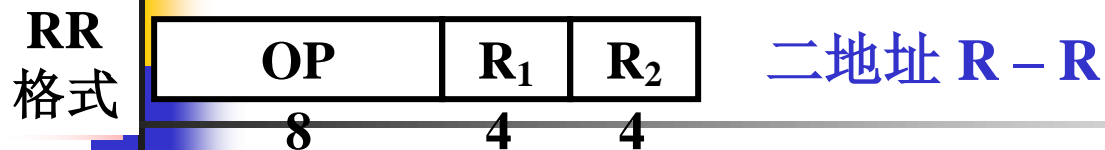
(PDP 11/23)



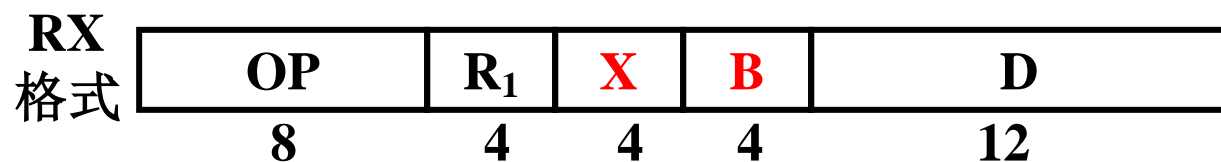
- **PDP-11**为美国**DEC**公司(**Digital Equipment Corp.**)于**1970**到**1980**年代, 所销售的一系列**16**位计算机。**PDP-11**是**DEC**公司的**PDP-8**系列的后续机种。**PDP-11**有着许多创新的特色, 而且比起其前代机种更容易编写程序。当**32**位的后续扩充机型**VAX-11**推出时, **PDP-11**已经广受程序员的喜爱。这两个机型后续的市场, 则多由**IBM PC**、苹果二号与升阳电脑的工作站电脑等个人电脑所取代。

3. IBM 360

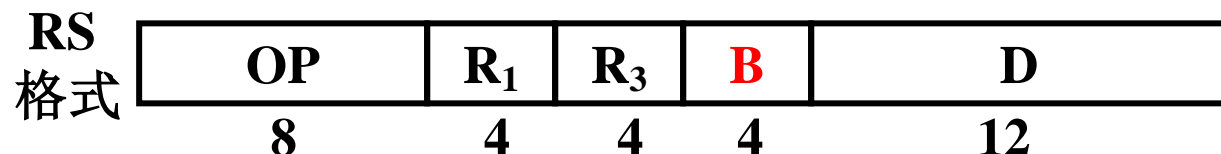
X: 变址
B: 基址
I: 立即数
R: 寄存器
D: 存储器地址



16位

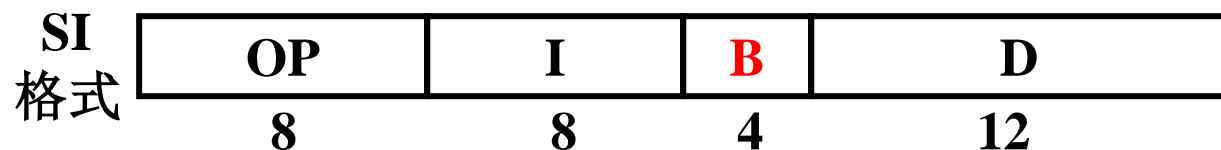


二地址 R - M
基址加变址寻址

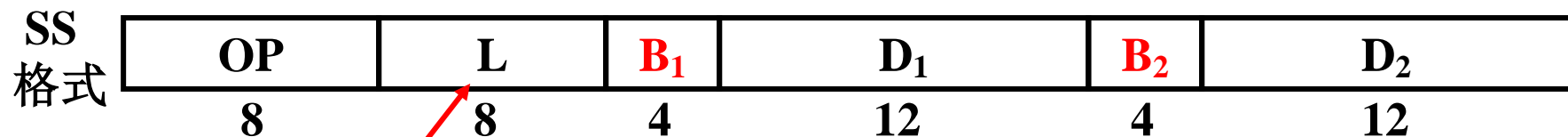


三地址 R - M
基址寻址

32位



立即数 - M
基址寻址

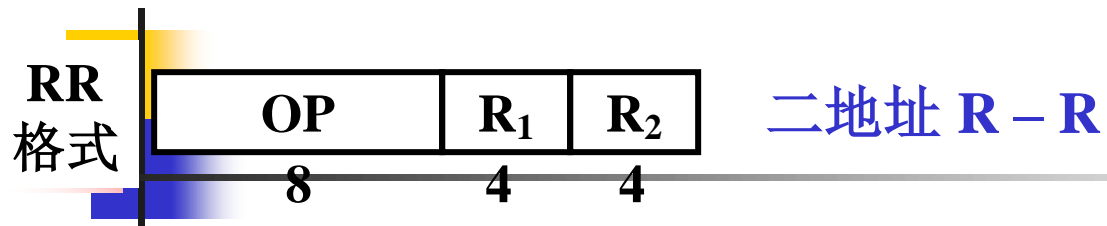


48位

L为数据长度字段

二地址 M - M
基址寻址

3. IBM 360



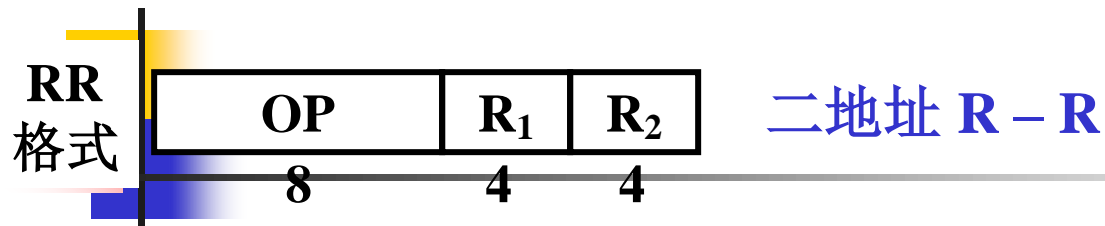
X: 变址
B: 基址
I: 立即数
R: 寄存器
D: 存储器地址

16位

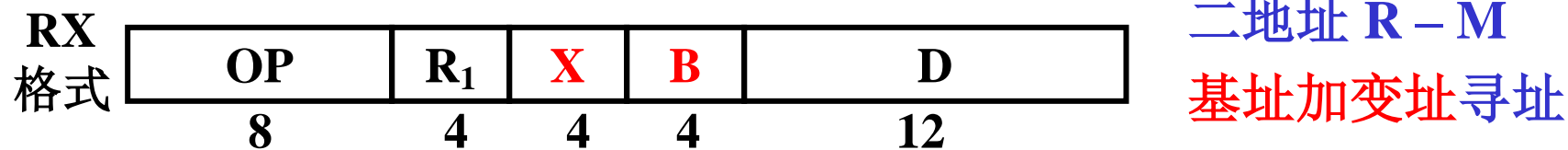
例如: **ADD AX, BX** 指令

3. IBM 360

X: 变址
B: 基址
I: 立即数
R: 寄存器
D: 存储器地址



16位

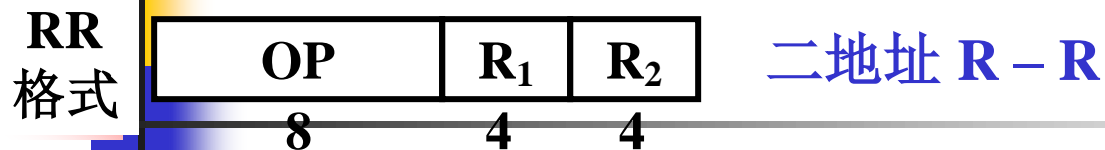


例如: **ADD AX, [X + B + 2300H]** 指令

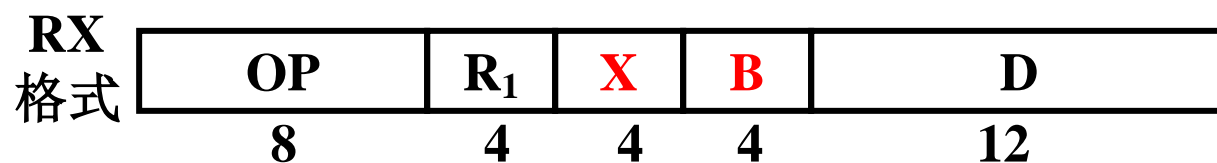
D

3. IBM 360

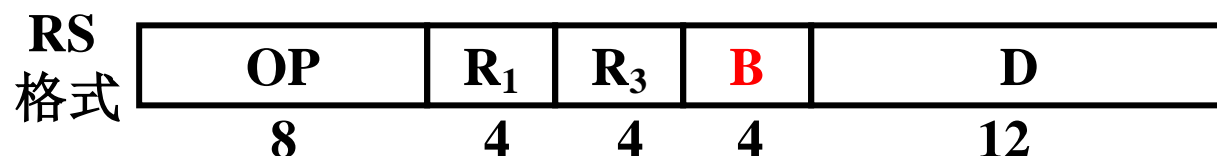
X: 变址
B: 基址
I: 立即数
R: 寄存器
D: 存储器地址



16位



二地址 R - M
基址加变址寻址



三地址 R - M
基址寻址

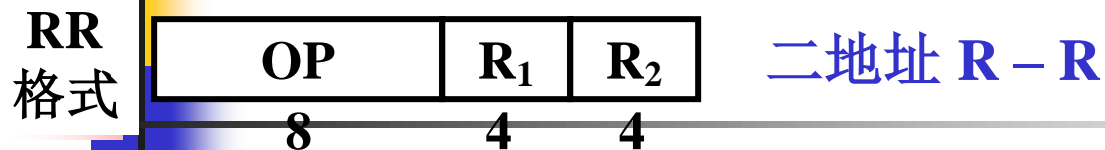
32位

例如: **ADD BX, AX, [B + 2300H]** 指令

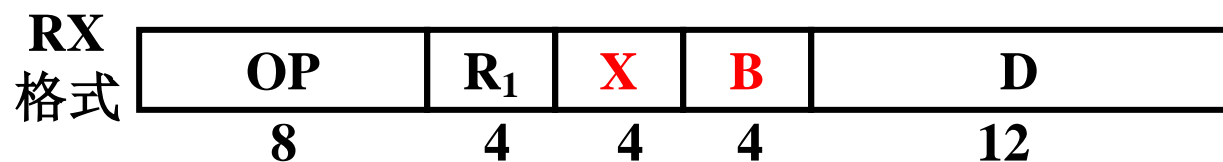
D

3. IBM 360

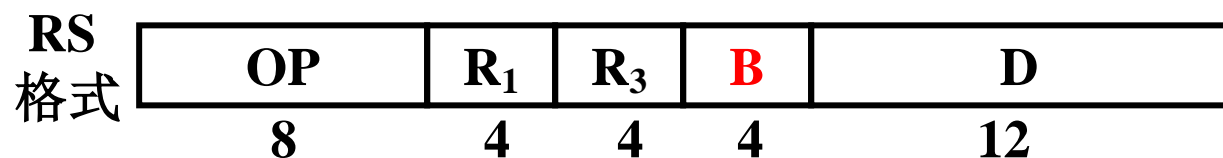
X: 变址
B: 基址
I: 立即数
R: 寄存器
D: 存储器地址



16位

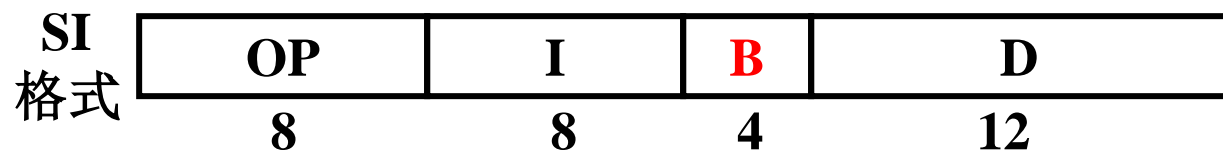


二地址 R - M
基址加变址寻址



三地址 R - M
基址寻址

32位



立即数 - M
基址寻址

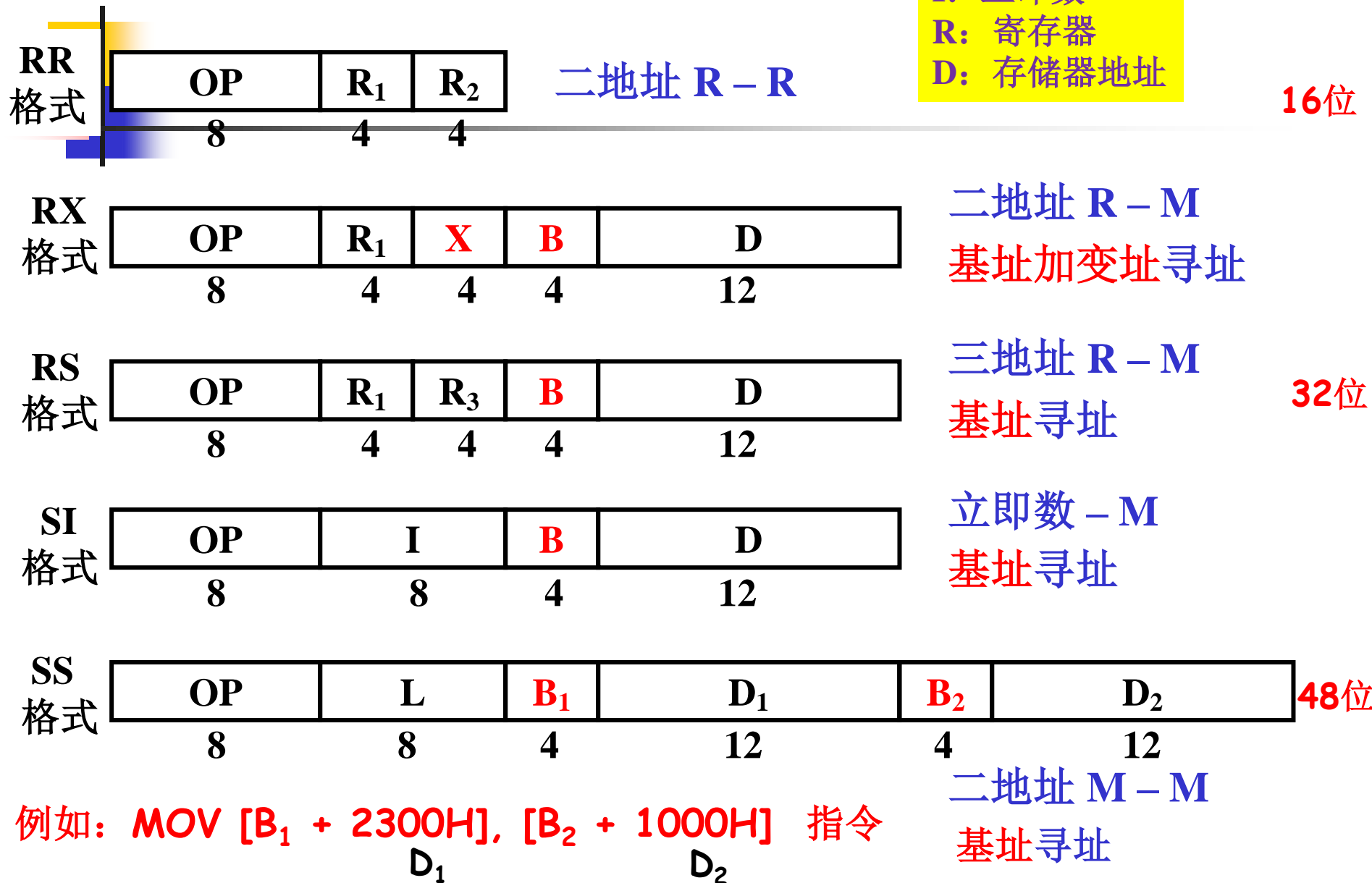
例如: **MOV [B + 2300H], 10H** 指令

D

I

3. IBM 360

X: 变址
B: 基址
I: 立即数
R: 寄存器
D: 存储器地址



IBM 360大型计算机



- **IBM System/360**是**IBM**在**1964**年**4**月**7**日，推出的划时代的大型电脑，这一系列是世界上首个指令集可兼容计算机。**1964**年以前，计算机厂商要针对每种主机量身定做操作系统，而**System/360**的问世则让单一操作系统适用于整系列的计算机。
- **IBM System/360**是**32位**的系列计算机。

4. Intel 8086

(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 = 1 + 4

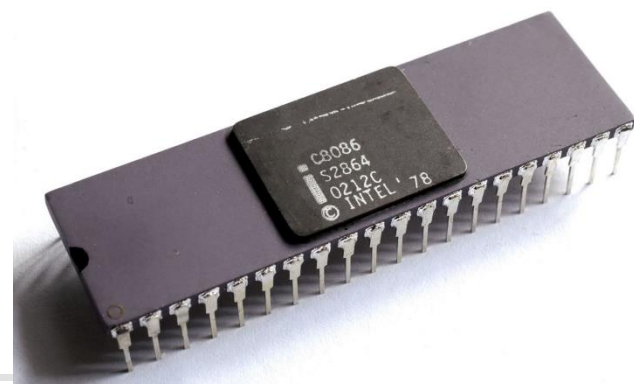
CALL 段内调用 3 字节 = 1 + 2

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

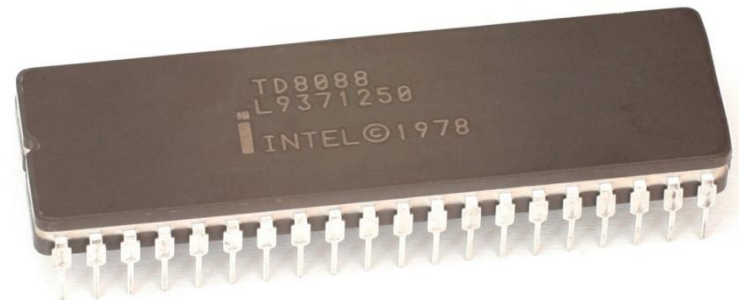
ADD AX, [3048H] 4 字节 寄存器 - 存储器

Intel 8086/8088 CPU



1978年6月INTEL推出了**8086**微处理器，主频**4.77MHz**，采用**16**位寄存器、**16**位数据总线和**29000**个**3**微米技术的晶体管，标志着第三代微处理器问世。售价**360**美元。不过当时由于**360**美元过于昂贵，大部分人都没有足够的钱购买使用此芯片的电脑，于是 Intel 在1年之后，推出了**8086**的简化版**8088**，是一款**4.77MHz**准**16**位微处理器。它在内部以**16**位运行，但支持**8**位数据总线，采用现有的**8**位设备控制芯片，包含**29000**个**3**微米技术的晶体管，可访问**1MB**内存地址，速度为**0.33MIPS**。**IBM**公司**1981**年生产的第一台电脑就是使用的这种芯片。这也标志着**x86**架构和**IBM PC** 兼容电脑的产生。

发布的时候，**8086**的时钟频率有**4.77**，**8**和**10MHz** 三个版本，包括了具有**300**个操作的指令集。其中**8MHz** 版本包含了大约**28,000**个 晶体管，具备**0.8 MIPS** 的能力。





三、指令格式设计举例

- **例7.4:** 字长**16**位，存储器直接寻址空间为**128**字，变址时的位移量为**-64 ~ +63**，**16**个通用寄存器均可作为变址寄存器。设计一套指令系统格式，满足下列寻址类型的要求：
 - (1) 直接寻址的二地址指令**3**条。
 - (2) 变址寻址的一地址指令**6**条。
 - (3) 寄存器寻址的二地址指令**8**条。
 - (4) 直接寻址的一地址指令**12**条。
 - (5) 零地址指令**32**条。

试问还有多少种代码未用？若安排寄存器寻址的一地址指令，还能容纳多少条？



三、指令格式设计举例

- **例7.4:** 字长**16**位，存储器直接寻址空间为**128**字，变址时的位移量为**-64 ~ +63**，**16**个通用寄存器均可作为变址寄存器。设计一套指令系统格式，满足下列寻址类型的要求：
 - (1) 直接寻址的二地址指令**3**条。
 - (2) 变址寻址的一地址指令**6**条。
 - (3) 寄存器寻址的二地址指令**8**条。
 - (4) 直接寻址的一地址指令**12**条。
 - (5) 零地址指令**32**条。

试问还有多少种代码未用？若安排寄存器寻址的一地址指令，还能容纳多少条？

- **解:** 直接寻址: **128**字 -> **7**位
变址寻址: **-64 ~ +63** -> **7**位
寄存器寻址: **16**个寄存器 -> **4**位

(1) 直接寻址的二地址指令**3**条

2

7

7

OP

A1

A2

OP

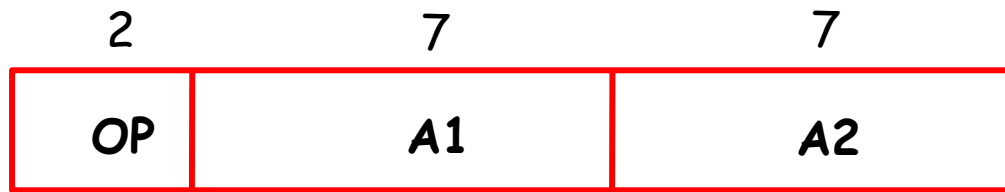
00

01

10

3条

(1) 直接寻址的二地址指令3条

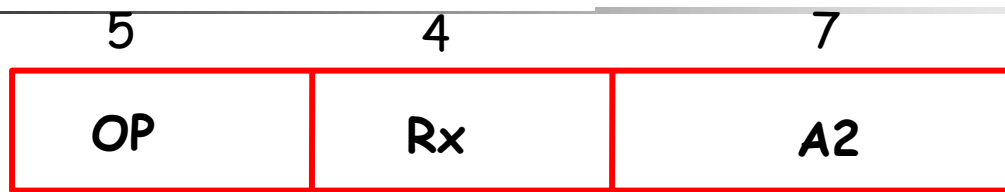


OP

00
01
10

3条

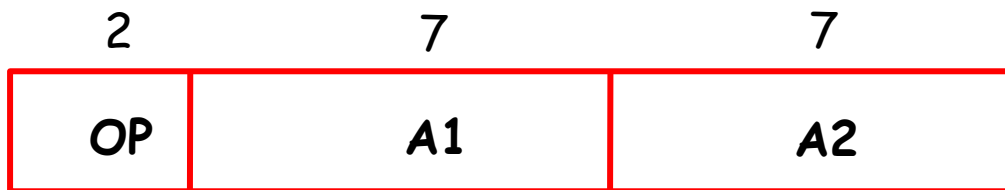
(2) 变址寻址的一地址指令6条



11000
.....
11101

6条

(1) 直接寻址的二地址指令3条

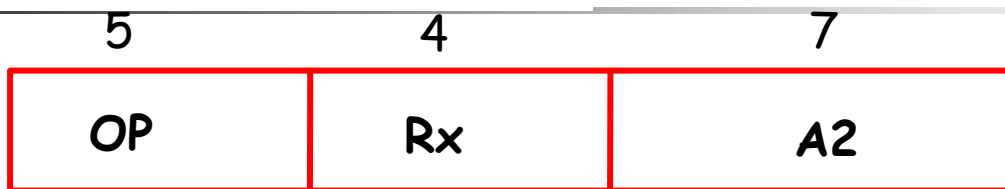


OP

00
01
10

3条

(2) 变址寻址的一地址指令6条

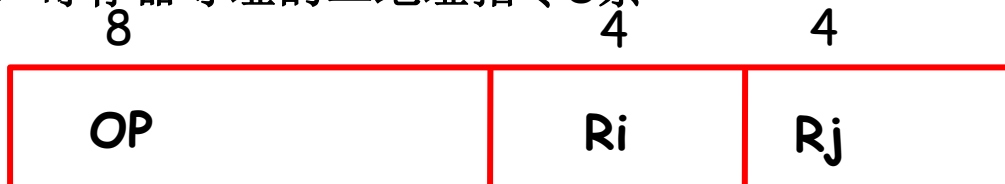


11000

.....
11101

6条

(3) 寄存器寻址的二地址指令8条

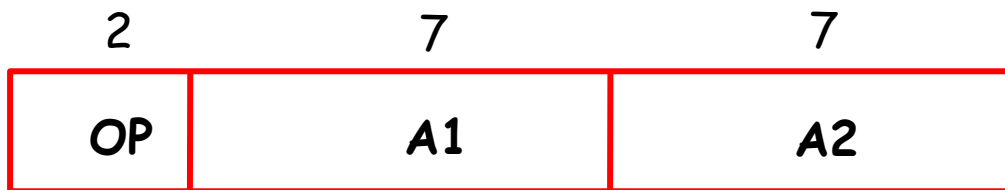


11110000

.....
11110111

8条

(1) 直接寻址的二地址指令3条

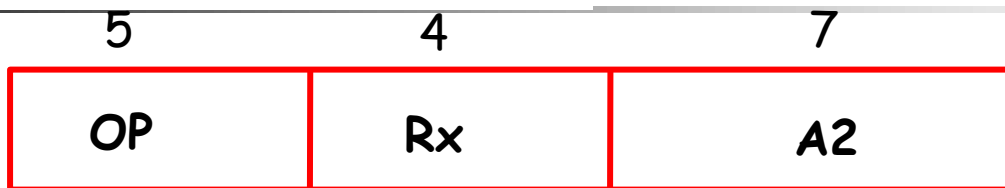


OP

00
01
10

3条

(2) 变址寻址的一地址指令6条



11000

.....
11101

6条

(3) 寄存器寻址的二地址指令8条

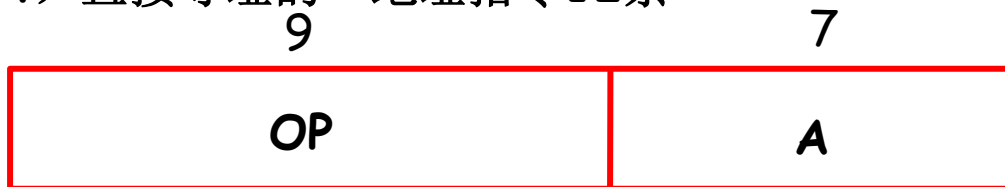


11110000

.....
11110111

8条

(4) 直接寻址的一地址指令12条

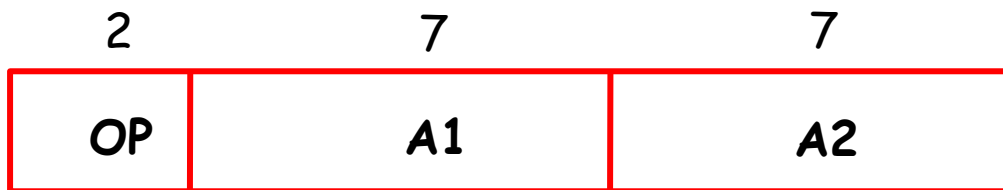


111110000

.....
111110111

12条

(1) 直接寻址的二地址指令**3**条

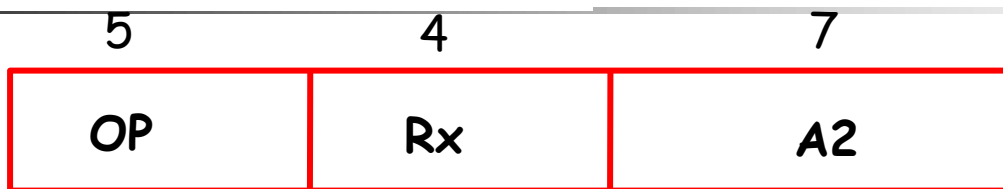


OP

00
01
10

3条

(2) 变址寻址的一地址指令**6**条

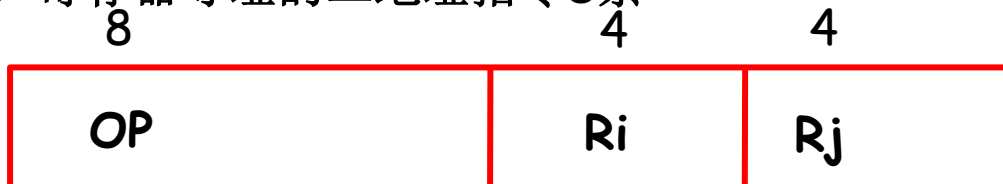


11000

.....
11101

6条

(3) 寄存器寻址的二地址指令**8**条

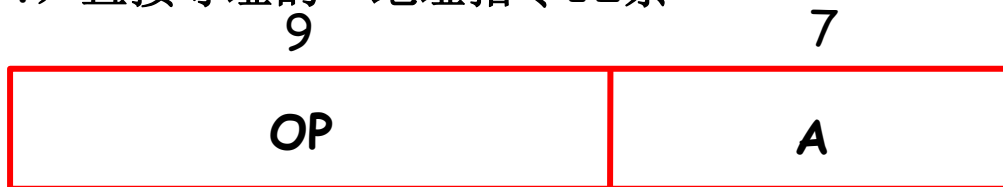


11110000

.....
11110111

8条

(4) 直接寻址的一地址指令**12**条

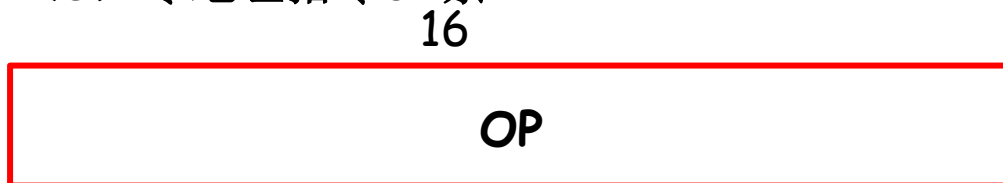


111110000

.....
111110111

12条

(5) 零地址指令**32**条



1111111000000000

.....
1111111000011111

32条

00 XXXXXXXXXXXX
01 XXXXXXXXXXXX
10 XXXXXXXXXXXX

3

(1) 直接寻址的二地址指令3条

11 000XXXXXXXXX
11 001XXXXXXXXX
11 010XXXXXXXXX
11 011XXXXXXXXX
11 100XXXXXXXXX
11 101XXXXXXXXX

6

(2) 变址寻址的一地址指令6条

11 110 000XXXXXXXX
11 110 001XXXXXXXX
11 110 010XXXXXXXX
11 110 011XXXXXXXX
11 110 100XXXXXXXX
11 110 101XXXXXXXX
11 110 110XXXXXXXX
11 110 111XXXXXXXX

8

(3) 寄存器寻址的二地址指令8条

11 111 0000XXXXXXXX
11 111 0001XXXXXXXX
11 111 0010XXXXXXXX
11 111 0011XXXXXXXX
11 111 0100XXXXXXXX
11 111 0101XXXXXXXX
11 111 0110XXXXXXXX
11 111 0111XXXXXXXX
11 111 1000XXXXXXXX
11 111 1001XXXXXXXX
11 111 1010XXXXXXXX
11 111 1011XXXXXXXX

12

(4) 直接寻址的一地址指令12条

11 111 1100XXXXXXXX

128

11 111 1101XXXXXXXX

128

11 111 1110XXXXXXXX

128

11 111 1111XXXXXXXX

11 111 1100 00 00000

32

(5) 零地址指令32条

11 111 1100 00 11111

11 111 1100 01 00000

32

11 111 1100 01 11111

11 111 1100 10 00000

32

11 111 1100 10 11111

11 111 1100 11 00000

32

11 111 1100 11 11111

试问还有多少种代码未用？

- 还有 $(128-32) + 128 + 128 + 128 = 480$ 种代码未用



- 11111 1100 0100000

-
■ 11111 1100 1111111

$128 - 32 = 32 + 32 + 32$

- 11111 1101 0000000

-
■ 11111 1101 1111111

128

- 11111 1110 0000000

-
■ 11111 1110 1111111

128

- 11111 1111 0000000

-
■ 11111 1111 1111111

128

```
00 XXXXXXXXXXXXXXXX
01 XXXXXXXXXXXXXXXX
10 XXXXXXXXXXXXXXXX
```

3

(1) 直接寻址的二地址指令3条

```
11 000XXXXXXXXXXXXX
11 001XXXXXXXXXXXXX
11 010XXXXXXXXXXXXX
11 011XXXXXXXXXXXXX
11 100XXXXXXXXXXXXX
11 101XXXXXXXXXXXXX
```

6

(2) 变址寻址的一地址指令6条

```
11 110 000XXXXXXXXX
11 110 001XXXXXXXXX
11 110 010XXXXXXXXX
11 110 011XXXXXXXXX
11 110 100XXXXXXXXX
11 110 101XXXXXXXXX
11 110 110XXXXXXXXX
11 110 111XXXXXXXXX
```

8

(3) 寄存器寻址的二地址指令8条

```
11 111 0000XXXXXXXX
11 111 0001XXXXXXXX
11 111 0010XXXXXXXX
11 111 0011XXXXXXXX
11 111 0100XXXXXXXX
11 111 0101XXXXXXXX
11 111 0110XXXXXXXX
11 111 0111XXXXXXXX
11 111 1000XXXXXXXX
11 111 1001XXXXXXXX
11 111 1010XXXXXXXX
11 111 1011XXXXXXXX
```

12

(4) 直接寻址的一地址指令12条

```
11 111 1100XXXXXXXX
```

```
11 111 1101XXXXXXX
```

128 8

```
11 111 1110XXXXXXX
```

128 8

```
11 111 1111XXXXXXX
```

128 8

```
11 111 1100 00 00000
```

32

(5) 零地址指令32条

```
11 111 1100 00 11111
```

```
11 111 1100 01 00000
```

32

```
11 111 1100 01 11111
```

```
11 111 1100 10 00000
```

32

```
11 111 1100 10 11111
```

```
11 111 1100 11 00000
```

32

```
11 111 1100 11 11111
```

6

若安排寄存器寻址的一地址指令，还能容纳多少条？

- 若安排寄存器寻址的一地址指令，除去末**4**位为寄存器地址外，还可容纳**30**条这种指令

■ 11 111 1100 010 XXXX

■ 6

■ 11 111 1100 111 XXXX

■ 11 111 1101 000 XXXX

■ 8

■ 11 111 1101 111 XXXX

■ 11 111 1110 000 XXXX

■ 8

■ 11 111 1110 111 XXXX

■ 11 111 1111 000 XXXX

■ 8

■ 11 111 1111 111 XXXX

■ **例7.5:** 某机器配有基址寄存器和变址寄存器，采用一地址格式的指令系统，允许直接和间接寻址，且指令字长、机器字长和存储字长均为**16**位：

（**1**）若采用单字长指令，共能完成**105**种操作，则指令可直接寻址的范围是多少？一次间接寻址的寻址范围是多少？画出其指令格式并说明各字段的含义。

（**2**）若存储字长不变，可采用什么方式直接访问容量为**16MB**的主存？

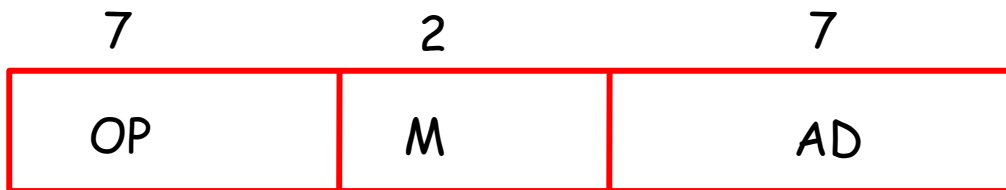
- **例7.5:** 某机器配有基址寄存器和变址寄存器，采用一地址格式的指令系统，允许直接和间接寻址，且指令字长、机器字长和存储字长均为**16**位：

(1) 若采用单字长指令，共能完成**105**种操作，则指令可直接寻址的范围是多少？一次间接寻址的寻址范围是多少？画出其指令格式并说明各字段的含义。

(2) 若存储字长不变，可采用什么方式直接访问容量为**16MB**的主存？

解: (1) **105**种操作 \rightarrow 操作码=**7**位 ($2^6=64$ $2^7=128$)

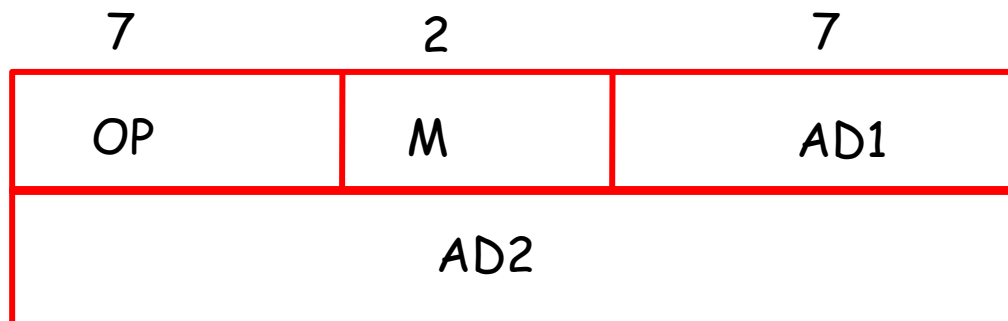
允许直接和间接寻址，有基址寄存器和变址寄存器 \rightarrow **2**位寻址特征位 (**4种组合**)



可直接寻址 $2^7=128$ ；一次间接寻址的范围是 $2^{16}=65536$



(2) 采用双字长指令



形式地址为**AD1//AD2**，共**7+16=23**位， **$2^{23}=8\text{M}$** ，存储字长为**16**位，故可以访问**16MB**的主存（ **$8\text{M} \times 16$ 位**）

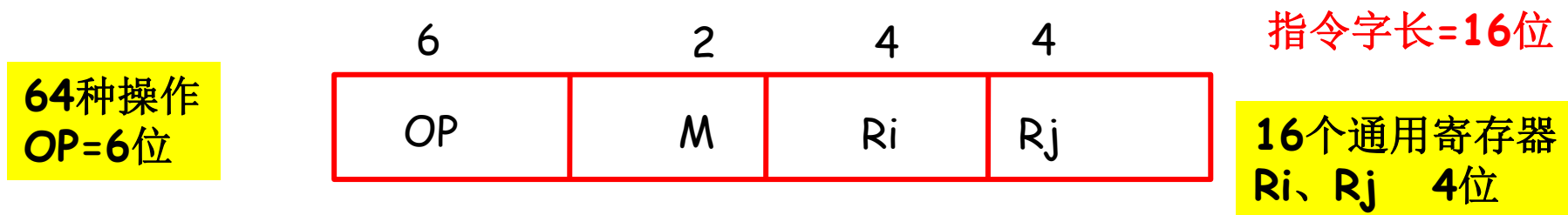
采用双字长指令直接访问容量为**16MB**的主存

- **例7.6:** 某模型机共有**64**种操作，操作码位数固定，且有以下特点：
 - (1) 采用一地址或二地址格式。
 - (2) 有寄存器寻址、直接寻址和相对寻址（**-128 ~ +127**）三种寻址方式。
 - (3) 有**16**个通用寄存器，算术运算和逻辑运算的操作数均在寄存器中，结果也在寄存器中。
 - (4) 取数/存数指令在通用寄存器和存储器之间传送数据。
 - (5) 存储器容量为**1MB**，按字节编址。

要求设计算术逻辑指令、取数/存数指令和相对转移指令的格式，并简述理由。

- **例7.6:** 某模型机共有**64**种操作，操作码位数固定，且有以下特点：
 - (1) 采用一地址或二地址格式。
 - (2) 有寄存器寻址、直接寻址和相对寻址（**-128 ~ +127**）三种寻址方式。
 - (3) 有**16**个通用寄存器，算术运算和逻辑运算的操作数均在寄存器中，结果也在寄存器中。
 - (4) 取数/存数指令在通用寄存器和存储器之间传送数据。
 - (5) 存储器容量为**1MB**，按字节编址。
 要求设计算术逻辑指令、取数/存数指令和相对转移指令的格式，并简述理由。

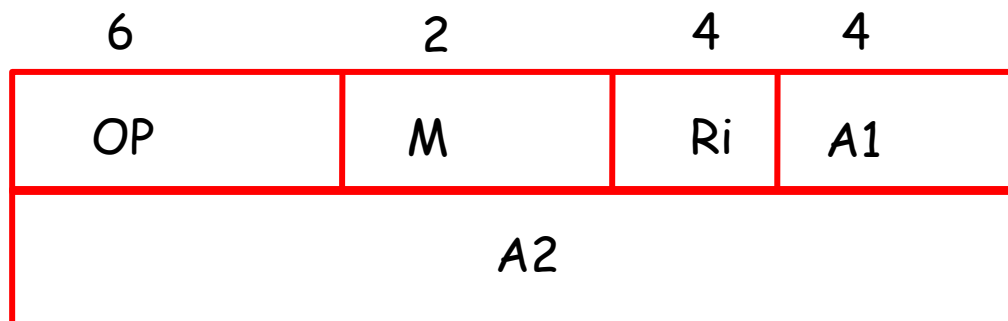
解：（1）算术逻辑指令格式为寄存器-寄存器型



M为寻址模式：可反映寄存器寻址、直接寻址、相对寻址三种寻址方式。



(2) 取数/存数寻址为寄存器-存储器型

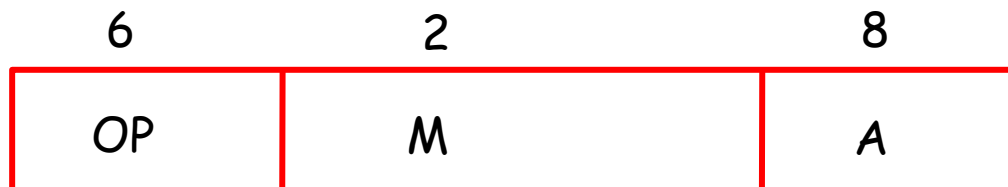


存储器地址=**A1**//**A2**=**20**位，可访问**1MB**容量的存储器

按字节编址



(3) 相对转移指令为一地址格式



A为位移量：相对寻址（ **-128 ~ +127** ）

■ **例7.7:** 某机共能完成**110**种操作，**CPU**有**8**个通用寄存器（**16**位），主存容量为**4M**字，采用寄存器-存储器型指令：

（**1**）欲使指令可直接访问主存的任一地址，指令字长应取多少位？画出指令格式。

（**2**）若在上述设计的指令字中设置一寻址特征位**X**，且**X=1**表示某个寄存器作基址寄存器，画出指令格式。试问基址寻址可否访问主存的任一单元？为什么？如果不能，提出一种方案，使其可访问主存的任一位置。

（**3**）若主存容量扩大到**4G**字，且存储字长等于指令字长，则在不改变上述硬件结构的前提下，可采用什么方法使指令可访问存储器的任一位置？

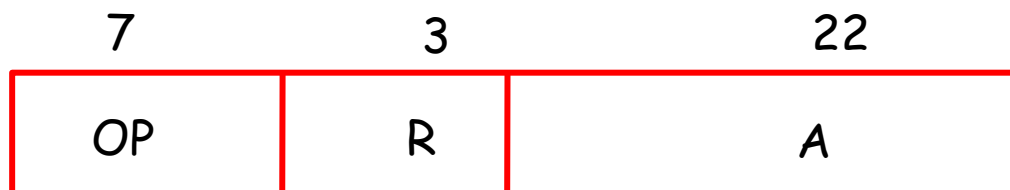
- **例7.7:** 某机共能完成**110**种操作，**CPU**有**8**个通用寄存器（**16**位），主存容量为**4M**字，采用寄存器-存储器型指令：

（1）欲使指令可直接访问主存的任一地址，指令字长应取多少位？画出指令格式。

（2）若在上述设计的指令字中设置一寻址特征位**X**，且**X=1**表示某个寄存器作基址寄存器，画出指令格式。试问基址寻址可否访问主存的任一单元？为什么？如果不能，提出一种方案，使其可访问主存的任一位置。

（3）若主存容量扩大到**4G**字，且存储字长等于指令字长，则在不改变上述硬件结构的前提下，可采用什么方法使指令可访问存储器的任一位置？

解: （1）欲使指令可直接访问主存的任一地址，采用寄存器-存储器型指令：



指令字长
32位

OP=7位: 110种操作

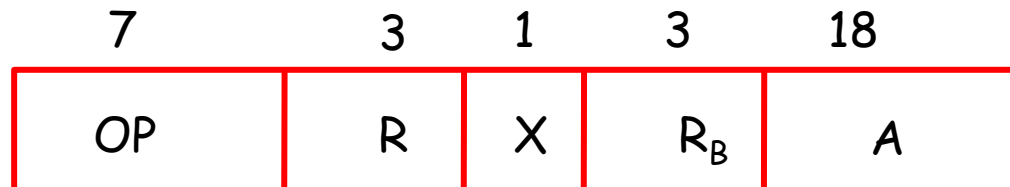
R=3位: 8个通用寄存器

A=22位: $2^{22}=4M$

(2) 在上述设计的指令字中设置一**寻址特征位X**，且**X=1**表示某个寄存器作基址寄存器**R_B**（因为要能将任一个寄存器都可以作为基址寄存器，故**R_B**的特征位要**3**位）

基址寻址方式不能访问主存的任一单元（**R_B=16**位、**A=18**位，得不到**4M**字的地址----需要**22**位）

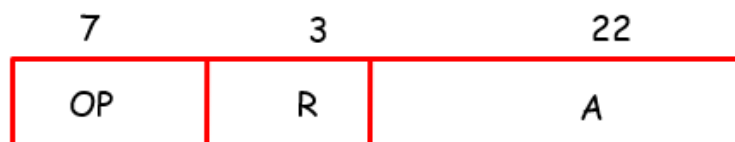
可将**R_B**寄存器内容左移**6**位，低位补**0**，形成**22**位基地址，然后与形式地址（**A**，**18**位）相加，得到的有效地址即可访问**4M**字存储器的任一单元



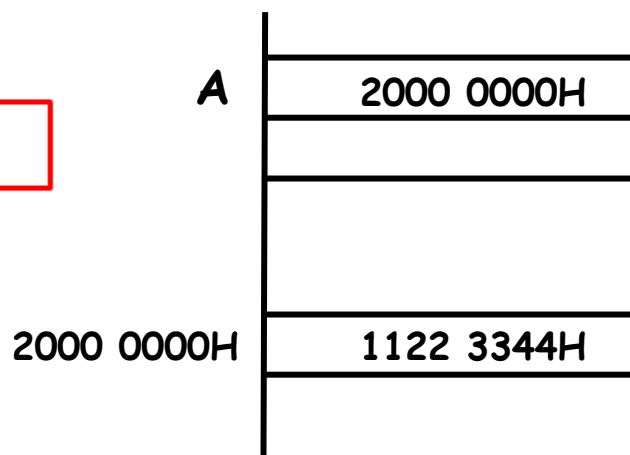
$$R_B \times 2^6 + A$$

主存容量=4G字，存储字长=指令字长=32位

(3) 若主存容量扩大到4G字，可采用一次间接寻址即可访问存储器的任一单元，因为间接寻址后得到的有效地址为32位(16位+16位)， $2^{32}=4G$



MOV AX, [[A]]





7.5 RISC 技术

- 一、RISC 的产生和发展
- 二、RISC 的主要特征
- 三、CISC 的主要特征
- 四、RISC和CISC 的比较

一、RISC 的产生和发展



精简指令系统计算机：RISC, Reduced Instruction Set Computer

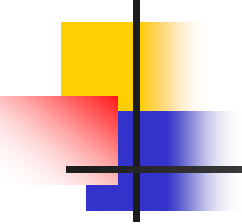
- 系列计算机的发展（指令系统向上兼容，如**80X86**系列计算机），使得计算机的指令系统越来越复杂

复杂指令系统计算机：CISC, Complex Instruction Set Computer

- 另一方面，计算机中的“**80 — 20 规律**”，即典型程序中 80% 的语句仅仅使用处理机中 20% 的指令（表7.3 IBM370机指令的使用频度）
- 导致，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？

表 7.3 IBM 370 机指令的使用频度(%)

指令类型	转移	逻辑	数据 存取	存 - 存 传送	整数 运算	浮点 运算	十进制 运算	其他
79.4% COBOL	24.6	14.6	40.2	12.4	6.4	0.0	1.6	0.6
74.8% FORTRAN	18.0	8.1	48.7	2.1	11.0	11.9	0.0	0.2
82.3% PASCAL	18.4	9.9	54.0	4.8	7.0	6.8	0.0	0.1

- 
- **1975年**，**IBM**公司的**John Cocke**提出了精简指令系统的设想
 - **1982年**，美国**UC Berkeley**的研究人员专门研究了如何有效利用**VLSI**（超大规模集成电路）的有效空间
 - **RISC I**（第一代**RISC**计算机）：字长**32**位，**128**个寄存器，**31**条指令，两种寻址方式，访存指令只有**2**条（**LOAD**和**STORE**），但是速度是同类**CISC**计算机（**303**条指令，**16**种寻址方式，**9**种数据格式）的**1**倍



John Cocke

- 约翰·科克是从机械到数学、又从数学转到计算机方向上来的学者。他生于**1925**年，**1946**年在杜克大学(**Duke University**)获得机械工程学士学位，干了几
年实际工作以后，又回到母校读研究生，于**1956**年取得数学博士学位。之后，他进入**IBM**，从此开始了他的计算机生涯并为**IBM**计算机市场的开拓和计算机科学技术的发展，尤其是**RISC架构**和**编译器优化**，做出了巨大的贡献。



UC Berkeley

- **加州大学伯克利分校 (University of California, Berkeley)**，简称伯克利，位于美国旧金山湾区伯克利市，是世界著名公立研究型大学、在学术界享有盛誉，位列**2018-19年USNews**世界大学排名世界第**4**、**ARWU**世界大学学术排名世界第**5**。
- 截止**2018年10月**，伯克利校友、教授及研究人员中共走出了**107**位诺贝尔奖得主（世界第三）、**14**位菲尔兹奖得主（世界第四）和**25**位图灵奖得主（世界第二）。
- 此外，伯克利为南湾的硅谷培养了大量人才，包括英特尔创始人戈登·摩尔、苹果公司创始人斯蒂夫·沃兹尼亚克、特斯拉创始人马克·塔彭宁等。

■ RISC体系结构的CPU芯片经历了三代：

- **第一代**：32位数据通路，支持**Cache**，软件支持较少，性能与**CISC**体系结构的产品相当。例如：**RISC I、MIPS、IBM801**
- **第二代**：具有单指令流水线，可同时执行多条指令，每个时钟周期发出一条指令。例如：**MIPS R3000**
- **第三代**：**64**位微处理器，采用超级流水线技术和超标量技术，提高了指令级的并行处理能力，每个时钟周期发出**2**条或**3**条指令。例如：**MIPS R4000**

表7.4：MIPS公司R系列RISC处理器比较

表7.5：不同公司第三代RISC处理器的性能比较

表 7.4 MIPS 公司 R 系列 RISC 处理器比较

机 种	R2000	R3000	R4000
宣布时间	1986	1988	1991
时钟频率	16.67 MHz	25/33 MHz	50/75 MHz
芯片规模	10 万晶体管	11.5 万晶体管	110 万晶体管
结构形式	流水线	流水线	超级流水线
寄存器集	32 × 32 位	32 × 32 位	32 × 64 位, 16 × 64 位
片上 Cache	—	—	16 KB
片外 Cache	最大 128 KB	最大 512 KB	128 KB ~ 4 MB
工艺	2 μm CMOS	1.2 μm CMOS	0.8 μm CMOS
功耗	3W	3.5W	
<u>SPEC 分</u>	11.2	17.6(25 MHz)	63(50 MHz)

SPEC (Standard Performance Evaluation Corporation, 标准性能评估组织)

CMOS, Complementary Metal Oxide Semiconductor, 互补金属氧化物半导体



MIPS公司

- **MIPS**技术公司是一家设计制造高性能、高档次及嵌入式**32**位和**64**位处理器的厂商，在**RISC**处理器方面占有重要地位。**1984**年，**MIPS**计算机公司成立。**1992**年，**SGI**收购了**MIPS**计算机公司。**1998**年，**MIPS**脱离**SGI**，成为**MIPS**公司。
- **MIPS**公司设计**RISC**处理器始于二十世纪八十年代初，**1986**年推出了**R2000**处理器，**1988**年推出了**R3000**处理器，**1991**年推出第一款**64**位商用微处理器**R4000**。之后又陆续推出**R8000**（于**1994**年）、**R10000**（于**1996**年）和**R12000**（于**1997**年）等型号。
- 随后，**MIPS**公司的战略发生变化，把重点放在嵌入式系统。**1999**年，**MIPS**公司发布**MIPS32**和**MIPS64**架构标准，为未来**MIPS**处理器的开发奠定了基础。新的架构集成了所有原来**MIPS**指令集，并且增加了许多更强大的功能。**MIPS**公司陆续开发了高性能、低功耗的**32**位处理器内核（**core**）**MIPS324Kc**与高性能**64**位处理器内核**MIPS645Kc**。**2000**年，**MIPS**公司发布了针对**MIPS32 4Kc**的版本以及**64**位**MIPS 64 20Kc**处理器内核。

表 7.5 第三代 RISC 处理器的性能比较

机 种	R4000	Alpha	Motorola 88110	Super SPARC	RS/6000	i860	C400
公司名称	MIPS	DEC	Motorola	Sun/TI	IBM	Intel	Intergraph
时钟频率 (MHz)	50/75	150/200	50	50/100	33	25/40/50	50
集成度 (万晶体管)	110	168	130	310	120	255	30
结构形式	超流水线	超标量	超标量	超标量	超标量	超长指令	超标量
寄存器集	32 × 64 16 × 64 浮	32 × 64 32 × 64 浮	32 × 64 32 × 64 (32 × 80)	32 × 32	32 × 64 32 × 64	32 × 32 16 × 64	32 × 32 16 × 64
片上 Cache	16 KB	16 KB	16 KB	36 KB	8 KB	32 KB	
片外 Cache	128 KB ~ 1 MB	最大可达 8 MB	256 KB ~ 1 MB	2 MB			128 KB
工艺	0.8 μm CMOS	0.75 μm CMOS	1 μm CMOS	0.8 μm CMOS			
功耗		23 W		8 W	4 W		7 W
SPEC 分	63 (50 MHz)	100(估计)	63.7(估计)	75(估计)	25.9	42	42

美国DEC公司即美国数字设备公司，英文：Digital Equipment Corporation，简称DEC

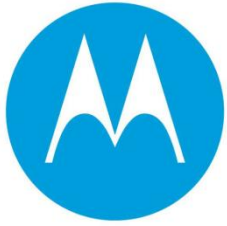
1957年诞生

DEC公司诞生于五十年代末，生产PDP-1人机对话型计算机，开创了小型机时代

PDP系列计算机、VAX系列计算机

随着PDP系列计算机技术的不断发展成熟，VAX战略把小型机的发展推上了顶端，终于与IBM比肩

到80年代末，DEC公司因战略问题开始走下坡路，最终在1998年被康柏收购，2001年惠普康柏宣布合并，DEC淡出历史舞台



MOTOROLA

Motorola公司

- **摩托罗拉**（**Motorola Inc**），原名：**Galvin Manufacturing Corporation**（加尔文制造公司），成立于**1928**年。**1947**年，改名为**Motorola**，从**1930**年代开始作为商标使用。
- **摩托罗拉总部**设在美国伊利诺伊州绍姆堡，位于**芝加哥市郊**。世界财富百强企业之一，是全球芯片制造、电子通讯的领导者。
- **2014**年**1**月**30**日，中国联想集团以**29**亿美元收购了摩托罗拉公司的手机业务。



Sun公司

- **Sun Microsystems**是IT及互联网技术服务公司（已被甲骨文（**Oracle**）收购）**Sun Microsystems** 创建于**1982**年。
- 主要产品是工作站及服务器。**1986**年在美国成功上市。**1992**年**Sun**推出了市场上第一台多处理器台式机**SPARC station 10 system**，并于**1993**年进入财富**500**强。
- **Sun**公司名称的由来很多人不知道，它其实是斯坦福大学校园网（**Stanford University Network**）的首字母缩写。当安迪·贝克托森（**Andy Bechtolsheim**）还是斯坦福大学研究生时，他设计出一种“三个百万”的小型图形计算机，称作图形工作站（**Graphic Work Station**）。贝克托森开发出原型机 **Sun-1** 后，便于 **1982** 年和斯科特·马可尼里（**Scott McNealy**）等斯坦福毕业生从学校出来创办了**Sun**公司。



TI公司

- 德州仪器 (**Texas Instruments**), 简称**TI**, 是全球领先的半导体公司, 为现实世界的信号处理提供创新的数字信号处理(**DSP**)及模拟器件技术。除半导体业务外, 还提供包括传感与控制、教育产品和数字光源处理解决方案。**TI**总部位于美国得克萨斯州的达拉斯, 并在**25**多个国家设有制造、设计或销售机构。



**TEXAS
INSTRUMENTS**



Intergraph公司

- **Intergraph**公司由前**IBM**工程师成立于**1969**年，是一家软件公司，主要的地理空间和工程软件的开发。该公司拥有三个主要部门，即电力和海事，鹰过程和地理空间内六角。它的总部设在亚拉巴马州亨茨维尔，美国。



Intergraph Corporation

ARM[®]

ARM公司

- 英国**ARM**公司是全球领先的半导体知识产权（**IP, Intellectual Property**）提供商。全世界超过**95%**的智能手机和平板电脑都采用**ARM**架构。**ARM**设计了大量高性价比、耗能低的**RISC**处理器、相关技术及软件。**2014**年基于**ARM**技术的全年全球出货量是**120**亿颗，从诞生到现在为止基于**ARM**技术的芯片有**600**亿颗。技术具有性能高、成本低和能耗省的特点。在智能机、平板电脑、嵌入控制、多媒体数字等处理器领域拥有主导地位。





ARM系列处理器

- **ARM7系列、ARM9系列、ARM9E系列、ARM10E系列、SecurCore系列、Intel的StrongARM、ARM11系列、Intel的Xscale。**
- **ARM公司在经典处理器ARM11以后的产品改用Cortex命名，并分成A、R和M三类，旨在为各种不同的市场提供服务。**
- **Cortex-A系列：Highest Performance（高性能）**
 - **Cortex-A53、Cortex-A57、Cortex-A76**
- **Cortex-R系列：Real-Time Processing（实时处理）**
 - **Cortex-R52**
- **Cortex-M系列：Lowest Power, Lower Cost（低功耗、低价格）**
 - **Cortex-M35P**



二、RISC 的主要特征

- 计算机执行程序所需要的时间**P**:

$$\mathbf{P = I \times C \times T}$$

- **I**: 高级语言程序编译后在机器上运行的机器指令数
- **C**: 执行每条机器指令所需的平均机器周期 (**CPI**)
- **T**: 每个机器周期的执行时间 (时钟周期)

- 表**7.6**: **RISC/CISC**的**I**、**C**、**T**比较

- **RISC**计算机的性能优于**CISC**计算机**2-5**倍

表 7.6 RISC/CISC 的 I, C, T 统计比较

	I	C	T
RISC	1.2 ~ 1.4	1.3 ~ 1.7	< 1
CISC	1	4 ~ 10	1

RISC计算机的性能优于**CISC**计算机**2-5**倍

$$(1/1.4) \times (4/1.7) = 1.68$$

$$(1/1.2) \times (10/1.3) = 6.4$$

二、RISC 的主要特征(续)

7个方面

- 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 流水线技术，一个时钟周期 内完成一条指令
- 采用 组合逻辑 实现控制器，不用微程序控制器
- 采用 优化 的 编译 程序

表 7.7 一些 RISC 机的指令条数

机器名	指令数	机器名	指令数
RISC II	39	ACORN	44
MIPS	31	INMOS	111
IBM801	120	IBM RT	118
MIRIS	64	HPPA	140
PYRAMID	128	CLIPPER	101
RIDGE	128	SPARC	89

RISC II指令系统举例

■ 指令种类

- 共有**39**条指令，分为**4**类：

- 寄存器-寄存器
- 取数/存数
- 控制转移指令
- 其他指令

■ 指令格式

- **2**种：短立即数格式、长立即数格式
- 指令长度：**32**位
- 图**7.23**

■ 寻址方式

- **2**种访存寻址方式：变址寻址、相对寻址
- 变址寄存器内容为**0** -> 直接寻址方式
- 位移量为**0** -> 寄存器间接寻址方式

MOV AX, (IX+23) 变址寻址

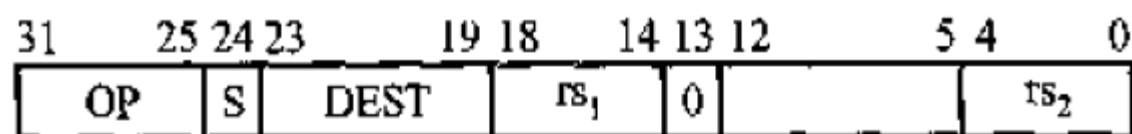
IX=0

MOV AX, (23) 直接寻址

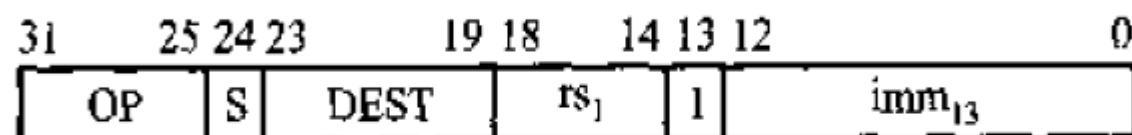
MOV AX, (AX+D) 相对寻址

D=0

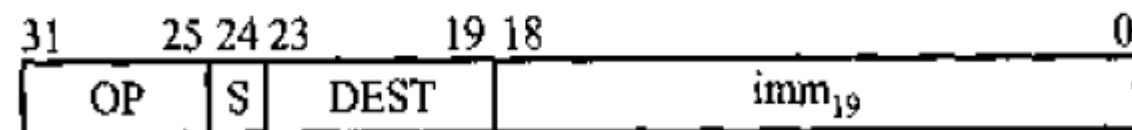
MOV AX, (AX) 寄存器间接寻址



(a) 第二源操作数在寄存器中的短立即数格式



(b) 第二源操作数为 imm₁₃ 的短立即数格式



(c) 长立即数格式

图 7.23 RISC II 的指令格式

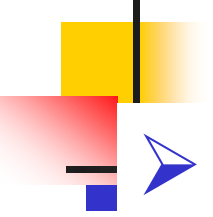


RISC指令系统的扩充

- 浮点指令
- 特权指令
 - 便于操作系统管理机器，防止用户破坏机器的运行环境
- 读后置数指令
 - 读 -> 修改 -> 写，用于寄存器与存储单元交换数据
- 一些简单的专用指令
 - 如乘法指令等

三、CISC 的主要特征

7个方面

- 
- 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

程序在嵌套时不必将寄存器内容保存到存储器中，故提高了速度

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 ， 采用寄存器窗口重叠技术

便于实现 指令流水（流水线工作）

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，有效支持高级语言编程

5. RISC 不易 实现 指令系统兼容

表 7.8 一些 CISC 与 RISC 微处理器的特征

特 征	CISC			RISC	
	IBM370/168	VAX11/780	Intel 80486	Motorola 88000	MIPS R4000
开发年份	1973	1978	1989	1988	1991
指令数	208	303	235	51	94
指令字长/B	2 ~ 6	2 ~ 57	1 ~ 11	4	4
寻址方式	4	22	11	3	1
通用寄存器数	16	16	8	32	32
控制存储器容量/Kb	420	480	246	—	—
Cache 容量/Kb	64	64	8	16	128

RISC与CISC技术的融合

ARM (RISC) 和x86 (CISC) 的技术差异

- **RISC**和**CISC**，这一对冤家，从诞生之日开始就处在不停的纠缠之中。直到今天，两者经过多年的发展后，都在各自领域打开了一片天地，并且相互渗透。
- **RISC**专注高性能、高性能功耗比、小体积以及移动设备领域，**CISC**专注桌面、高性能和民用市场。
- 现在，**RISC**的代表是**ARM**，而**CISC**的代表则是我们耳熟能详的**x86**。
- 那么，他们的技术差异在哪里？究竟是怎样的技术分歧带来了两者如此大的差别呢？



- 虽然从原理来看，**RISC**和**CISC**可谓井水不犯河水。但**RISC**和**CISC**在发展过程中，彼此反而取长补短，各有所得。
- 对**CISC**来说，指令集本身随着计算要求不断发展，肯定会越来越多。**CISC**继续发展下去，其实际**CPU**产品的晶体管数量会难以抑制地上升，性能功耗比和成本表现很难让人满意。从设计角度来看，**CISC**指令集长度不固定、执行时间也不固定、设计困难很多，很难找出一条高效率的通用设计道路来完成指令的执行。此外，由于**CISC**处理器和存储器之间的速度差距，缓存变得越来越重要。这也意味着**CPU**本身需要更为精简高效，节省的空间需要用于容纳越来越重要的各级缓存。
- 为了解决这些问题，**现代的CISC处理器开始认真学习RISC的思想**。**CISC**的问题在于指令集复杂多变，为每一个指令制定专门的硬件优化显然不可能。那么，可不可以换一个思路呢？**将那些最常使用的指令集挑选出来，然后为其进行专门优化，就可以大大提高效率；至于不常用的指令，则可以用几个基础指令组合的方式完成——这正是RISC的思想**。有所不同的是，**RISC**让思想完成在指令层面，而**CISC**将这个思想实现在硬件层面。

- 以英特尔的**Nehalem**或者**AMD**的**K10**处理器为例。
- 首先，这些**x86**处理器内部都会使用“微指令”。所谓微指令，就是一些基础的指令，**CISC**指令中大部分都可以被拆分为几条简单而固定的微指令。
- 其次，**CPU**内部设计了“翻译单元”，一般是由解码单元来执行的。在运行中，**CPU**接受一条**x86**指令，然后解码单元将接收到的比较复杂的**X86**指令拆解为一个或几个微指令。比如**Nehalem**设计了**3**个简单的解码单元和**1**个复杂的解码单元，可以将**x86**指令解码拆分、“翻译”为**1~4**条微指令。
- 第三，**CPU**内部针对这些微指令会做出充足的优化，让其执行效率和速度都达到令人满意的程度。当然，在解码处理的过程中，不是所有的**x86**指令都会得到平均对待。那些最常用的指令比如**mov**、**push**、**call**、**cmp**、**add**等会被重点、优先、加速处理，不常用的指令要么被拆分为常用指令，要么进入普通循环进行处理。效率虽然有影响，但考虑到其使用几率很低，因此这样的设计完全可以接受。在微指令解码和处理过程中，如何更有效率、更为高效地执行**x86**命令；微指令应该如何表达、运行；微指令和**x86**指令的关系以及哪些微指令是最常用、最优先的指令，成为最影响**CPU**性能的核心内容。

- 在采用了**RISC**思想、对**x86**处理器的设计进行革新后，如今的**CISC**处理器基本上可以解决**CISC**指令复杂、体积庞大，晶体管耗费多等问题。对厂商来说，一个设计优秀的**x86**处理器的解码和流水线核心可以维持数代发展而不落伍。厂商可以在发展过程中不断对已经设计好的核心进行调整和调配，在缓存、总线配置上进行更改以获取更好的性能。
- **CISC**借鉴**RISC**的思想，让自己获得了新生。相对来说，**RISC**对**CISC**也有借鉴，但不算太多。**RISC**指令简单并且相对固定，处理快速，在设计上甚至可以使用更长的流水线来达到高频率，并最终获得更优秀的效能。但**RISC**的主要问题在于指令集简单，因此在处理一些比较复杂的应用时，存储器需要读入的指令总数耗费时间更多，部分场合下性能表现不理想也是**RISC**的硬伤。
- 因此，在**RISC**的发展中，**RISC**也在逐渐注入**CISC**的思想。比如紧跟时代加入一些新的指令集，更进一步优化内部架构，运行周期变成不固定周期等。**RISC**发展到现在，指令也逐渐增多，浮点计算等重要性能也日益强大。以**ARM**为例，不但逐渐增强浮点计算性能、新增专门的浮点指令，还计划在现有的基础上开展高性能**ARM**处理器的发展，以增强未来应对市场变化特别是对**x86**处理器竞争的能力。

- 未来的处理器，功耗更低，效能更高。
- 从目前**CPU**的发展来看，无论是**ARM**还是**x86**，无论是**CISC**还是**RISC**，除了努力巩固自己的性能优势，加强产品的性能外，还积极吸取对方产品的特色，取长补短，期望有所突破。不过无论如何，未来的**CPU**肯定在朝着高性能、低功耗的方向发展。目前移动计算大潮已经来临，竞争日趋激烈。但说到底就是性能功耗比的竞争，谁能在低功耗下提供高性能，谁就有希望获得成功。未来的处理器，功耗将更低，效能会更高。

本章小结

- 指令 = 操作码 (OP) + 操作数 (地址码)

- 扩展操作码技术

操作码的位数随地址数的减少而增加

	OP	A ₁	A ₂	A ₃	指令的总长度固定 (16位)
4 位操作码	0000 0001 ⋮ 1110	A ₁ A ₁ ⋮ A ₁	A ₂ A ₂ ⋮ A ₂	A ₃ A ₃ ⋮ A ₃	最多15条三地址指令
8 位操作码	1111 1111 ⋮ 1111	0000 0001 ⋮ 1110	A ₂ A ₂ ⋮ A ₂	A ₃ A ₃ ⋮ A ₃	最多15条二地址指令
12 位操作码	1111 1111 ⋮ 1111	1111 1111 ⋮ 1111	0000 0001 ⋮ 1110	A ₃ A ₃ ⋮ A ₃	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

- 指令字长 (指令翻译成机器码的长度)
 - 指令字长固定 (**RISC**计算机): 如**32**位
 - 指令字长可变 (**CISC**计算机): **8**位、**16**位、**24**位.....



本章小结

- 数据在存储器中的存放方式
 - 边界对准：(例如，**32**位的字存放在**0000H-0003H**存储单元中)
 - 边界未对准：(例如，**32**位的字存放在**0002H-0005H**存储单元中)

- 指令的类型（操作类型、操作码的类型）
 - 数据传送指令
 - 算术逻辑指令
 - 移位指令
 - 转移指令
 - 无条件转移
 - 条件转移
 - 子程序调用和返回
 - 陷进指令（例如，**INT 21H**）
 - 输入输出指令
 - 其他指令



本章小结

■ 寻址方式

- 指令寻址——确定下一条指令的地址
- 数据寻址——确定操作数的地址

■ 指令寻址

- **PC+1 -> PC** **PC+2->PC** **PC+3->PC**
- 跳转：转移指令、子程序调用与返回指令、陷进指令、中断

本章小结

■ 数据寻址

1. 立即数寻址: **MOV AX, 2300H**
2. 直接寻址: **MOV BX, [1000H]**
3. 隐含寻址: **MUL BL** **AL*BL -> AX**
4. 间接寻址: **MOV AL, [[2300H]]**
5. 寄存器寻址: **MOV AX, BX**
6. 寄存器间接寻址: **MOV CH, [SI]**
7. 基址寻址: **MOV AX, ARRAY[BP]** **MOV AX, ARRAY[BX]**
8. 变址寻址: **MOV AX, 10H[SI]** **MOV TABLE[DI], 12H**
9. 基址变址寻址: **MOV AX, 200H[BX][SI]**
10. 相对寻址: **JMP LOOP** 怎么计算转移指令的位移量? 例7.2
11. 堆栈寻址: **PUSH AX** **POP AX** 例7.3 (计算PC、SP、堆栈栈顶的内容)

本章小结

- 指令系统的兼容性：向上兼容（**80286**的指令系统要兼容**8086**的指令系统）
- **RISC: Reduced Instruction Set Computer**, 精简指令系统计算机
- **CISC: Complex Instruction Set Computer**, 复杂指令系统计算机
- 计算机执行程序所需要的时间 $P = I * C * T$
 - **I**: 高级语言程序编译后在机器上运行的指令数
 - **C**: 执行每条机器指令所需的平均周期（需要多少个时钟周期？） **CPI**
 - **T**: 每个机器周期的执行时间（即时钟周期，主频的倒数）

表 7.6 RISC/CISC 的 I, C, T 统计比较			
	I	C	T
RISC	1.2 ~ 1.4	1.3 ~ 1.7	< 1
CISC	1	4 ~ 10	1

- **RISC** 计算机的性能优于 **CISC** 计算机 **2-5**倍



本章小结

■ RISC的主要特征（7个方面）

- 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实现
- 指令长度固定、指令格式简单、指令种类少、寻址方式少
- 只有 **LOAD / STORE**指令访存
- 采用流水技术，一个时钟周期内能完成（执行）一条指令
- 采用组合逻辑实现控制器，不用微程序控制器
- **CPU** 中有多个通用寄存器
- 采用优化的编译程序

■ CISC的主要特征（7个方面）

- 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、指令格式复杂、指令种类多、寻址方式多
- 访存指令不受限制
- 大多数指令需要多个时钟周期才能执行完毕
- 采用微程序控制器
- **CPU** 中设有专用寄存器
- 难以用优化编译生成高效的目代码



本章小结

- **RISC**计算机与**CISC**计算机的比较（**4**个优点、**1**个缺点）
 - **RISC** 更能充分利用 **VLSI** 芯片的面积
 - **RISC** 更能提高计算机运算速度
 - 指令数 少、指令格式简单、寻址方式少
 - 通用寄存器多，采用组合逻辑控制，采用寄存器窗口重叠技术
 - 便于实现指令流水
 - **RISC** 便于设计，可降低成本，提高可靠性
 - **RISC** 有利于编译程序代码优化
 - **RISC** 不易实现指令系统兼容



第13次作业——习题（P335-P336）

■ 7.1

■ 7.2

■ 7.3

■ 7.6

■ 7.7

■ 7.9

■ 7.10

■ 7.14

■ 7.20

■ 7.21



关于作业的提交

- **1周内**必须提交（上传到学院的**FTP**服务器上），否则认为是迟交作业；如果期末仍然没有提交，则认为是未提交作业
 - 作业完成情况成绩=第**1**次作业提交情况*第**1**次作业评分+第**2**次作业提交情况*第**2**次作业评分+.....+第**N**次作业提交情况*第**N**次作业评分
 - 作业评分：**A**（好）、**B**（中）、**C**（差）三挡
 - 作业提交情况：按时提交（**1.0**）、迟交（**0.5**）、未提交（**0.0**）
- 请采用电子版的格式（**Word**文档）上传到**FTP**服务器上，文件名取“学号+姓名+第**X**次作业.doc”
 - 例如：**11920192203642+袁佳哲+第13次作业.doc**
- 第**13**次作业提交的截止日期为：**2021年5月26日晚上24点**



The End

Thanks