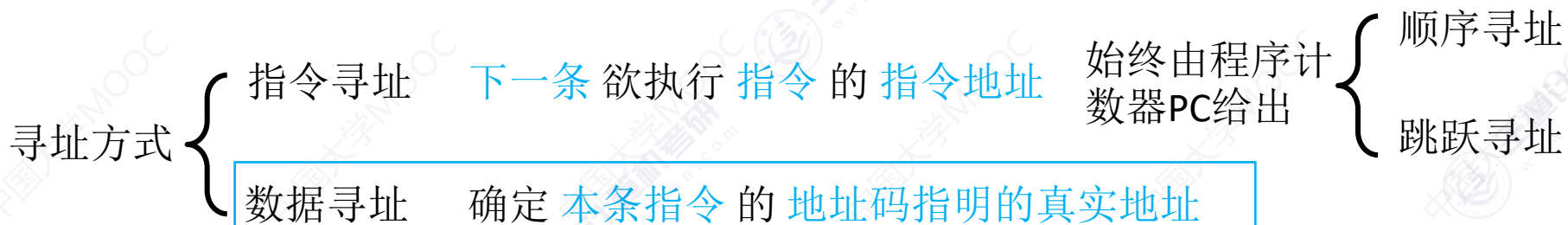


本节内容

# 数据寻址2

(偏移寻址)

# 指令寻址 v.s. 数据寻址



以某个地址作为起点  
形式地址视为“偏移量”

0	LDA	1000
1	ADD	1001
2	DEC	1200
3	JMP	7
4	LDA	2000
5	SUB	2001
6	INC	
7	LDA	1100
8	...	

起始

100

100	LDA	1000
101	ADD	1001
102	DEC	1200
103	JMP	7
104	LDA	2000
105	SUB	2001
106	INC	
107	LDA	1100
108	...	

PC

100

101

102

103

104

105

106

107

108

100	LDA	1000
101	ADD	1001
102	DEC	1200
103	JMP	3
104	LDA	2000
105	SUB	2001
106	INC	
107	LDA	1100
108	...	

# 知识总览

一地址指令

操作码 (OP)

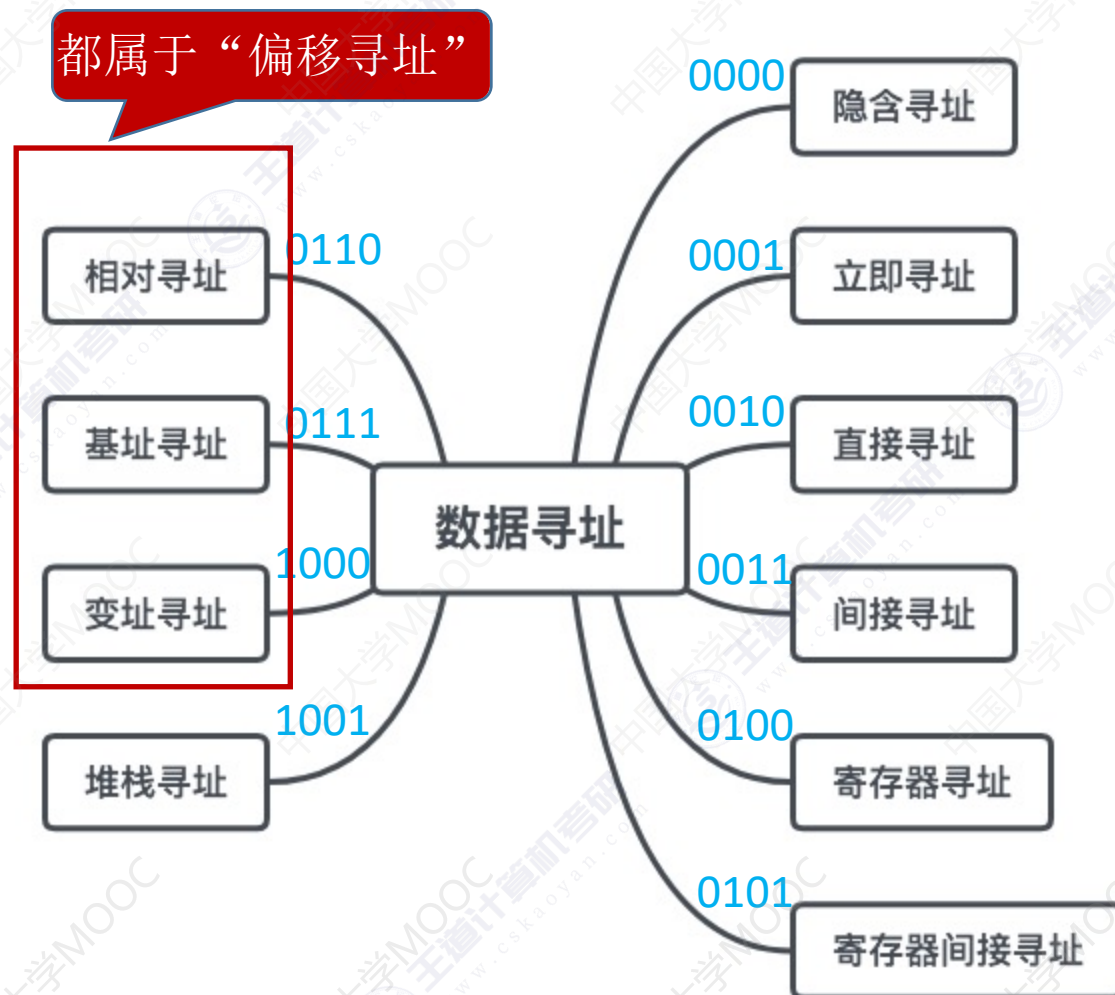
寻址特征

形式地址 (A)

求出操作数的真实地址，称为有效地址(EA)。

都属于“偏移寻址”

寻址方式位



# 偏移寻址



区别在于偏移的“起点”不一样

不一样  
我们不一样



基址寻址：以程序的起始存放地址作为“起点”  
变址寻址：程序员自己决定从哪里作为“起点”  
相对寻址：以程序计数器PC所指地址作为“起点”

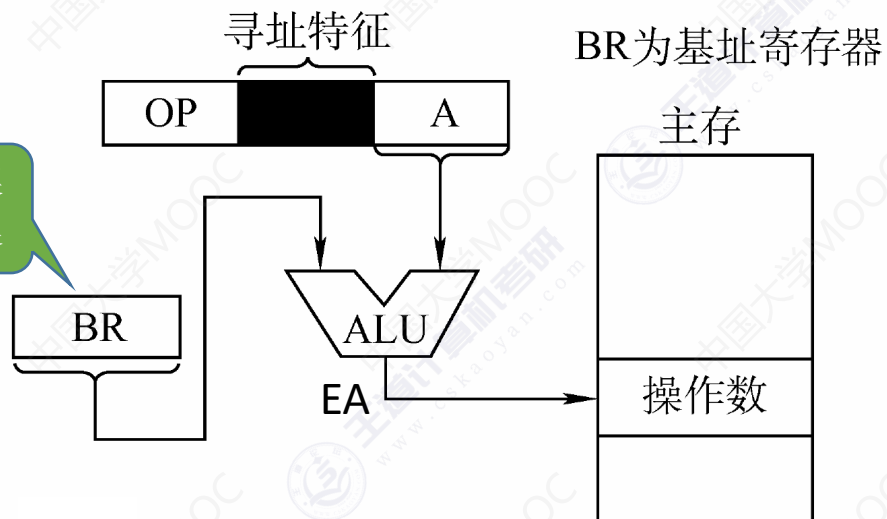
# 基址寻址

注: BR——base address register  
EA——effective address

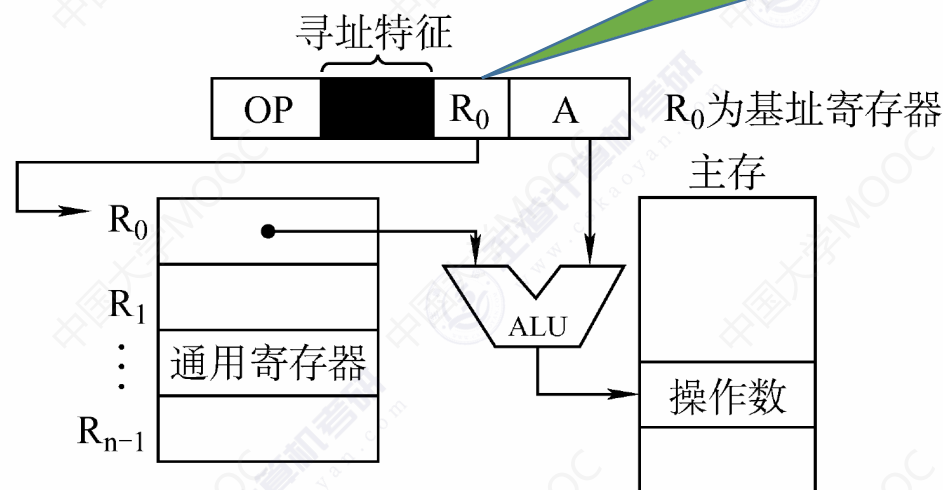
基址寻址: 将CPU中**基址寄存器 (BR)**的内容加上指令格式中的形式地址A, 而形成操作数的有效地址, 即  $EA = (BR) + A$ 。

在指令中指明, 要将哪个通用寄存器作为基址寄存器使用

专门的基址寄存器



(a) 采用专用寄存器BR作为基址寄存器



(b) 采用通用寄存器作为基址寄存器

Tips: 可对比操作系统第三章第一节学习, OS课中的“重定位寄存器”就是“基址寄存器”

要用几个bit 指明寄存器?

根据通用寄存器总数判断

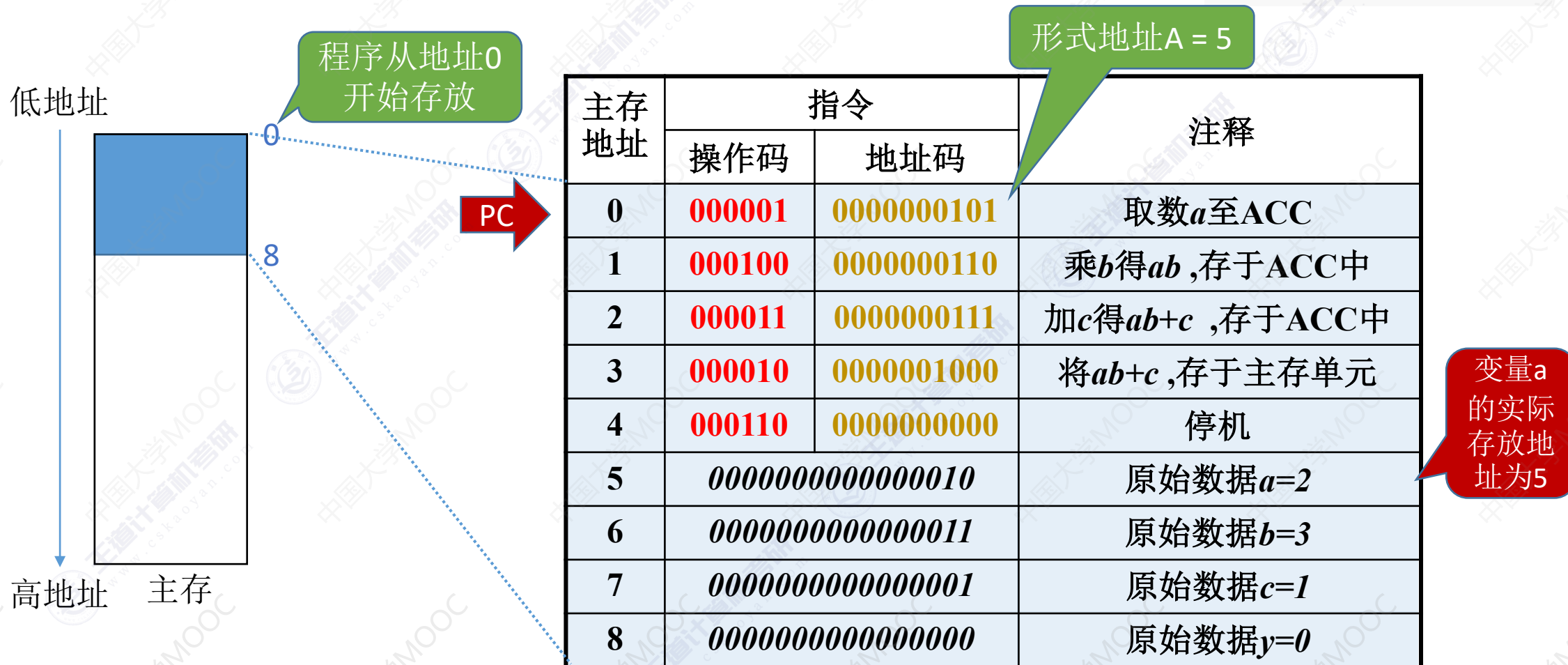


稍加思考

# 基址寻址的作用

基址寻址：将CPU中基址寄存器（BR）的内容加上指令格式中的形式地址A，而形成操作数的有效地址，即 $EA=(BR)+A$ 。

```
int a=2,b=3,c=1,y=0;
void main(){
    y=a*b+c;
}
```





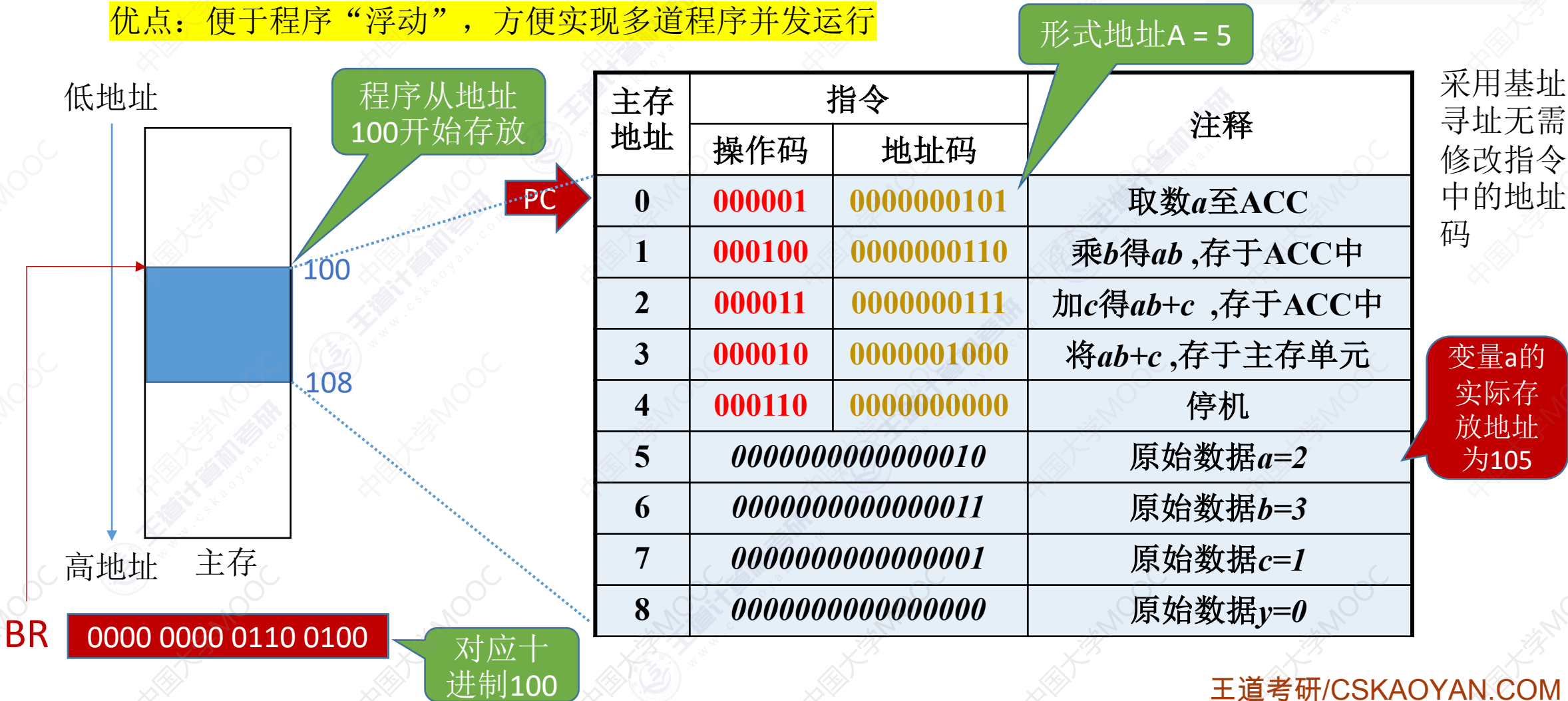
拓展：程序运行前，CPU将BR的值修改为该程序的起始地址（存在操作系统PCB中）

# 基址寻址的作用

```
int a=2,b=3,c=1,y=0;
void main(){
    y=a*b+c;
}
```

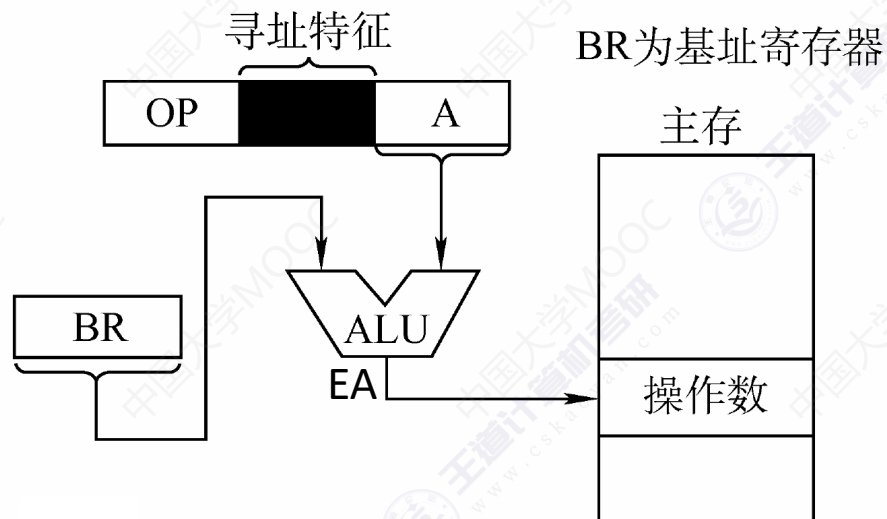
基址寻址：将CPU中基址寄存器（BR）的内容加上指令格式中的形式地址A，而形成操作数的有效地址，即EA=(BR)+A。

优点：便于程序“浮动”，方便实现多道程序并发运行

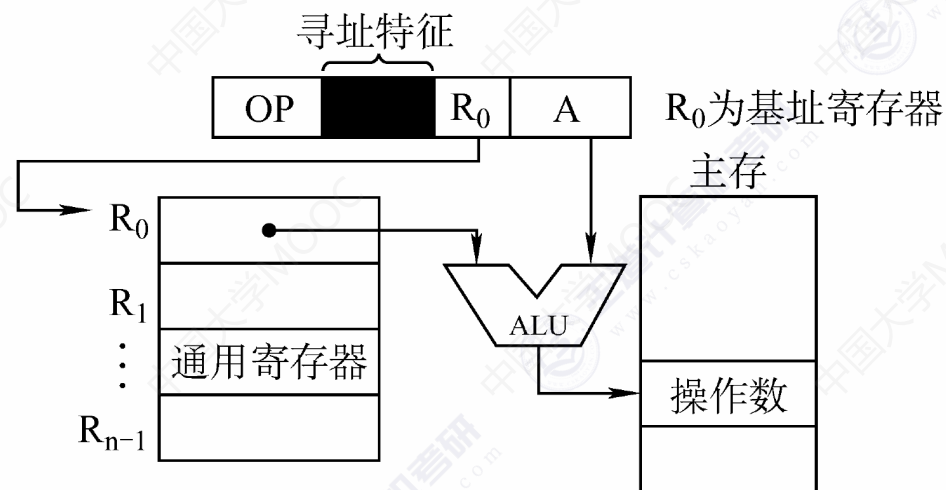


## 基址寻址

基址寻址：将CPU中**基址寄存器（BR）**的内容加上指令格式中的形式地址A，而形成操作数的有效地址，即 **$EA=(BR)+A$** 。



(a) 采用专用寄存器BR作为基址寄存器



(b) 采用通用寄存器作为基址寄存器

注：基址寄存器是**面向操作系统**的，其**内容由操作系统或管理程序确定**。在程序执行过程中，基址寄存器的内容不变（作为基地址），形式地址可变（作为偏移量）。

当采用通用寄存器作为基址寄存器时，可由**用户决定哪个寄存器作为基址寄存器**，但其**内容仍由操作系统确定**。

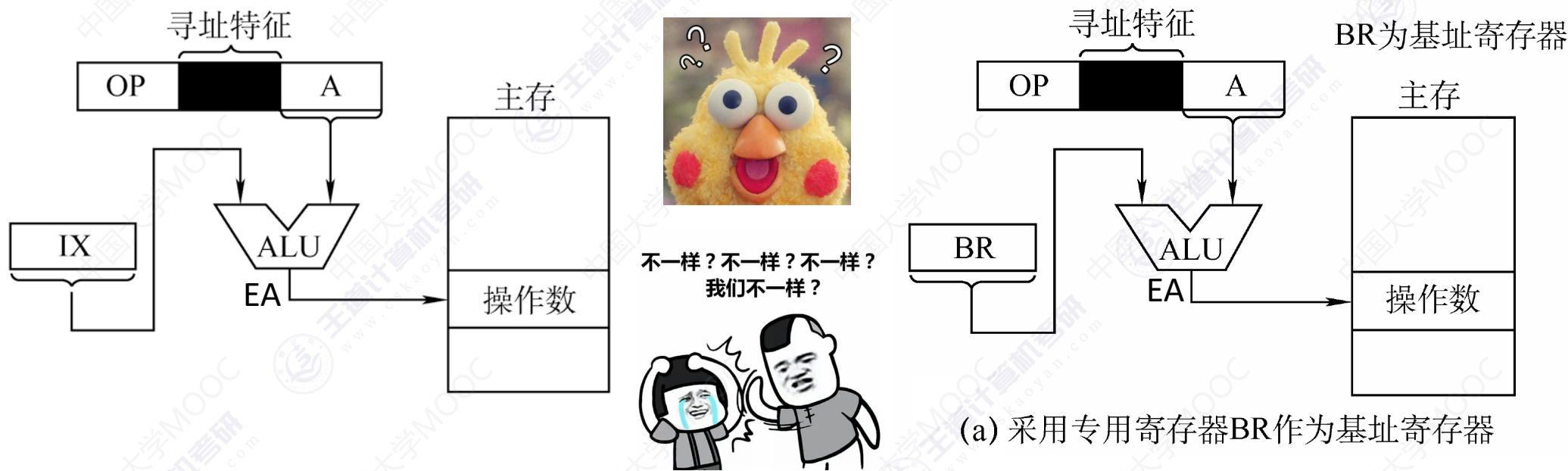
优点：可扩大寻址范围（基址寄存器的位数大于形式地址A的位数）；用户不必考虑自己的程序存于主存的哪一空间区域，故**有利于多道程序设计**，以及可用于**编制浮动程序**（整个程序在内存里边的浮动）。



## 变址寻址

注：IX — index register

变址寻址：有效地址EA等于指令字中的形式地址A与变址寄存器IX的内容相加之和，即 $EA = (IX) + A$ ，其中IX可为变址寄存器（专用），也可用通用寄存器作为变址寄存器。



注：变址寄存器是面向用户的，在程序执行过程中，变址寄存器的内容可由用户改变（IX作为偏移量），形式地址A不变（作为基地址）。

基址寻址中，BR保持不变作为基地址，A作为偏移量

## 变址寻址的作用

注：此处未添加“寻址特征”位，但实际上每条指令都会指明寻址方式。  
此处讲解仅用口头描述

```
for(int i=0; i<10; i++){  
    sum += a[i];  
}
```

立即寻址

ACC

0

直接寻址

主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0（立即数）	立即数 0 → ACC
1	ACC加法	12（a[0]地址）	(ACC)+a[0] → ACC
2	ACC加法	13（a[1]地址）	(ACC)+a[1] → ACC
...	ACC加法	14	(ACC)+a[2] → ACC
9	...	...	.....
10	ACC加法	21	(ACC)+a[9] → ACC
11	从ACC存数	22	(ACC)→ sum变量
12	随便什么值		a[0]
13	随便什么值		a[1]
...	...		...
21	随便什么值		a[9]
22	初始为0		sum变量

是时候召唤  
“变址寻址”  
了！



# 变址寻址的作用

```
for(int i=0; i<10; i++){
    sum += a[i];
}
```

ACC

0

IX

10

在数组处理过程中，可设定A为数组的首地址，不断改变变址寄存器IX的内容，便可很容易形成数组中任一数据的地址，特别适合编制循环程序。

立即寻址

变址寻址

立即寻址

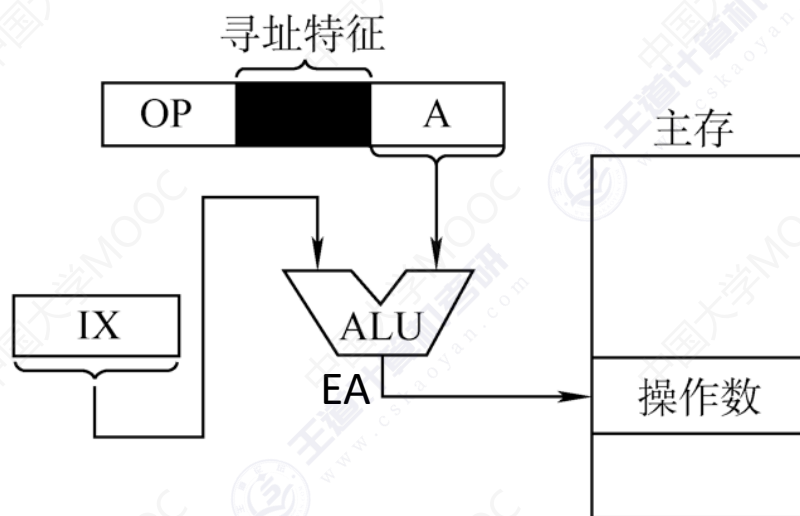
直接寻址

主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数 0 → ACC
1	取数到IX	#0	立即数 0 → IX
2	ACC加法	7 (数组始址)	(ACC)+(7+(IX))→ ACC
3	IX加法	#1	(IX) + 1→ IX
4	IX比较	#10	比较10-(IX)
5	条件跳转	2	若结果>0 则PC跳转到2
6	从ACC存数	17	(ACC)→ sum变量
7	随便什么值		a[0]
8	随便什么值		a[1]
9	随便什么值		a[2]
...	...		...
16	随便什么值		a[9]
17	初始为0		sum变量

留个坑

## 变址寻址

变址寻址：有效地址EA等于指令字中的形式地址A与变址寄存器IX的内容相加之和，即 $EA = (IX) + A$ ，其中IX可为变址寄存器（专用），也可用通用寄存器作为变址寄存器。



注：变址寄存器是面向用户的，在程序执行过程中，变址寄存器的内容可由用户改变（作为偏移量），形式地址A不变（作为基地址）。

优点：在数组处理过程中，可设定A为数组的首地址，不断改变变址寄存器IX的内容，便可很容易形成数组中任一数据的地址，特别适合编制循环程序。

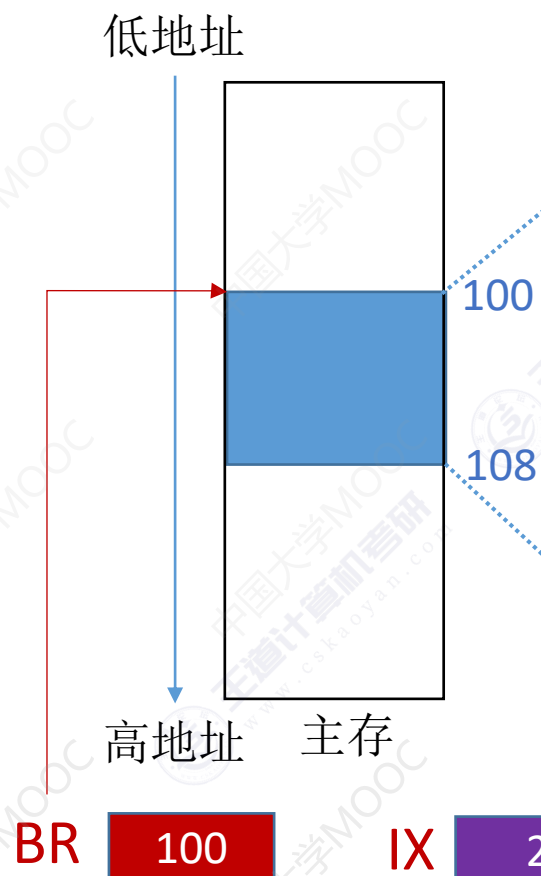
## 基址&变址复合寻址

注：实际应用中往往需要多种寻址方式复合使用（可理解为复合函数）

基址寻址： $EA=(BR)+A$

变址寻址： $EA=(IX)+A$

先基址后变址寻址： $EA=(IX)+(BR)+A$



主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数 0 → ACC
1	取数到IX	#0	立即数 0 → IX
2	ACC加法	7 (数组始址)	$(ACC) + (7 + (IX)) \rightarrow ACC$
3	IX加法	#1	$(IX) + 1 \rightarrow IX$
4	IX比较	#10	比较 $10 - (IX)$
5	条件跳转	2	若结果 > 0 则PC跳转到2
6	从ACC存数	17	$(ACC) \rightarrow \text{sum变量}$
7	随便什么值		a[0]
8	随便什么值		a[1]
9	随便什么值		a[2]
...	...		...
16	随便什么值		a[9]
17	初始为0		sum变量

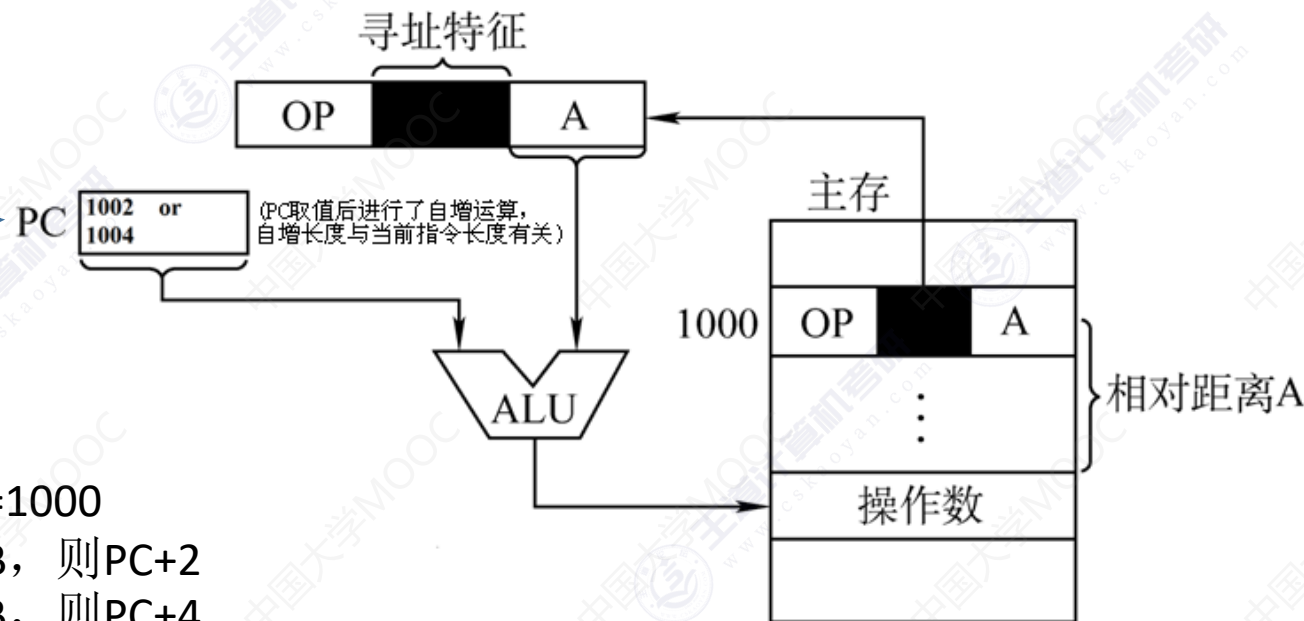


## 相对寻址

相对寻址：把程序计数器PC的内容加上指令格式中的形式地址A而形成操作数的有效地址，即 $EA=(PC)+A$ ，其中A是相对于PC所指地址的位移量，可正可负，补码表示。

注：王道书的小错误——“A是相对于当前指令地址的位移量” ❌

取出当前指令后，PC+“1”指向下一条指令



当前指令存放地址=1000  
若当前指令字长=2B，则PC+2  
若当前指令字长=4B，则PC+4

因此取出当前指令后PC可能为 1002 or 1004



## 相对寻址的作用

```
for(int i=0; i<10; i++){
    sum += a[i];
}
```

问题：随着代码越写越多，  
你想挪动for循环的位置

注：站在  
汇编语言  
程序员的角度思考

for循环主体

直接寻址

主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数 0 → ACC
1	取数到IX	#0	立即数 0 → IX
2	ACC加法	7 (数组始址)	(ACC)+(7+(IX))→ ACC
3	IX加法	#1	(IX) + 1 → IX
4	IX比较	#10	比较10-(IX)
5	条件跳转	2	若结果>0 则PC跳转到2
6	从ACC存数	17	(ACC)→ sum变量
7	随便什么值		a[0]
8	随便什么值		a[1]
9	随便什么值		a[2]
...	...		...
16	随便什么值		a[9]
17	初始为0		sum 变量

## 相对寻址的作用

```
for(int i=0; i<10; i++){  
    sum += a[i];  
}
```

问题：随着代码越写越多，  
你想挪动for循环的位置

注：站在  
汇编语言  
程序员的角度思考

for循环主体

采用直接寻址  
会出现错误

PC

M+4

主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数 0 → ACC
1	取数到IX	#0	立即数 0 → IX
2	...	...	其他代码
3	...	...	其他代码
4	...	...	其他代码
5	...	...	其他代码
...	...	...	其他代码
M	ACC加法	7 (数组始址)	(ACC)+(7+(IX))→ ACC
M+1	IX加法	#1	(IX) + 1 → IX
M+2	IX比较	#10	比较10-(IX)
M+3	条件跳转	2	若结果>0 则PC跳转到2
M+4	...	...	...
...	...	...	...

拓展：ACC加法指令的地址码，可采用“分段”方式解决，即程序段、数据段分开。

```
for(int i=0; i<10; i++){  
    sum += a[i];  
}
```

问题：随着代码越写越多，你想挪动for循环的位置

相对寻址： $EA=(PC)+A$ ，其中A是相对于PC所指地址的位移量，可正可负，补码表示

优点：这段代码在程序内浮动时不用更改跳转指令的地址码

## 相对寻址的作用

主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数0 → ACC
1	取数到IX	#0	立即数0 → IX
2	...	...	其他代码
3	...	...	其他代码
4	...	...	其他代码
5	...	...	其他代码
...	...	...	其他代码
M	ACC加法	7（数组始址）	$(ACC)+(7+(IX)) \rightarrow ACC$
M+1	IX加法	#1	$(IX) + 1 \rightarrow IX$
M+2	IX比较	#10	比较10-(IX)
M+3	条件跳转	-4(补码表示)	若结果>0 则PC跳转到M
M+4	...	...	...
...	...	...	...

for循环主体

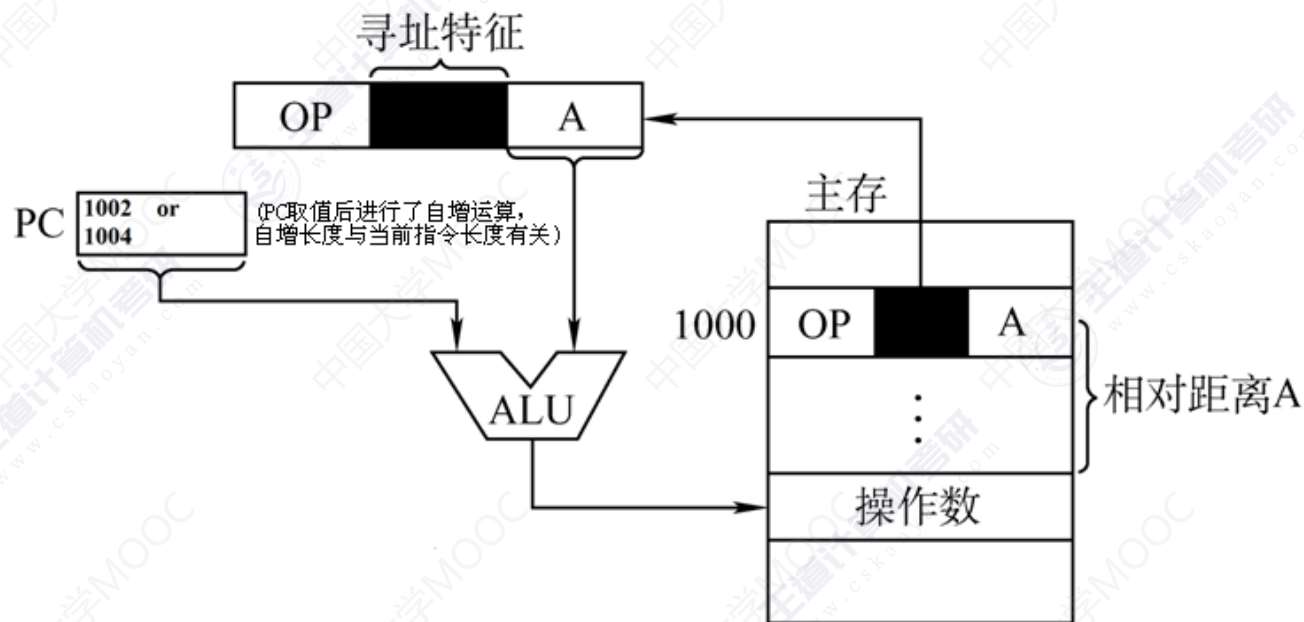
用相对寻址

PC

M+4

## 相对寻址

相对寻址：把程序计数器PC的内容加上指令格式中的形式地址A而形成操作数的有效地址，即 $EA=(PC)+A$ ，其中A是相对于PC所指地址的位移量，可正可负，补码表示。



优点：操作数的地址不是固定的，它随着PC值的变化而变化，并且与指令地址之间总是相差一个固定值，因此便于程序浮动（一段代码在程序内部的浮动）。

相对寻址广泛应用于转移指令。

## 本节回顾

寻址方式	有效地址	访存次数(指令执行期间)
隐含寻址	程序指定	0
立即寻址	A即是操作数	0
直接寻址	$EA=A$	1
一次间接寻址	$EA=(A)$	2
寄存器寻址	$EA=R_i$	0
寄存器间接一次寻址	$EA=(R_i)$	1
转移指令 相对寻址	$EA=(PC)+A$	1
多道程序 基址寻址	$EA=(BR)+A$	1
循环程序 变址寻址 数组问题	$EA=(IX)+A$	1

### 偏移寻址

注意：取出当前指令后，PC会指向下一条指令，相对寻址是相对于下一条指令的偏移



高级语言视角:

```
if (a>b){
```

```
    ...  
} else {  
    ...  
}
```

汇编语言中, **条件跳转指令**有很多, 如 `je 2` 表示当比较结果为 `a=b` 时跳转到2  
`jg 2` 表示当比较结果为 `a>b` 时跳转到2

硬件视角:

- 通过“**cmp指令**”比较 `a` 和 `b` (如 `cmp a, b`), 实质上是用 `a-b`
- 相减的结果信息会记录在程序状态字寄存器中 (PSW)
- 根据PSW的某几个标志位进行条件判断, 来决定是否转移

有的机器把PSW称为“标志寄存器”

PSW中有几个比特位记录上次运算的结果

- 进位/借位标志 **CF**: 最高位有进位/借位时 `CF=1`
- 零标志 **ZF**: 运算结果为0则 `ZF=1`, 否则 `ZF=0`
- 符号标志 **SF**: 运算结果为负, `SF=1`, 否则为0
- 溢出标志 **OF**: 运算结果有溢出 `OF=1` 否则为0

## 硬件如何实现数的“比较”

注: 无条件转移指令 `jmp 2`, 就不会管PSW的各种标志位

主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数 0 → ACC
1	取数到IX	#0	立即数 0 → IX
2	ACC加法	7 (数组始址)	(ACC)+(7+(IX))→ ACC
3	IX加法	#1	(IX) + 1→ IX
4	IX比较	#10	比较10-(IX)
5	条件跳转	2	若结果>0 则PC跳转到2
6	从ACC存数	17	(ACC)→ sum变量
7	随便什么值		a[0]
8	随便什么值		a[1]
9	随便什么值		a[2]
...	...		...
16	随便什么值		a[9]
17	初始为0		sum变量





公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研