

本节内容

数据寻址2  
(偏移寻址)

王道考研/CSKAOYAN.COM

1

指令寻址 v.s. 数据寻址

寻址方式

指令寻址

下一条欲执行指令的指令地址

始终由程序计数器PC给出

顺序寻址

数据寻址

确定本条指令的地址码指明的真实地址

跳跃寻址

以某个地址作为起点  
形式地址视为“偏移量”

起始

PC

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

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

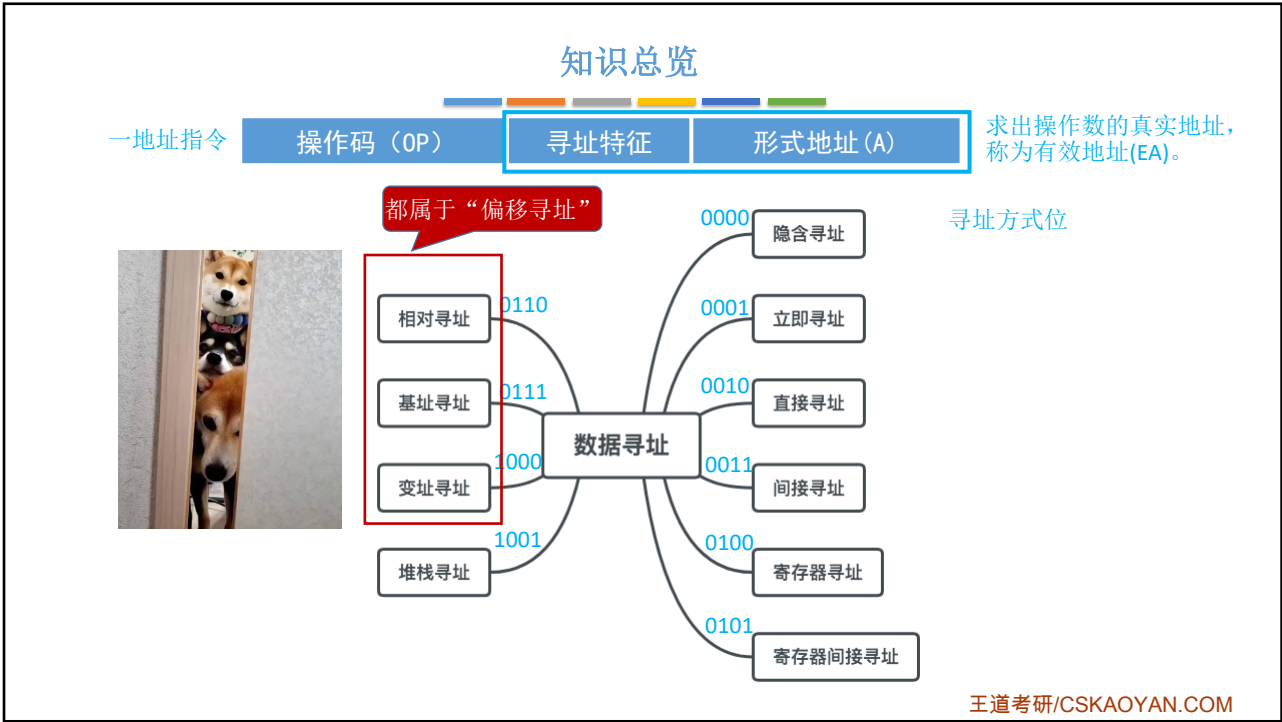
100	LDA	1000
101	ADD	1001
102	DEC	1200
103	JMP	3
104	LDA	2000
105	SUB	2001
106	INC	
107	LDA	1100
108	...	

王道考研/CSKAOYAN.COM

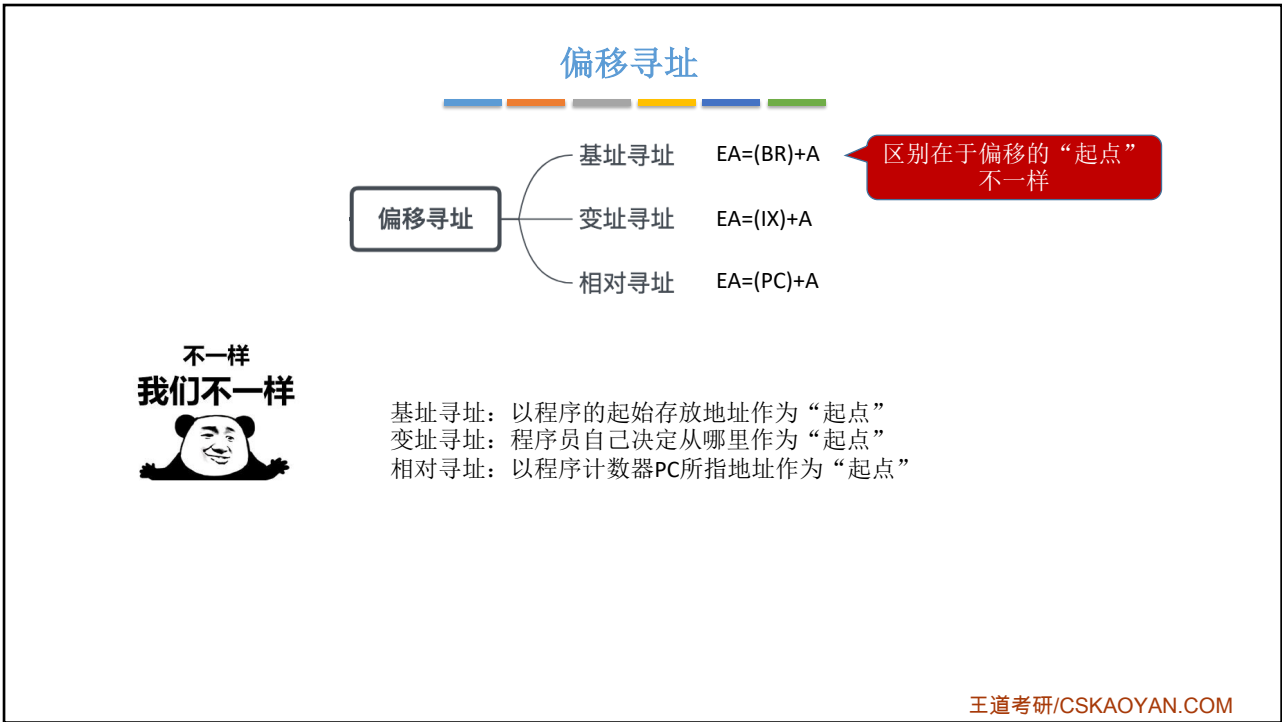
2

王道考研/CSKAOYAN.COM

1



3



4

### 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

寻址特征

OP A

BR

ALU

EA

主存

操作数

(a) 采用专用寄存器BR作为基址寄存器

寻址特征

OP R<sub>0</sub> A

R<sub>0</sub>为基址寄存器

主存

操作数

(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

王道考研/CSKAOYAN.COM

5

### 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;
}
```

程序从地址0开始存放

形式地址A = 5

低地址

高地址

主存

主存地址	指令		注释
	操作码	地址码	
0	000001	000000101	取数a至ACC
1	000100	000000110	乘b得ab,存于ACC中
2	000011	000000111	加c得ab+c,存于ACC中
3	000010	0000001000	将ab+c,存于主存单元
4	000110	0000000000	停机
5	000000000000010		原始数据a=2
6	000000000000011		原始数据b=3
7	000000000000001		原始数据c=1
8	000000000000000		原始数据y=0

变量a的实际存放地址为5

王道考研/CSKAOYAN.COM

6

王道考,cskaoyan.com

3

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）

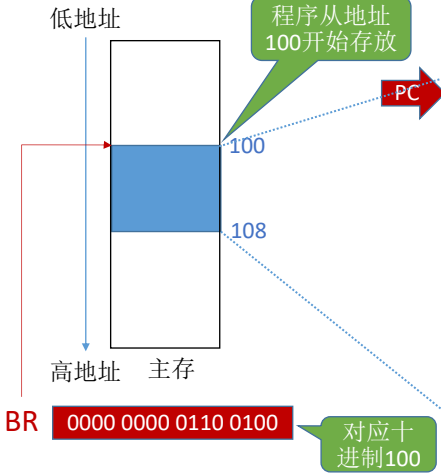
### 基址寻址的作用

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;
}
```

优点：便于程序“浮动”，方便实现多道程序并发运行

形式地址A = 5



主存地址	指令		注释
	操作码	地址码	
0	000001	0000000101	取数a至ACC
1	000100	0000000110	乘b得ab,存于ACC中
2	000011	0000000111	加c得ab+c,存于ACC中
3	000010	0000001000	将ab+c,存于主存单元
4	000110	0000000000	停机
5	0000000000000010		原始数据a=2
6	0000000000000011		原始数据b=3
7	0000000000000001		原始数据c=1
8	0000000000000000		原始数据y=0

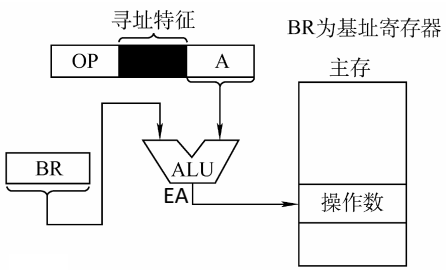
采用基址寻址无需修改指令中的地址码

变量a的实际存放地址为105

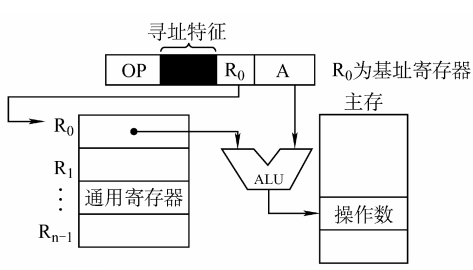
王道考研/CSKAOYAN.COM

### 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）。

王道考研/CSKAOYAN.COM

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。

寻址特征

OP A

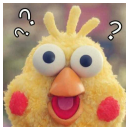
IX

ALU

EA

主存

操作数



不一样？不一样？不一样？  
我们不一样？

寻址特征

OP A

BR

ALU

EA

主存

操作数

BR为基址寄存器

(a) 采用专用寄存器BR作为基址寄存器

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

王道考研/CSKAOYAN.COM

9

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

是时候召唤“变址寻址”了！



如果您觉得太麻烦的话

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

王道考研/CSKAOYAN.COM

10

王道考 研/CSKAOYAN.COM

5

### 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**。

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

王道考研/CSKAOYAN.COM

11

### 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。

寻址特征

OP

A

IX

ALU

EA

主存

操作数

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**。

王道考研/CSKAOYAN.COM

12

基址寻址:  $EA=(BR)+A$   
变址寻址:  $EA=(IX)+A$   
先基址后变址寻址:  $EA=(IX)+(BR)+A$

低地址

高地址

主存

BR 100 IX 2

访问目标

### 基址&变址复合寻址

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)	$(ACC)+(7+(IX)) \rightarrow ACC$
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

王道考研/CSKAOYAN.COM

13

### 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

寻址特征

OP A

PC 1002 or 1004 (PC取值后进行了自增运算, 自增长度与当前指令长度有关)

主存

1000

OP A

操作数

相对距离A

ALU

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

王道考研/CSKAOYAN.COM

14

### 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

王道考研/CSKAOYAN.COM

15

### 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	...		...
...	...		...

王道考研/CSKAOYAN.COM

16



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

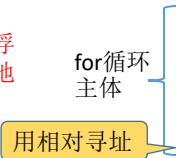
相对寻址： $EA=(PC)+A$ ，其中A是相对于PC所指地址的位移量，可正可负，补码表示

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

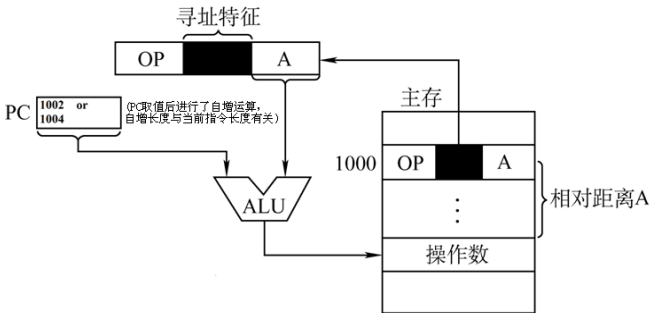
主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数0 → ACC
1	取数到IX	#0	立即数0 → IX
2	...	...	其他代码
3	...	...	其他代码
4	...	...	其他代码
5	...	...	其他代码
...	...	...	其他代码
M	ACC加法	7 (数组始址)	$(ACC)+(7+(IX)) \rightarrow ACC$
M+1	IX加法	#1	$(IX)+1 \rightarrow IX$
M+2	IX比较	#10	比较10-(IX)
M+3	条件跳转	-4(补码表示)	若结果>0 则PC跳转到M
M+4	...	...	...
...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

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

王道考研/CSKAOYAN.COM

19

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

高级语言视角：

```
if (a>b){
    ...
} else {
    ...
}
```

汇编语言中，条件跳转指令有很多种，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

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

王道考研/CSKAOYAN.COM

20