本节内容

数据寻址2

（偏移寻址）

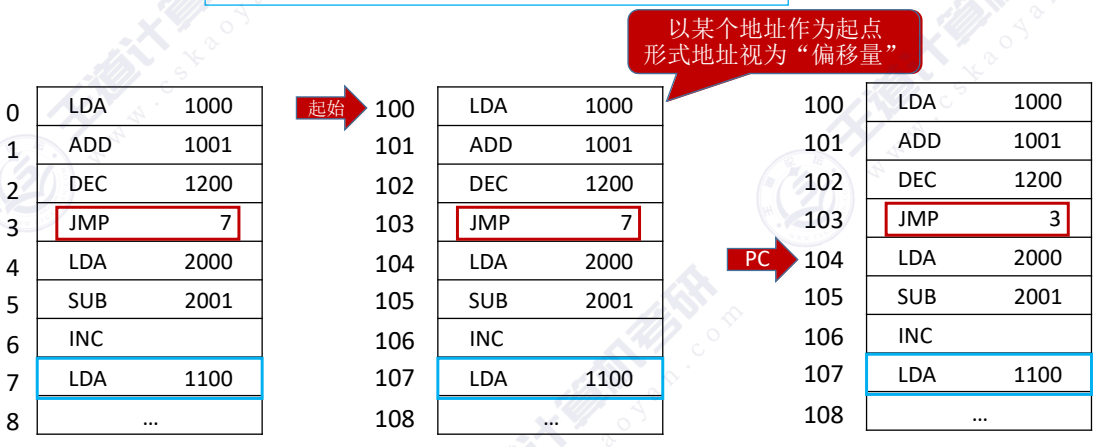
王道考研/CSKAOYAN.COM

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
    - 跳跃寻址
- 数据寻址：确定 本条指令 的 地址码指明的真实地址



王道考研/CSKAOYAN.COM

2

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

- 0110 相对寻址
- 0111 基址寻址
- 1000 变址寻址
- 1001 堆栈寻址

数据寻址

- 0000 隐含寻址
- 0001 立即寻址
- 0010 直接寻址
- 0011 间接寻址
- 0100 寄存器寻址
- 0101 寄存器间接寻址

王道考研/CSKAOYAN.COM

3

偏移寻址

偏移寻址
- 基址寻址 EA=(BR)+A
- 变址寻址 EA=(IX)+A
- 相对寻址 EA=(PC)+A

区别在于偏移的“起点”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”

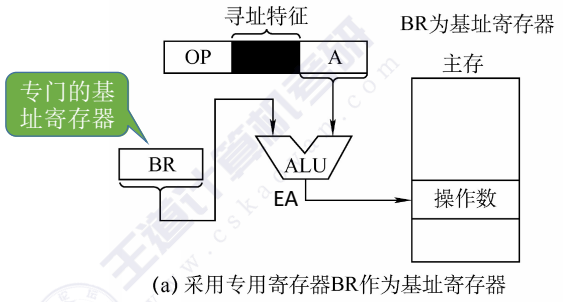
王道考研/CSKAOYAN.COM

4

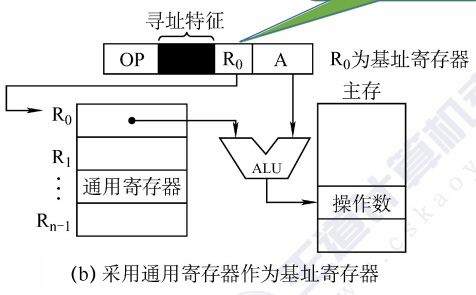
基址寻址

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。



(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

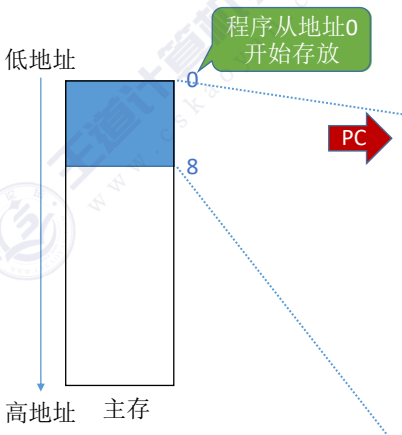
Tips：可对比操作系统第三章第一节学习，OS课中的“重定位寄存器”就是“基址寄存器”

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;
}
```



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

PC

变量a的实际存放地址为5

6

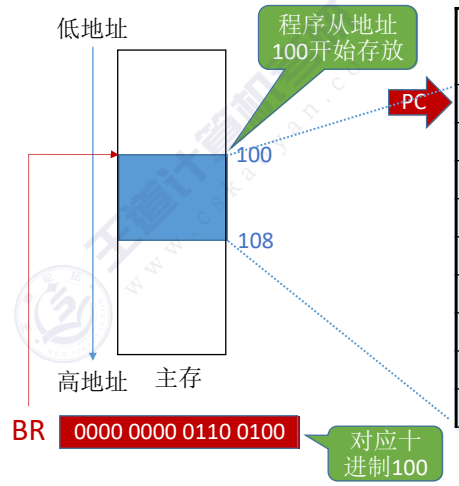
拓展：程序运行前，CPU将BR的值修改为该程序的起始地址（存在操作系统PCB中）

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



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*

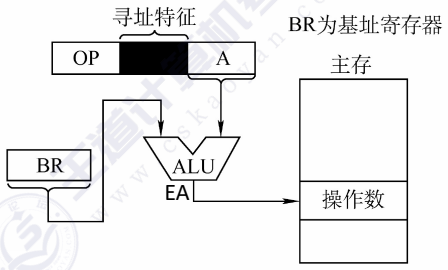
采用基址寻址无需修改指令中的地址码

变量a的实际存放地址为105

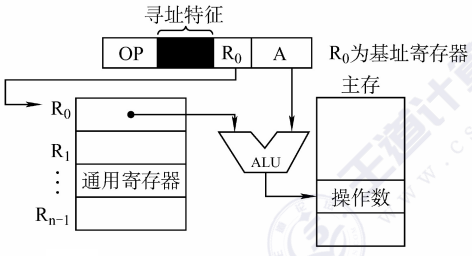
7

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**。

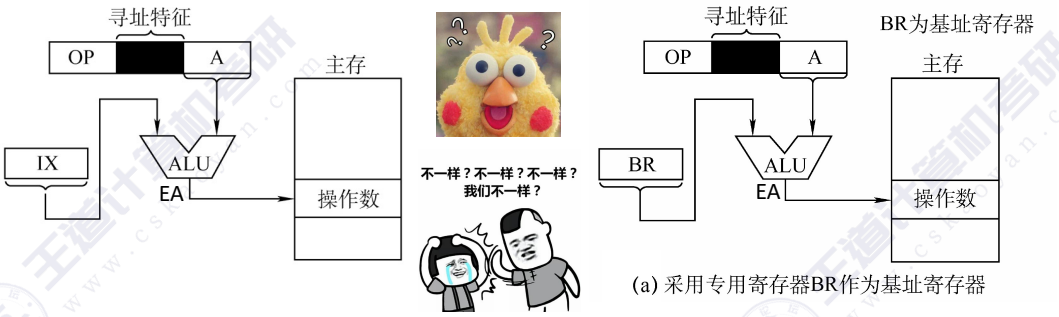
优点：可扩大寻址范围（基址寄存器的位数大于形式地址A的位数）；用户不必考虑自己的程序存于主存的哪一空间区域，故**有利于多道程序设计**，以及可用于编制浮动程序（整个程序在内存里边的浮动）。

8

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。



(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

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];
}
```

ACC 0

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）	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！

10

变址寻址的作用

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

ACC 0

IX 10

	主存地址	指令 操作码	指令 地址码	注释
立即寻址	0	取数到ACC	#0	立即数 0 → ACC
	1	取数到IX	#0	立即数 0 → IX
变址寻址	2	ACC加法	7（数组始址）	(ACC)+(7+(IX))→ ACC
立即寻址	3	IX加法	#1	(IX) + 1→ IX
	4	IX比较	#10	比较10-(IX)
直接寻址	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*

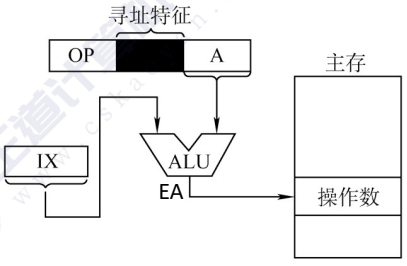
留个坑

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**。

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。



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**。

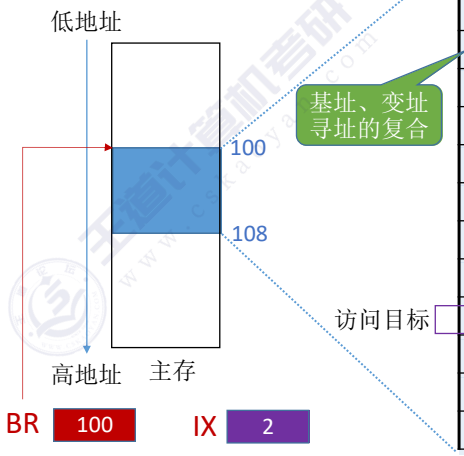
12

基址&变址复合寻址

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

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）



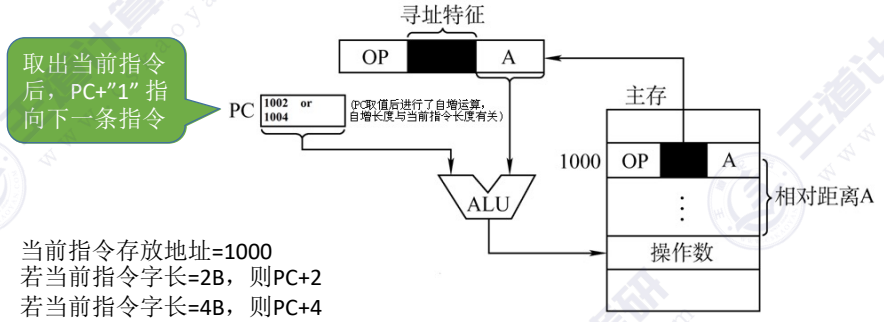
主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数 0 → ACC
1	取数到IX	#0	立即数 0 → IX
2	ACC加法	7（数组始址）	(ACC)+(7+(IX))→ ACC
3	IX加法	#1	(IX) + 1→ IX
4	IX比较	#10	比较10-(IX)
5	条件跳转	2	若结果>0 则PC跳转到2
6	从ACC存数	17	(ACC)→ sum变量
7	*随便什么值*		a[0]
8	*随便什么值*		a[1]
9	*随便什么值*		a[2]
…	…		…
16	*随便什么值*		a[9]
17	*初始为0*		*sum变量*

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” ✗



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

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*

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	…		…
…	…		…

16

相对寻址的作用

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

PC　M+4

用相对寻址

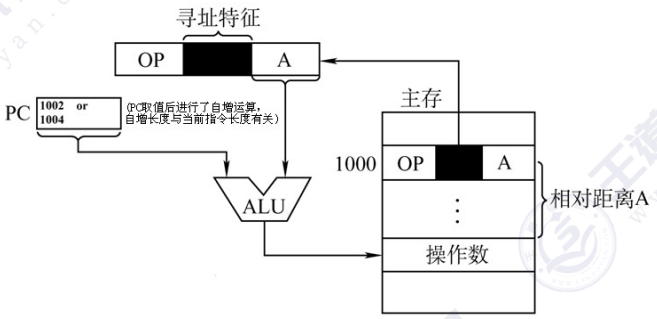
for循环主体

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	-4(补码表示)	若结果>0 则PC跳转到M
M+4	…		…
…	…		…

17

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。

18

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

19

硬件如何实现数的“比较”

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

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*

20

21