第4章 寻址方式

- 茫茫内存，何处寻觅操作数?
- CPU如何知道操作数的地址?
- 日常生活中，有哪些给出地址的方式?
- C程序中，有哪些给出地址的方式?

第4章 寻址方式


```
int   i, j;
int   a[10];
int   *p;
int   *q[10];
int   b[20][10];
```

```
struct point {
    int px;
    int py;
};
```

```
point  original;
```

```
point  lists[10];
```

思考：如何得到要访问单元的地址？

```
a[i] =5;
b[i][j] =10;
original.px=12;
lists[5].px=15;
```

第4章　寻址方式

一、本章的学习内容

立即寻址

寄存器寻址

直接寻址

寄存器间接寻址

变址寻址

基址加变址寻址

寻址方式的综合举例

x86机器指令编码规则

4.1 寻址方式概述

对一条指令，关注的焦点有哪些？

- 执行什么操作？
- 操作数在哪里？

操作数的存放位置即存放地址
- CPU内的寄存器
- 主存
- I/O设备端口

操作数在主存时：关注段址/段选择符、段内偏移

- 操作数的类型　字节/字/双字？

寻找操作数存放地址的方式称为寻址方式。

4.1 寻址方式概述

双操作数的指令格式　　操作符 OPD, OPS

Intel格式

ADD　EAX，EBX

目的操作数地址　　源操作数地址

(OPD) + (OPS) → OPD

(EAX) + (EBX) → EAX

Question：操作结束后，运算结果保存在哪？

源操作数是否变化？

4.2 立即寻址

- 操作数直接放在指令中，在指令的操作码后；
- 操作数是指令的一部分，位于代码段中；
- 指令中的操作数是8位、16位或32位二进制数。

使用格式： n

操作码及目的操作数寻址方式码
立即操作数 n

立即操作数只能作为源操作数。

例： MOV EAX , 12H

机器码： B8 12 00 00 00

4.2 立即寻址


```
int   global;   // 注：global 是全局变量
global = 2 * 5 ;
00521A17      C7 05 B8 A5 52 00 0A 00 00 00
              mov  dword ptr [global (052A5B8h)], 0Ah
```

4.3 寄存器寻址

使用格式： R

功　　能： 寄存器R中的内容即为操作数。

说　　明： 除个别指令外，R可为任意寄存器。

例1：DEC　BL

执行前 ： (BL) = 43H

执　行： (BL) –1 = 43 H –1 = 42H → BL

执行后： (BL)=42H

4.4 直接寻址

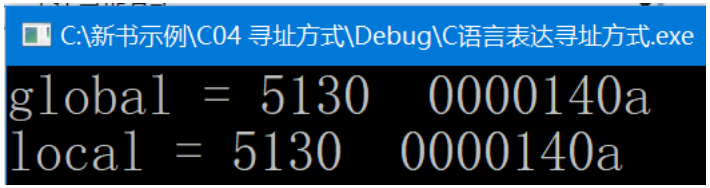
- 操作数在内存中；
- 操作数的偏移地址EA紧跟在指令操作码后面。

格式：[地址偏移量]

4.4 直接寻址


```
int global;
void address_compare()
{
    int local;
    global = 2*5;
    *((char *)&global + 1) = 20;
    printf("global = %d  %08x\n", global, global);

    local = 10;
    *((char*)&local + 1) = 20;
    printf("local = %d  %08x\n", local, local);
}
```



虽然全局/局部变量的访问形式相同，
但是它们对应的寻址方式是不同的。

4.4 直接寻址

- 操作数在内存中；
- 操作数的偏移地址EA紧跟在指令操作码后面。

全局变量的访问：直接寻址

```
    global = 2*5;
C7 05 44 A1 E9 00 0A 00 00 00 mov         dword ptr ds:[00E9A144h],0Ah
    *((char*)&global + 1) = 20;
C6 05 45 A1 E9 00 14 mov         byte ptr ds:[00E9A145h],14h
```

局部变量的访问：变址寻址

```
    local = 10;
C7 45 F4 0A 00 00 00 mov         dword ptr [ebp-0Ch],0Ah
    *((char*)&local + 1) = 20;
C6 45 F5 14          mov         byte ptr [ebp-0Bh],14h
```

4.4 直接寻址

Q ： 下面三种写法各自的含义是什么？

```
*((char*)&global + 1) = 20;
*((char*)(&global + 1)) = 20;
*(int *)((char*)&global + 1) = 20;
```

注意：C语言的写法 与 机器语言程序的差异！

```
    global = 2*5;
C7 05 CC A5 99 00 0A 00 00 00 mov         dword ptr ds:[0099A5CCh],0Ah
    *((char*)&global + 1) = 20;
C6 05 CD A5 99 00 14 mov         byte ptr ds:[0099A5CDh],14h
    *((char*)(&global + 1)) = 20;
C6 05 D0 A5 99 00 14 mov         byte ptr ds:[0099A5D0h],14h
    *(int *)((char*)&global + 1) = 20;
C7 05 CD A5 99 00 14 00 00 00 mov         dword ptr ds:[0099A5CDh],14h
```

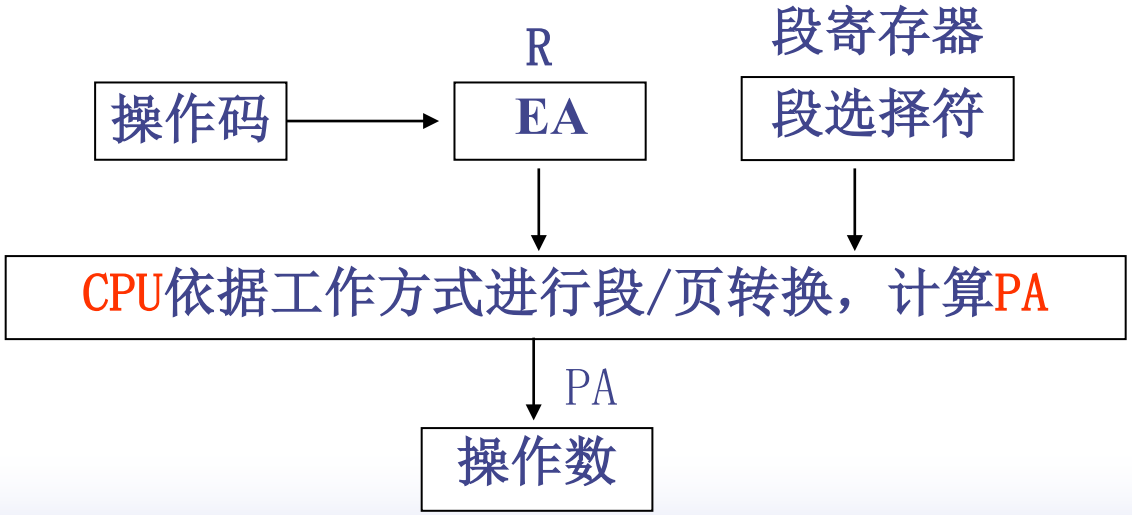
地址差异、机器指令中的类型 ！

4.5 寄存器间接寻址

格式：[R]

功能：操作数在内存中，操作数的偏移地址在寄存器R中。即（R)为操作数的偏移地址.

例如：MOV　AX, [ESI]



寄存器间接寻址方式的寻址过程

4.5 寄存器间接寻址

- R 可以是:
  8个32位通用寄存器中的任意一个
  EAX, EBX, ECX, EDX,ESI,EDI,ESP,EBP

- 操作数的偏移地址在指令指明的寄存器中

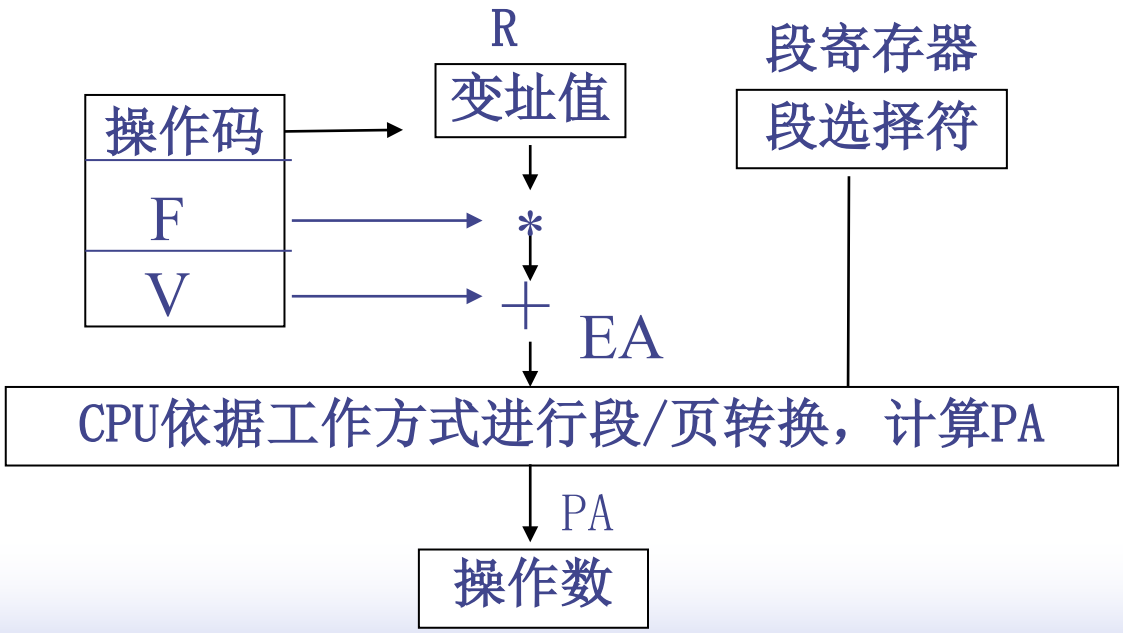
- 操作数所在的段是?
  扁平内存管理模式下，（DS）=（SS）

4.6 变址寻址

格式：V[R×F]

功能：R中的内容×F + V　为操作数的偏移地址。

例如：MOV　AL, 5[EBX*2]



4.6 变址寻址

- R 可以是:

  8个32位通用寄存器

  EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP

- F 可为 1，2，4，8

4.6 变址寻址

華中科技大學

```
19  short gsa[5];
20  int main(int argc, char* argv[])
21  {
22      short lsa[5];
23      int i = 2;
24      gsa[1] = 20;
25      lsa[1] = 20;
26      gsa[i] = 30;
27      lsa[i] = 30;
```

```
    gsa[1] = 20;
007D1A19 B8 02 00 00 00       mov         eax,2
007D1A1E C1 E0 00             shl         eax,0
007D1A21 B9 14 00 00 00       mov         ecx,14h
007D1A26 66 89 88 C0 A5 7D 00 mov         word ptr [eax+007DA5C0h],cx
```

全局变量 编译后对应一个数值化的地址

4.6 变址寻址

全局变量数组访问：比较常量下标与变量下标的差异

```
    gsa[1] = 20;
007D1A19 B8 02 00 00 00       mov         eax,2
007D1A1E C1 E0 00             shl         eax,0
007D1A21 B9 14 00 00 00       mov         ecx,14h
007D1A26 66 89 88 C0 A5 7D 00 mov         word ptr [eax+007DA5C0h],cx
```

```
    gsa[i] = 30;
007D1A3F B8 1E 00 00 00       mov         eax,1Eh
007D1A44 8B 4D E0             mov         ecx,dword ptr [ebp-20h]
007D1A47 66 89 04 4D C0 A5 7D 00 mov         word ptr [ecx*2+007DA5C0h],ax
```

全局变量 编译后对应一个数值化的地址

4.6 变址寻址

局部变量数组访问：比较常量下标与变量下标的差异

```
    lsa[1] = 20;
007D1A2D B8 02 00 00 00       mov         eax,2
007D1A32 C1 E0 00             shl         eax,0
007D1A35 B9 14 00 00 00       mov         ecx,14h
007D1A3A 66 89 4C 05 EC       mov         word ptr [ebp+eax-14h],cx

    lsa[i] = 30;
007D1A4F B8 1E 00 00 00       mov         eax,1Eh
007D1A54 8B 4D E0             mov         ecx,dword ptr [ebp-20h]
007D1A57 66 89 44 4D EC       mov         word ptr [ebp+ecx*2-14h],ax
```

局部变量 编译后对应一个地址为 [ebp - n]

4.7 基址加变址寻址

格式： [BR＋IR×F+V]

或 V[BR][IR×F] 或 V [IR×F][BR]

或 V[BR+IR×F]

功 能：操作数的偏移 ＝变址寄存器IR中的内容×比例因子F +位移量V+基址寄存器BR中的内容 。

EA = (IR)*F + V + (BR)

例如：MOV EAX, -6[EDI*2][EBP]

4.7 基址加变址寻址

◆F 可为 1，2，4，8

◆当使用32位寄存器时

BR可以是 EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP 之一；

IR 可以是除ESP外的任一32位寄存器；

未带比例因子的寄存器是 BR；

当没有比例因子时，写在前面的寄存器是BR.

4.7 基址加变址寻址

特别说明：

当V中存在全局变量或标号时，用的段都是 DS。

在 VS2019中， (DS)=(SS)。

CS中的全局变量等同 DS段中的变量。

- 操作数的类型：

  若V为变量，则操作数类型为变量的类型；

  若V为常量，类型未知。

4.8 寻址方式综合举例

寻址方式有6种。
根据操作数的存放位置，寻址方式归为3类：

立即方式
寄存器方式
存储器方式
- 寄存器间接寻址
- 变址寻址
- 基址加变址寻址
- 直接寻址

4.8 寻址方式综合举例

思考题

从机器语言的角度，有六种寻址方式。

C语言程序是要编译成机器语言程序的。

C程序中存储单元的各种访问方式，

分别会翻译成怎样的寻址方式呢?

4.9 x86机器指令编码规则

从机器语言的角度，看待指令的组成部分；

判断写的汇编语句是否正确，可以看它能否编译成机器指令。

即各部分能否转换成指令中组成的要素。

- 计算机 是 0-1世界
- 编码与解码　　汇编与反汇编
- 指令组成的核心

  操作与操作数地址

4.9 x86机器指令编码规则

指令前缀	操作码	内存/寄存器操作数	索引寻址描述	地址偏移量	立即数

① 指令前缀(prefix，非必需，0个或多个字节)

② 操作码(opcode，必须，1字节～3字节)

③ 内存/寄存器操作数(ModR/M，非必需)

指明寻址方式

④ 索引寻址描述 (SIB，非必需)

指明：基址寄存器、变址寄存器、比例因子

⑤ 地址偏移量(Displacement，非必需)

⑥ 立即数(Immediate，非必需)

4.9 x86机器指令编码规则

指令前缀

种类	名称	二进制码	说明
LOCK	LOCK	F0H	让指令在执行时候先禁用数据线的复用特性，用在多核的处理器上，一般很少需要手动指定
REP	REPNE/REPNZ	F2H	用 CX(16 位下)或 ECX(32 位下)或 RCX(64 位下)作为指令是否重复执行的依据
	REP/REPE/REPZ	F3H	同上
Segment Override	CS	2EH	段重载(默认数据使用 DS 段)
	SS	36H	同上
	DS	3EH	同上
	ES	26H	同上
	FS	64H	同上
	GS	65H	同上
REX	64 位	40H-4FH	x86-64 位的指令前缀，见第 18 中的介绍
Operand size Override	Operand size Override	66H	用该前缀来区分：访问 32 位或 16 位操作数；也用来区分 128 位和 64 位操作数
Address Override	Address Override	67H	64 位下指定用 64 位还是 32 位寄存器作为索引

4.9 x86机器指令编码规则

操作码

- 指明了要进行的操作
- 指明操作数的类型（一般看最后一个二进制）

  0：字节操作； 1：字操作（32位指令中为双字操作）；

  有指令前缀 66H时，对字操作。
- 指明源操作数是寄存器寻址，还是目的操作数是寄存器寻址 （操作码的倒数第二个二进制位）

  1：目的操作数是寄存器寻址，0：源操作数寄存器寻址

  与[寻址方式字节]配合使用
- 在有些指令中（如立即数传送给寄存器），操作码含有寄存器的编码。
- 一般在opcode的编码中体现了源操作数是否为立即数。

4.9 x86机器指令编码规则

内存/寄存器操作数(ModR/M)

Mod （6-7 位）	Reg/Opcode（3-5 位）	R/M （0-2 位）

- Mod由2个二进制位组成，取值是00、01、10、11。
- Mod与为R/M配合使用，明确一个操作的获取方法。
- Reg/Opcode 确定另外一个寄存器寻址的寄存器编码

4.9 x86机器指令编码规则

内存/寄存器操作数(ModR/M)

Mod （6-7 位）	Reg/Opcode(3-5 位)	R/M （0-2 位）

R/M	Mod
[EAX], [ECX], [EDX], [EBX], [--][--], disp32, [ESI], [EDI]	00
[EAX]+disp8, [ECX]+disp8, [EDX]+disp8, [EBX]+disp8, [--][--]+disp8, [EBP]+disp8, [ESI]+disp8, [EDI]+disp8	01
[EAX]+disp32, [ECX]+disp32, [EDX]+disp32, [EBX]+disp32, [--][--]+disp32, [EBP]+disp32, [ESI]+disp32, [EDI]+disp32	10
EAX/AX/AL/MM0/XMM0, ECX/CX/CL/MM1/XMM1, EDX/DX/DL/MM2/XMM2, EBX/BX/BL/MM3/XMM3, ESP/SP/AH/MM4/XMM4, EBP/BP/CH/MM5/XMM5, ESI/SI/DH/MM6/XMM6, EDI/DI/BH/MM7/XMM7	11

Mod=00, R/M =000，表示 用 [EAX] 寻址

Mod=00，R/M=100, 表示用[--][--] ，无位移量的基址加变址

在 SIB 字节指明基址/变址寄存器的编码

4.9 x86机器指令编码规则

内存/寄存器操作数(ModR/M)

Mod （6-7 位）	Reg/Opcode（3-5 位）	R/M （0-2 位）

000	001	010	011	100	101	110	111
AL	CL	DL	BL	AH	CH	DH	BH
AX	CX	DX	BX	SP	BP	SI	DI
EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
MMO	MM1	MM2	MM3	MM4	MM5	MM6	MM7
XMMO	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7

Reg/Opcode 的编码

- 同一编码有多个寄存器
- 用哪一个寄存器，取决于指令前缀和操作码中的编码

mov eax, [ebx] ；机器码是：8B 03 =》 00 000 011 [EBX]

mov [ebx], eax ；机器码是：89 03 =》 1000 1001 VS 1000 1011

操作码的倒数第二位： 1：OPD是寄存器， 0：OPS是寄存器

4.9 x86机器指令编码规则

内存/寄存器操作数(ModR/M)

Mod （6-7 位）	Reg/Opcode（3-5 位）	R/M （0-2 位）

000	001	010	011	100	101	110	111
AL	CL	DL	BL	AH	CH	DH	BH
AX	CX	DX	BX	SP	BP	SI	DI
EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
MMO	MM1	MM2	MM3	MM4	MM5	MM6	MM7
XMMO	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7

mov eax, [ebx] ；机器码是：8B 03 =》 00 000 011 [EBX]

mov [ebx], eax ；机器码是：89 03 =》 1000 1001 VS 1000 1011

mov ax, [ebx] ；机器码是：66 8B 03 指令前缀，字类型操作

mov al, [ebx] ；机器码是：8A 03 => 1000 1010

32位指令 op码的最后一位为 0，字节操作；为1；双字操作

有前缀 66，为 1：字操作

4.9 x86机器指令编码规则

索引寻址描述 SIB

Scale （6-7 位）	Index(3-5 位)	Base （0-2 位）

000	001	010	011	100	101	110	111
AL	CL	DL	BL	AH	CH	DH	BH
AX	CX	DX	BX	SP	BP	SI	DI
EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7

- 与内存/寄存器操作数(ModR/M) 配合使用
- 在32位指令系统中，基址寄存器与变址寄存器都是 32 位的
- Base =000，表示基址寄存器为 EAX

```
mov   eax,[ebx+ecx*4]      ; 机器码是 8B 04 8B
 10 001 011 ：比例因子10（4），变址寄存器 001 ，即ECX
                基址寄存器 011，即 EBX
04 ： 00 000 100 =》  00 /100 对应的就是 [-][-]
```

4.9 x86机器指令编码规则

地址偏移量(Displacement)

- 由1、2 或 4 个字节组成，分别对应8位、16位或32位的偏移量;
- 数据按照小端顺序存放，即数据的低位存放在小地址单元中。

立即数(Immediate)

- 对应立即寻址方式
- 占1、2 或 4个字节, 按照小端顺序存放。

test.cpp

```
#include <iostream>
int  g;
int  a[5];
int  b[3][5];
char *p = (char*)malloc(10);
int main()
{
    int  i, j, k;
    g = 10;
    *((char*)&g + 1) = 20;
    printf("%d %x \n", g, g);
    a[0] = 20;
    a[1] = 25;
    i = 2;
    a[i] = 10;
    *(a + 3) = 17;
    j = 3;
    b[i][j] = 12;
    k = b[i][j];
    *p = 'a';
    *(p + 1) = 'b';
    *(p + 2) = 0;
    return 0;
}
```

反汇编，强化六种寻址方式的用法

C中变量访问与 机器指令寻址方式