13015 计算机系统原理

第三章考点解析（1）

13015 计算机系统原理【第三章】

重点强调

理解我讲解的考点内容即可，不要去深究，拿分是关键

理解我讲解的考点内容即可，不要去深究，拿分是关键

理解我讲解的考点内容即可，不要去深究，拿分是关键

13015 计算机系统原理【第三章】

考点1 机器指令及汇编指令

指令包括操作码、地址码等字段

R型：寄存器　　op为0000（传送mov）、0001（加add）操作

M型：主存单元　　op为1110（取数load）、1111（存数store）操作

例如：**机器指令**“**11100110**”对应的**汇编指令**为“**load r0,6#**”。

用若干个**助记符**表示的与机器指令一一对应的指令称为**汇编指令**

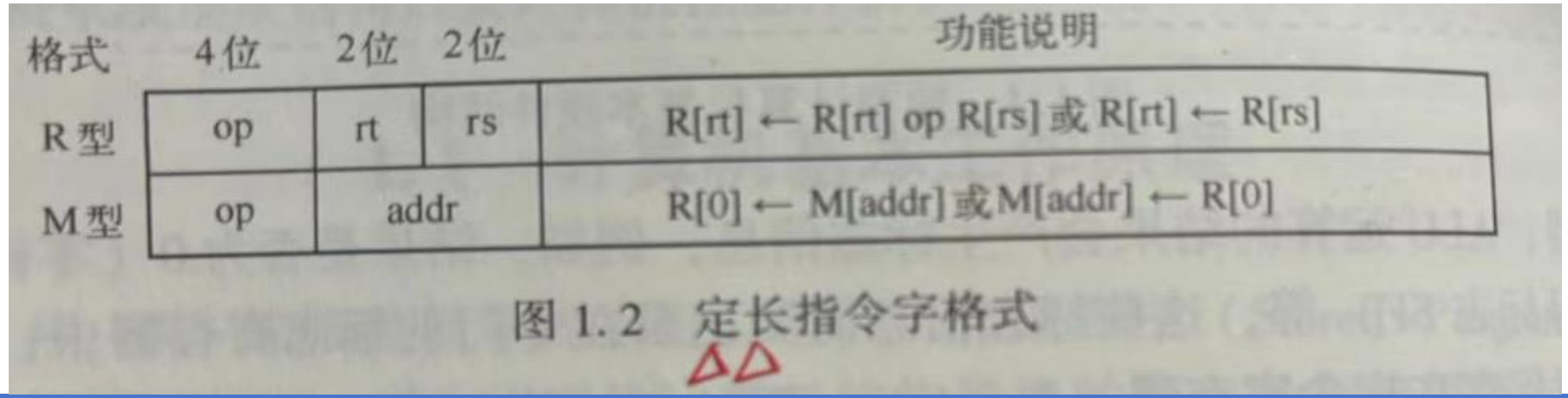


图 1.2　定长指令字格式

13015 计算机系统原理【第三章】

考点2 寄存器传送语言(Register Transfer Language, RTL)

本章中多处需要对指令功能进行描述，为简化对指令功能的说明，将采用**寄存器传送语言(RTL)** 来说明。

R[r]：表示寄存器r的内容

M[addr]：表示存储单元addr的内容

M[R[r]]：表示寄存器r的内容所指的存储单元的内容

传送方向用 ← 表示，右：传送源，左：传送目的

例如：对于**AT&T**格式指令“movw **4(%ebp)**, **%ax**”,其功能为

R[ax]←M[R[ebp]+4],

含义是：将寄存器ebp的内容和4相加得到的地址对应的内容送到寄存器ax中。

13015 计算机系统原理【第三章】

考点3 指令系统设计风格

现代计算机都采用**通用寄存器型**指令系统，使用**通用寄存器**来存放运算过程中所用的临时数据。指令的操作数有以下三种：

立即数（Immediate，简称I）：通常用于表示一个固定的数值，这个数值可以直接出现在指令中，而不需要从内存或其他寄存器中读取。

例如：movl $80, -8(%ebp) # M[R[ebp]-8]← 80

寄存器（Register，简称R）：是CPU内部的一种存储单元，用于存储数据和地址。寄存器可以直接被CPU访问，因此它们是执行指令时非常快速的数据存储方式。在指令中，寄存器通过其名称或编号来引用，以便于快速访问和操作数据。**例如：movl $80, -8(%ebp) # M[R[ebp]-8]← 80**

13015 计算机系统原理【第三章】

考点3 指令系统设计风格

存储单元（Memory，简称S）：是计算机内存中的一种数据存储方式，可以存储程序代码、数据和其他信息。在指令中，存储单元通过**地址**来引用。存储单元用于数据的长期存储和读取，尽管访问速度较寄存器慢，但它们提供了大量的数据存储空间。

例如：movl $80, -8(%ebp) # M[R[ebp]-8]← 80

两种类型的指令系统（名词解释题）

CISC：复杂指令集计算机

RISC：精简指令集计算机

13015 计算机系统原理【第三章】

考点4 Intel和AT&T格式指令区别

	Intel格式指令	AT&T格式指令（本教材使用）
指令	[ebp+edx*4+8]	8(%ebp, %edx, 4)
形式	[基址寄存器+变址寄存器*比例因子+偏移量]	偏移量(基址寄存器，变址寄存器，比例因子)
区别	不带%	带%
	内存用[]	内存用()
	左：目的，右：源 eg:mov ax [ebp+edx*4+8]	左：源，右：目的 eg:movw 8(%ebp,%edx,4), %ax
RTL	R[ax]←M[R[ebp]+R[edx]×4+8]	
含义	将寄存器ebp的内容＋4倍的寄存器edx的内容＋8得到的地址对应的内容送到寄存器ax中	

汇编格式指令为：“op src,dst”，含义为“dst←dst op src”。

如“addl (,%ebx,2),%eax”的含义为“R[eax]←R[eax]+M[R[ebx]×2]”

13015 计算机系统原理【第三章】

考点5 数据类型及格式

现在Intel把32位**x86架构**的名称x86-32改称为**IA-32**

	Intel操作数类型	汇编指令长度后缀	存储长度/位
char	字节	b	8
short	字	w	16
int	双字	l	32
long	双字	l	32
char*	双字	l	32

GCC生成的汇编代码中的指令助记符大部分都有**长度**后缀

例如：movb(字节传送)、movw(字传送)、movl(双字传送)

注意：汇编指令长度后缀的b是字节，不是比特

13015 计算机系统原理【第三章】

考点6 定点寄存器组

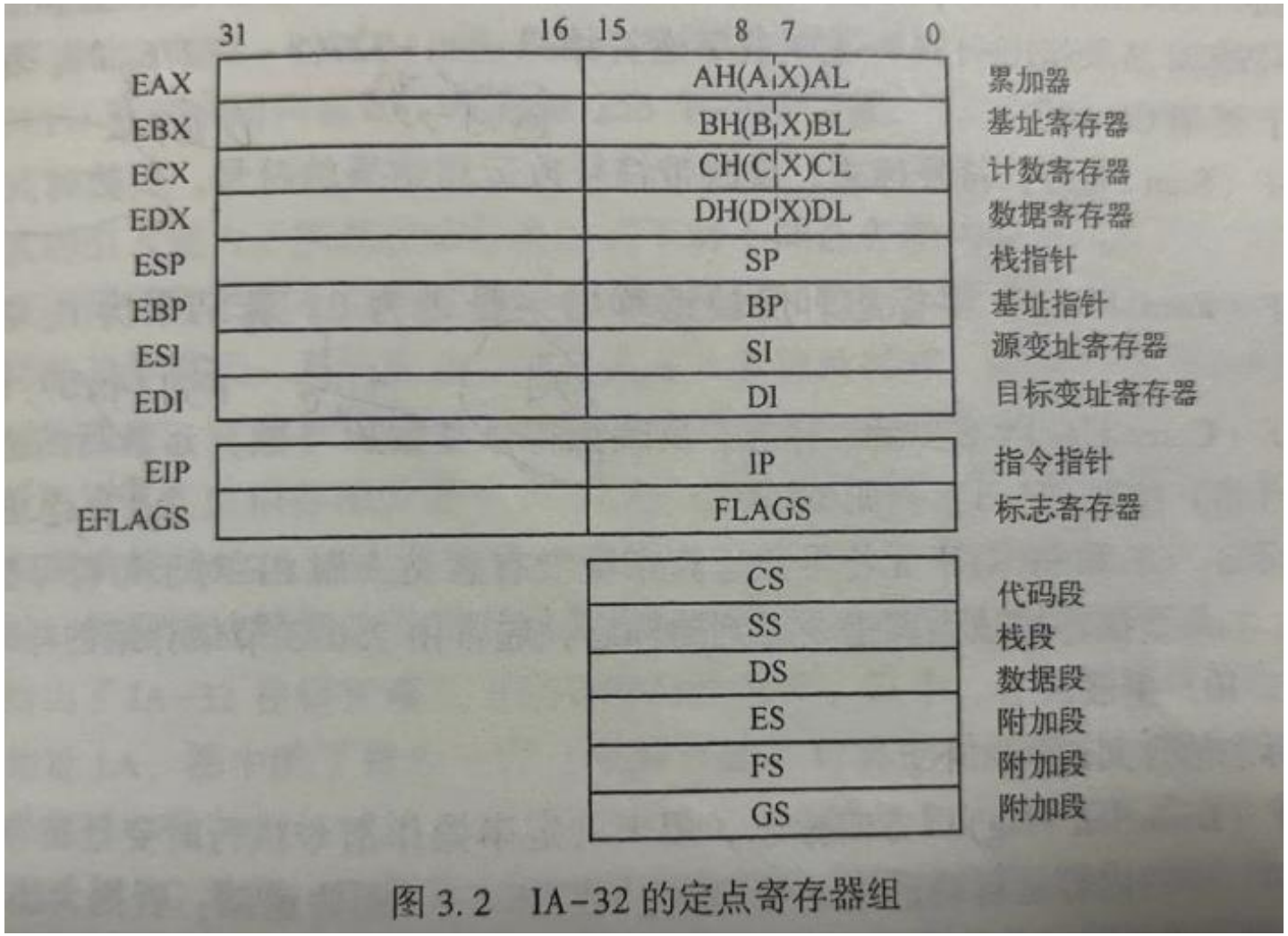


图 3.2 IA-32 的定点寄存器组

用“EAX”举例说明长度

AL → 8位（低Low）

AH → 8位（高High）

AX → 16位

EAX → 32位

EAX是“Extended Accumulator Register”的缩写

意为：扩展的累加器寄存器

助记：2字母16位，3字母32位，看见末尾L、H是8位

13015 计算机系统原理【第三章】

考点6 定点寄存器组

8个通用寄存器

EAX、EBX、ECX和EDX主要用来存放操作数

ESP、EBP、ESI和EDI主要用来存放变址值或指针

ESP是栈指针寄存器，EBP是基址指针寄存器

2个专用寄存器

EIP是指令指针寄存器，EFLAGS是标志寄存器

6个段寄存器

13015 计算机系统原理【第三章】

考点7 条件标示和控制标示

条件标志

OF(Overflow Flag)：溢出标示，主要发生在**有符号数**的运算中，当运算结果超出了该数据类型所能表示的数值范围时，就发生了溢出。

例如：对于8位有符号整数，其表示范围是 -128 ~ 127，如果计算结果超出了这个范围，就发生了溢出。

CF(Carry Flag)：进/借位标示，通常发生在**无符号数**的加法运算中。当两个数相加，结果的某一位产生了向更高位的进位时，就称为产生了进位。

例如：在二进制加法中，两个 1 相加，结果为 0 并向更高位进位 1。

13015 计算机系统原理【第三章】

考点7 条件标示和控制标示

条件标志

SF(Sign Flag)：符号标示，**有符号数**运算结果的符号。

负数时，SF=1；否则SF=0;　　**助记：0正1负**

ZF(Zero Flag)：零标示，运算结果是否为0。若为0，ZF=1;否则ZF=0;

控制标志

DF：方向标志，DF=1↓；否则↑

IF：中断允许标志，IF=1，表示允许响应中断；否则禁止

TF：陷阱标志，TF=1,CPU按单步方式执行指令

13015 计算机系统原理【第三章】

考点7 条件标示和控制标示

例 3.4 假设 R[ax] = FFFAH，R[bx] = FFF0H，则执行 Intel 格式指令“add ax，bx”后，AX、BX 中的内容各是什么？标志 CF、OF、ZF、SF 各是什么？要求分别将操作数作为无符号整数和带符号整数来解释并验证指令执行结果。

无符号

Intel格式指令“add ax,bx” ==> R[ax]←R[ax]+R[bx]

R[ax] = FFFAH = 65530，R[bx] = FFF0H = 65520，相加：65530 + 65520 = 131050

R[ax] = FFFAH + FFF0H = 1FFEAH，R[bx] = FFF0H

重点：由于无符号整数是 16 位，所以结果超出 16 位的部分舍去，结果为 FFEAH 存放在 AX 中

CF=1（无符号进位1），OF=0（无符号无意义），SF=0（无符号无意义），ZF=0（结果不为0）

重点关注：**教材有点模棱两可，没说到底是无符号还是有符号，应该分开解释**

13015 计算机系统原理【第三章】

考点7 条件标示和控制标示

例 3.4 假设 R[ax] = FFFAH，R[bx] = FFF0H，则执行 Intel 格式指令“add ax，bx”后，AX、BX 中的内容各是什么？标志 CF、OF、ZF、SF 各是什么？要求分别将操作数作为无符号整数和带符号整数来解释并验证指令执行结果。

有符号

Intel格式指令“add ax,bx” ==> R[ax]←R[ax]+R[bx]

R[ax] = FFFAH = -6，R[bx] = -16，相加：-6 + -16 = -22

R[ax] = FFFAH + FFF0H = FFEAH，R[bx] = FFF0H

FFFAH 对应的数是 -6，其原码：符号位-数值位，1111 1111 1111 1010，最高位是1（0正1负），当负数时，数值位各位取反，末位加1，即：1000 0000 0000 0101 + 1 = 1000 0000 0000 0110 = -6

CF=0（有符号无意义），OF=0（有符号无溢出），SF=1（有符号最高位），ZF=0（结果不为0）

13015 计算机系统原理【第三章】

考点8 寻址方式

根据指令给定信息得到操作数或操作数地址的方式称为**寻址方式**

立即寻址：指令中直接给出操作数

寄存器寻址：指令中给出操作数所存放的**寄存器**的编号。

其他寻址：操作数都在**存储单元**中，称为存储器操作数

- 实地址模式
- 保护模式

13015 计算机系统原理【第三章】

考点9 通用数据传送指令

MOV：一般的传送指令，包括mov**b**，mov**w**和mov**l**等。

MOVS：符号扩展传送指令，将**短**的**源数据**高位符号扩展后传送到目的地址

例如：movs**bw**：表示把一个字节进行符号扩展后送到一个16位寄存器中。

MOVZ：零扩展传送指令，将**短**的**源数据**高位零扩展后传送到目的地址

例如：movz**wl**：表示把一个字的高位进行零扩展后送到一个32位寄存器中。

请注意：MOVS和MOVZ指令的目的地址只能是寄存器编号。

重点：涉及到扩展，**有符号**的是**符号扩展**，**无符号**的是**零扩展**

XCHG：数据交换指令，将两个寄存器内容互换。

例如：xchgb表示字节交换。

13015 计算机系统原理【第三章】

考点9 通用数据传送指令

PUSH：先执行R[sp]←R[sp]-2或R[esp]←R[esp]-4，然后将一个字或双字从指定寄存器送到SP或ESP指示的栈单元中。

例如：pushl表示双字压栈，pushw表示字压栈

POP：就是PUSH的反操作。

说明：当进行 Push（压栈）操作时，使用 R[esp]（栈指针）减去 4 通常是因为所操作的数据以 4 字节（32 位）为单位。在许多常见的计算机架构中，内存按字节编址，但数据的存储和操作往往以特定的字节长度为单位。对于 32 位的系统，一个数据单元通常是 4 个字节。当进行压栈操作时，**要为新的数据元素预留空间，所以将栈指针 R[esp] 减去 4 个字节，以便能够正确地存储下一个要压入栈的数据。**

13015 计算机系统原理【第三章】

考点9 通用数据传送指令

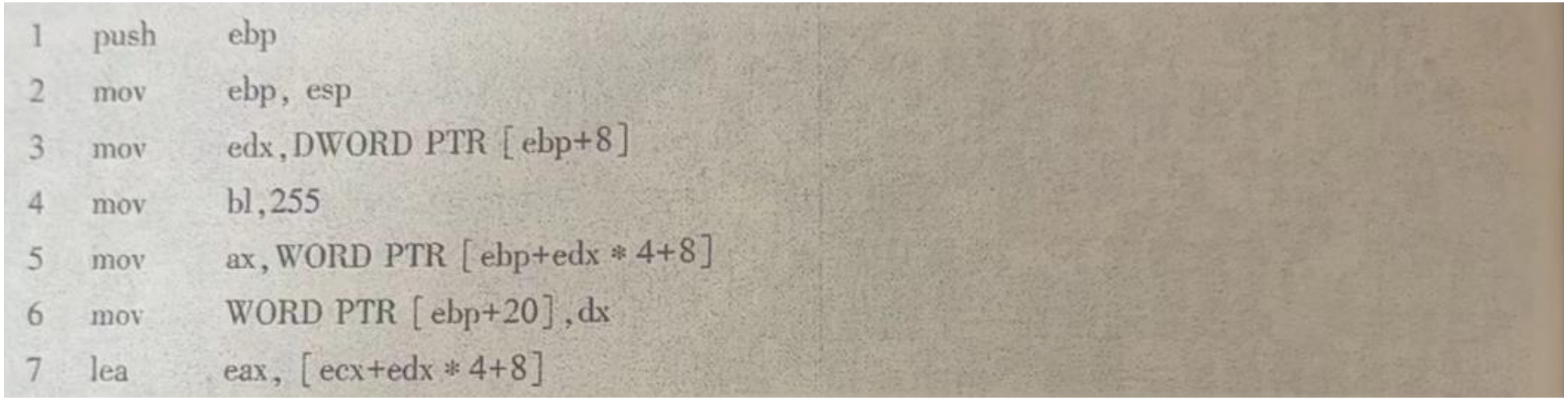
LEA（**L**oad **E**ffect **A**ddress）：加载有效地址

例如：

leal 8(%ecx,%edx,4), %eax #R[eax]←R[ecx]+R[edx]×4+8**传地址**

movl 8(%ecx,%edx,4), %eax #R[eax]←**M[**R[ecx]+R[edx]×4+8**]** **传值**

13015 计算机系统原理【第三章】2404 第三道计算题



```
1   push    ebp
2   mov     ebp, esp
3   mov     edx,DWORD PTR [ebp+8]
4   mov     bl,255
5   mov     ax,WORD PTR [ebp+edx*4+8]
6   mov     WORD PTR [ebp+20],dx
7   lea     eax, [ecx+edx*4+8]
```

```
1 pushl %ebp                  #R[esp]←R[esp]-4,M[R[esp]]←R[ebp]
2 movl %esp,%ebp              #R[ebp]←R[esp]
3 movl 8(%ebp),%edx           #R[edx]←M[R[ebp]+8]
4 movb $255,%bl               #R[bl]←255
5 movw 8(%ebp,%edx,4),%ax     #R[ax]←M[R[ebp]+R[edx]×4+8]
6 movw %dx,20(%ebp)           #M[R[ebp]+20]←R[dx]
7 leal 8(%ecx,%edx,4),%eax    #R[eax]←R[ecx]+R[edx]×4+8
```

说明1

①预留空间

②把基址寄存器压到栈里面

此时栈顶就是基址寄存器

内存操作M[]（栈内存）

13015 计算机系统原理【第三章】

考点9 通用数据传送指令

例 3.3 假设变量 val 和 ptr 的类型声明如下：

```
val_type val;
contofptr_type * ptr;
```

已知上述类型 val_type 和 contofptr_type 是用 typedef 声明的数据类型，且 val 存储在累加器 AL/AX/EAX 中，ptr 存储在 EDX 中。现有以下两条 C 语言语句：

```
1   val=(val_type) * ptr;
2   * ptr=(contofptr_type) val;
```

当 val_type 和 contofptr_type 是表 3.3 中给出的组合类型时，应分别使用什么样的 MOV 指令来实现这两条 C 语句？要求用 GCC 默认的 AT&T 形式写出。

表 3.3 例 3.3 中 val_type 和 contofptr_type 的类型

val_type	contofptr_type	val_type	contofptr_type
char	char	int	unsigned char
int	char	unsigned	unsigned char
unsigned	int	unsigned short	int

val 存储在

AL → 8位

AX → 16位

EAX → 32位

ptr 存储在

EDX → 32位

***（指针）可以看作是取值操作，从主存取值，用M**

13015 计算机系统原理【第三章】

考点9 通用数据传送指令 下图左：内存单元，右：寄存器

```
2    * ptr=(contofptr_type) val;
```

序号	val_type		contofptr_type
1	char	→	char
2	int	→	char
3	unisigned	→	int
4	int	→	unisigned char
5	unisigned	→	unisigned char
6	unisigned short	→	int

语句2对应的指令及操作	
movb %al, (%edx)	#传送
movb %al, (%edx)	#截断,传送
movl %eax, (%edx)	#传送
movb %al, (%edx)	#截断,传送
movb %al, (%edx)	#截断,传送
movzwl %ax, %eax movl %eax, (%edx)	#零扩展 #传送

13015 计算机系统原理【第三章】

考点9 通用数据传送指令 下图左：内存单元，右：寄存器

```
2    * ptr=(contofptr_type) val;
```

ptr：存储在**EDX**，**val**：存储在**AL/AX/EAX**类型

伪代码：EDX（目的）←AL/AX/EAX（源）

将累加器**AL/AX/EAX中的内容**送到**地址为R[EDX]的存储单元**中

1 **相同操作类型**(**即使一个是带符号整数类型，另一个是无符号整数类型**)，用**直接传送的指令**即可

2 int→char，大→小，截断传送，直接丢弃寄存器中高24位，仅将最低8位送到存储单元中，所以用%al

3 同1　　4 同2　　5 同2

6 unisigned short→int，小→大，零扩展（**无符号**），**MOVZ**指令的目的地**只能**为**寄存器**

当前目的是*ptr，地址为R[EDX]的存储单元，不是寄存器，因而需两条指令；（好好理解）

第1条是零扩展，第2条是传送

总结：**源根据低位操作数类型判断用AL/AX/EAX,目的**都是**(%edx)**

13015 计算机系统原理【第三章】

考点9 通用数据传送指令 下图左：寄存器，右：内存单元

```
1    val=(val_type) *ptr;
```

序号	val_type		contofptr_type
1	char	←	char
2	int	←	char
3	unisigned	←	int
4	int	←	unisigned char
5	unisigned	←	unisigned char
6	unisigned short	←	int

语句 1 对应的指令及操作	
movb (%edx), %al	#传送
movsbl (%edx), %eax	#符号扩展,传送
movl (%edx), %eax	#传送
movzbl (%edx), %eax	#零扩展,传送
movzbl (%edx), %eax	#零扩展,传送
movw (%edx), %ax	#截断,传送

13015 计算机系统原理【第三章】

考点9 通用数据传送指令 下图左：寄存器，右：内存单元

```
1    val=(val_type) *ptr;
```

ptr：存储在**EDX**，**val**：存储在**AL/AX/EAX**类型

伪代码：AL/AX/EAX（目的）←EDX（源）

将地址为R[EDX]的**存储单元**内容送到**累加器AL/AX/EAX**中

1 **相同操作类型**(**即使一个是带符号整数类型，另一个是无符号整数类型**)，用**直接传送的指令**即可

2 char→int，小→大，符号扩展（默认有符号）

3 同1

4 unisigned char→int，小→大，零扩展（unisigned无符号）

5 同4

6 int→unisigned short，大→小，截断传送

总结：**源**都是**(%edx)**,**目的根据低位操作数类型判断用AL/AX/EAX**

13015 计算机系统原理【第三章】

考点10 逻辑运算指令

例 3.6　假设 short 型变量 x 被编译器分配在寄存器 AX 中，R[ax]=FF80H，则以下编代码段执行后变量 x 的机器数和真值分别是多少？

```
movw    %ax, %dx
salw    $2, %ax
addw    %dx, %ax
sarw    $1, %ax
```

SAL：算术左移，SAR：算术右移

1 R[dx]←R[ax]　　# 设 R[dx] = x

2 R[ax]<<2　　# R[ax] = x × 2^2 = 4x

3 R[ax]←R[ax] + R[dx]　　# R[ax] = x × 2^2 + x= 5x

4 R[ax]>>1　　# R[ax] = 5x/ 2 = 2.5x

13015 计算机系统原理【第三章】

考点10 逻辑运算指令

SAL：算术左移，SAR：算术右移

FF80H　1111 1111 1000 0000

1111 1110 0000 0000　算术左移2位，低位补0

1111 1111 1000 0000　　+ FF80H

1111 1101 1000 0000

1111 1110 1100 0000　　算术右移1位，高位补符号

原码：FEC0H，真值为：-320，所以 x = -320 / 2.5 = -128（前、后符号位不变）

复习一下：原码求真值，数值位：0正1负，若负，数值位各位取反，末位+1

13015 计算机系统原理【第三章】

考点11 程序控制流控制指令

无条件跳转指令：直接跳转到目标地址处执行

条件跳转指令：以标志位或标志组合作为跳转依据。例如：CF=1

条件设置指令：setc %dl，含义：若CF=1，则R[dl]=1，否则R[dl]=0

条件传送指令：cmovc %eax(源),%edx(目的)，含义：若CF=1，则R[edx]←R[eax]，否则R[dl]=0

调用和返回指令：CALL调用指令、RET返回指令

陷阱指令：系统调用

13015 计算机系统原理【第三章】

考点11 程序控制流控制指令

例 3.7 以下各组指令序列用于将变量 x 和 y 的某种比较结果记录到 CL 寄存器，请分析以下各组指令序列，分别判断变量 x 和 y 在 C 语言程序中的数据类型，并说明指令序列的功能。

```
第一组：cmpl    %eax, %edx      #R[eax]=x,R[edx]=y
        setb    %cl
第二组：cmpl    %eax, %edx      #R[eax]=x,R[edx]=y
        setne   %cl
第三组：cmpw    %ax, %dx        #R[ax]=x,R[dx]=y
        setl    %cl
第四组：cmpb    %al, %dl        #R[al]=x,R[dl]=y
        setae   %cl
```

13015 计算机系统原理【第三章】

考点11 程序控制流控制指令

通过查表可知：

1 setb %cl，若x < y，R[cl] = 1，否则 R[cl] = 0	无符号整数小于比较
2 setne %cl，若x != y，R[cl] = 1，否则 R[cl] = 0	两个位串不相等比较
3 setl %cl，若x < y，R[cl] = 1，否则 R[cl] = 0	带符号整数小于比较
4 setae %cl，若x >= y，R[cl] = 1，否则 R[cl] = 0	无符号整数大于等于比较

13015 计算机系统原理【第三章】

考点12 过程调用

过程调用的执行步骤（简答题）

假定过程 P 调用过程 Q，则 P 称为调用者（Caller），Q 称为被调用者（Callee）。过程调用的执行步骤如下。

1）P 将入口参数（实参）放到 Q 能访问到的地方。

2）P 将返回地址存到特定的地方，然后将控制转移到 Q。

3）Q 保存 P 的现场，并为自己的非静态局部变量分配空间。

4）执行 Q 的过程体（函数体）。

5）Q 恢复 P 的现场，并释放局部变量所占空间。

6）Q 取出返回地址，将控制转移到 P。

13015 计算机系统原理【第三章】

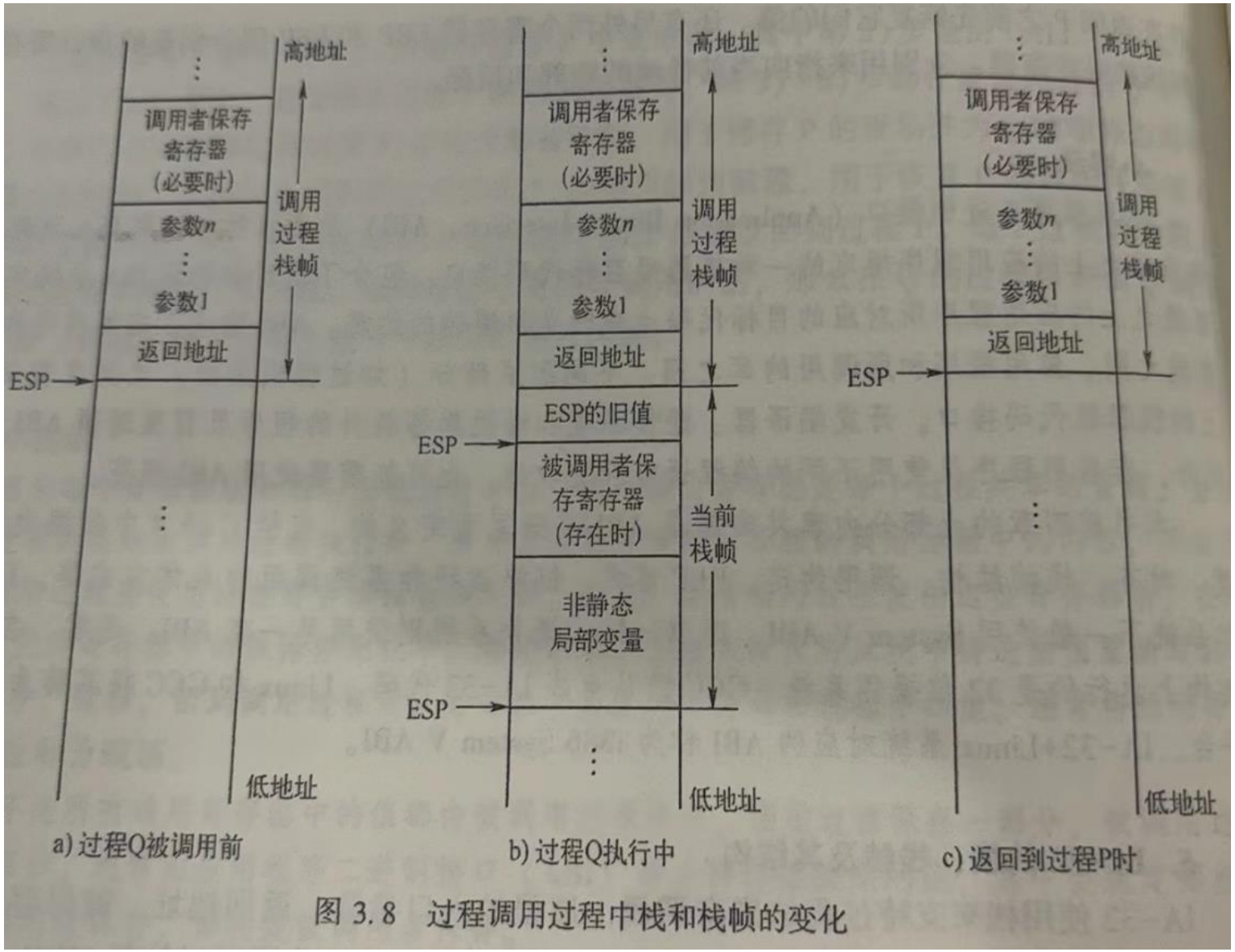


图 3.8　过程调用过程中栈和栈帧的变化

一个栈有多个栈帧

EBP 栈底，是帧指针寄存器

ESP 栈顶

当前栈帧范围：EBP ~ ESP

ESP是动态变化的，压栈、出栈的原因。

高地址

↓向下增长

低地址

13015 计算机系统原理【第三章】

考点12 过程调用

```
1  int add(int x, int y)
2      return x+y;
3  |
4
5  int caller()
6      int temp1 = 125;
7      int temp2 = 80;
8      int sum = add(temp1, temp2);
9      return sum;
10 |
```

```
1   caller:
2   pushl   %ebp
3   movl    %esp, %ebp
4   subl    $24, %esp
5   movl    $125, -12(%ebp)
6   movl    $80, -8(%ebp)
7   movl    -8(%ebp), %eax
8   movl    %eax, 4(%esp)
9   movl    -12(%ebp), %eax
10  movl    %eax, (%esp)
11  call    add
12  movl    %eax, -4(%ebp)
13  movl    -4(%ebp), %eax
14  leave
15  ret
```

#R[esp]←R[esp]-4, M[R[esp]]←R[ebp]
#R[ebp]←R[esp]

#M[R[ebp]-12]←125, 即 temp1=125
#M[R[ebp]-8]←80, 即 temp2=80
#R[eax]←M[R[ebp]-8], 即 R[eax]=temp2
#M[R[esp]+4]←R[eax], 即 temp2 入栈
#R[eax]←M[R[ebp]-12], 即 R[eax]=temp1
#M[R[esp]]←R[eax], 即 temp1 入栈
#调用 add, 将返回值保存在 EAX 中
#M[R[ebp]-4]←R[eax], 即 add 返回值送 sum
#R[eax]←M[R[ebp]-4], 即 sum 作为 caller 返回值

13015 计算机系统原理【第三章】

考点12 过程调用

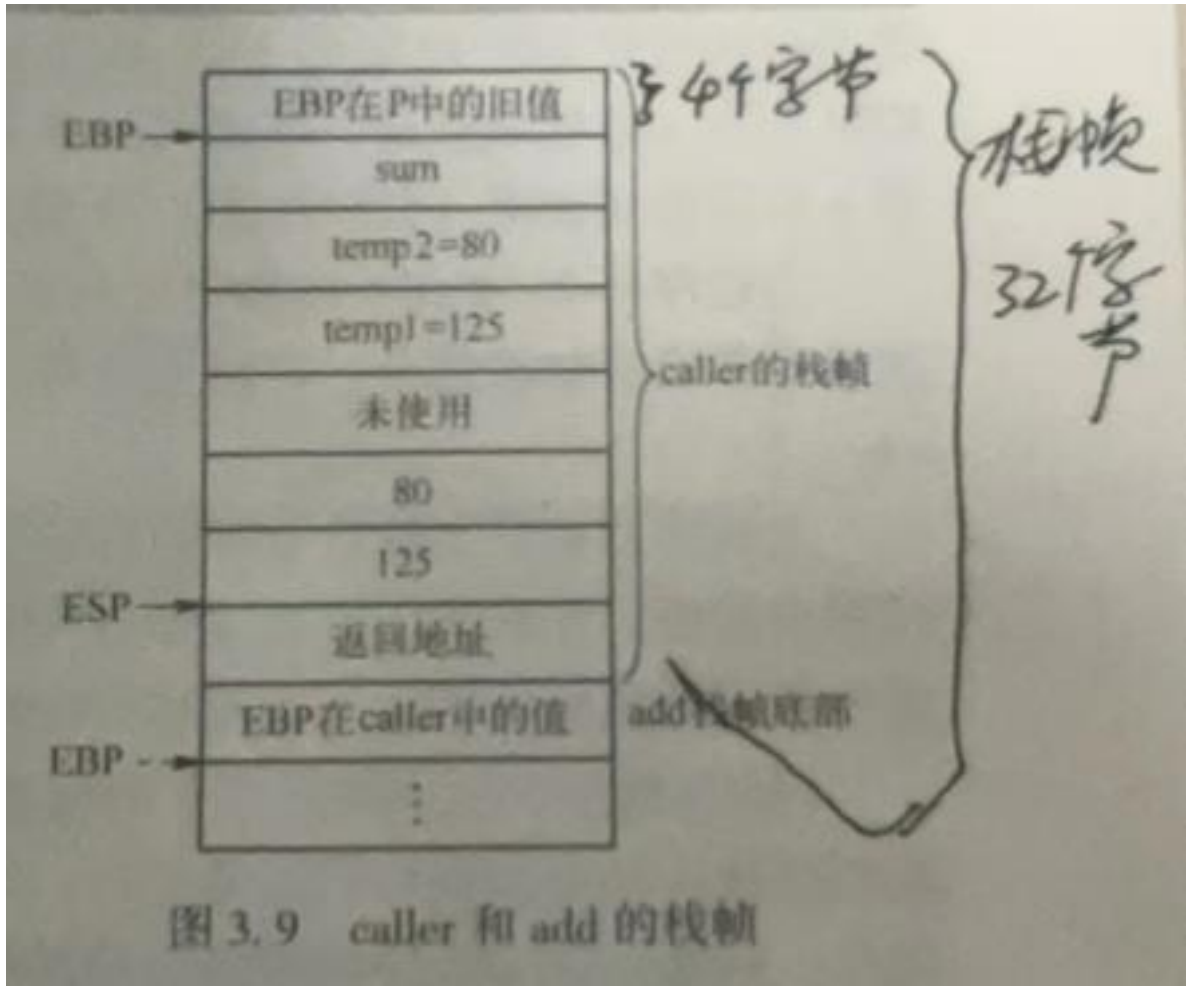


图 3.9 caller 和 add 的栈帧

一个栈有多个栈帧

EBP 栈底，是帧指针寄存器

ESP 栈顶

当前栈帧范围：EBP ~ ESP

ESP是动态变化的，压栈、出栈的原因。

高地址

↓向下增长

低地址

13015 计算机系统原理【第三章】

考点13 值传递和地址传递

按值传参、**按地址**传参

按值传参

　　基本数据类型（整形、浮点数类型、字符型）

按地址传参

　　复杂类型（数组、结构体、联合体、指针）

13015 计算机系统原理【第三章】

考点13 值传递和地址传递

程序一

```
#include <stdio.h>
main ( ) {
        int a=15, b=22;
        printf ("a=%d\tb=%d\n", a, b);
        swap (&a, &b);
        printf ("a=%d\tb=%d\n", a, b);
}
swap (int*x, int*y ) {
        int t=*x;
        *x=*y;
        *y=t;
}
```

程序二

```
#include <stdio.h>
main ( ) {
        int a=15, b=22;
        printf ("a=%d\tb=%d\n", a, b);
        swap (a, b);
        printf ("a=%d\tb=%d\n", a, b);
}
swap (int x, int y) {
        int t=x;
        x=y;
        y=t;
}
```

图 3.11　按值传递参数和按地址传送参数的程序示例

上述图 3.11 中两个程序的输出结果如图 3.12 所示。

程序一的输出：

```
a=15    b=22
a=22    b=15
```

程序二的输出：

```
a=15    b=22
a=15    b=22
```

图 3.12　图 3.11 中程序的输出结果

13015 计算机系统原理【第三章】

考点13 值传递和地址传递

程序一汇编代码片段：

```
main:
        ...
    leal    -8(%ebp), %eax
    movl    %eax, 4(%esp)
    leal    -4(%ebp), %eax
    movl    %eax, (%esp)
    call    swap
        ...
    ret
```

程序二汇编代码片段：

```
main:
        ...
    movl    -8(%ebp), %eax
    movl    %eax, 4(%esp)
    movl    -4(%ebp), %eax
    movl    %eax, (%esp)
    call    swap
        ...
    ret
```

图 3.13　两个程序中传递 swap 过程参数的汇编代码片段

13015 计算机系统原理【第三章】

考点14 选择语句

```
1   int get_lowaddr_content(int *p1, int *p2){
2       if ( p1 > p2 )
3           return *p2;
4       else
5           return *p1;
```

已知形式参数p1和p2对应的实参已压入调用过程的栈帧，p1和p2对应实参的存储地址分别为R[ebp]+8、R[ebp]+12，这里，EBP指向当前栈帧底部，**返回结果**存放在EAX中，请写出上述函数体对应的汇编代码，要求用GCC默认的AT&T格式书写

```
1    movl    8(%ebp),%eax      #R[eax]←M[R[ebp]+8]，即 R[eax]=p1
2    movl    12(%ebp), %edx    #R[edx]←M[R[ebp]+12]，即 R[edx]=p2
3    cmpl    %edx,%eax         #比较 p1 和 p2，即根据 p1-p2 的结果置标志
4    jbe     .L1               #若 p1<=p2，则转 L1 处执行
5    movl    (%edx),%eax       #R[eax]←M[R[edx]]，即 R[eax]=M[p2]
6    jmp     .L2               #无条件跳转到 L2 执行
7  .L1:
8    movl    (%eax), %eax      #R[eax]←M[R[eax]]，即 R[eax]=M[p1]
9  .L2
```

13015 计算机系统原理【第三章】

考点15 循环语句

书上有个例题3.10，大家有兴趣的可以自行看一下，有不懂的地方可以私聊我，我可以给大家解释说明。

根据对应的汇编代码填写函数中缺失的C语言代码
这个考点，既得懂汇编代码，还得懂C语言代码，成本太高了，这个大概率是不考。

谢 谢 大 家