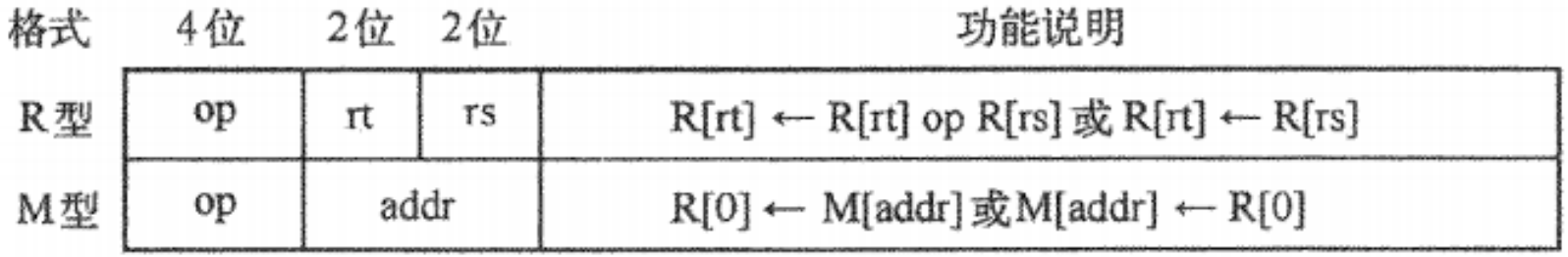
13015 计算机系统原理

第3章 程序的转换及机器级表示

13015 计算机系统原理【第3章考点解析】

考点1 指令



格式	4位	2位	2位	功能说明
R型	op	rt	rs	R[rt] ← R[rt] op R[rs] 或 R[rt] ← R[rs]
M型	op	addr		R[0] ← M[addr] 或M[addr] ← R[0]

图 1.2 定长指令字格式

op为操作码字段：

1）**R型指令** 的 op 为**0000**定义为**传送**(**mov**)操作，op 为**0001**定义为**加**(**add**)操作

比如：指令 **0001 0001**的功能为 **R[0]←R[0]+R[1]**，表示将 0 号 和 1 号寄存器内容相加的结果送 0 号寄存器。

2）**M型指令**的 op 为**1110**定义为**取数**(**load**)操作， op 为**1111**定义为**存数**(**store**)操作。

比如：指令 **1110 0110** 的功能为 **R[0]←M[0110]**，表示将 6 号主存单元（地址为 0110）中的内容取到 0 号寄存器

13015 计算机系统原理【第3章考点解析】

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

R[r]：表示寄存器 r 的内容

M[addr]：表示存储单元 addr 的内容

M[PC]：表示 PC 所指存储单元的内容

M[R[r]]：表示寄存器 r 的内容所指的存储单元的内容

AT&T格式指令：源，目的　　**顺序**（本教材使用）

Intel 格式指令：目的，源　　**逆序**

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]

13015 计算机系统原理【第3章考点解析】

考点3 指令系统设计风格

按指令格式的**复杂度**分以下两种：

1）**CISC** 风格指令系统（**复杂**指令集计算机，**C**omplex **I**nstruction **S**et **C**omputer）

例如：本书介绍的 **Intel x86** 指令系统就是典型的 **CISC** 架构

2）**RISC** 风格指令系统（**精简**指令集计算机，**R**educed **I**nstruction **S**et **C**omputer）

① 指令数目少　　② 指令格式规整，采用定长指令字方式

③ 只有Load/Store指令中的数据需要访存　　④ 采用大量通用寄存器

指令的操作数有以下三种：

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

2）通用寄存器（R）　　例如：movl $80, -8(%ebp)　　# M[R[ebp]-8] ← 80

3）存储单元（S）　　例如：movl $80, -8(%ebp)　　# M[R[ebp]-8] ← 80

指令类型可以是：RR、RS、SI、SS等

13015 计算机系统原理【第3章考点解析】

考点3 指令系统设计风格

25. 简述 RISC 指令集的主要特点，并写出它的全称。

【答案】：

RISC：精简指令集计算机。

RISC 指令系统的主要特点如下：

① 指令数目少；

② 指令格式规整，采用定长指令字方式，操作码和操作数地址等字段的长度固定；

③ 只有 Load/Store 指令中的数据需要访存，这种称为 Load/Store 型指令风格；

④ 采用大量通用寄存器。

【2025年10月】

13015 计算机系统原理【第3章考点解析】

考点4 生成机器代码的过程

一个 **C 语言程序**转换为可执行目标代码的过程分为以下 4 个步骤

1）预处理　　使用命令 "gcc -E **prog1.c** -o **prog1.i**"

2）编译　　使用命令 "gcc -S **prog1.i** -o **prog1.s**" 或 "gcc -E **prog1.c** -o **prog1.s**"

3）汇编　　使用命令 "gcc -c **prog1.s** -o **prog1.o**"

4）链接　　使用命令 "gcc **prog1.o prog1.o** -o **prog**"

也可以用命令一步到位生成最终的可执行文件：gcc –O1 **prog1.c prog2.c** -o **prog**

13015 计算机系统原理【第3章考点解析】

考点4 生成机器代码的过程

例 3.1 在 IA-32+Linux 平台上，对下列源程序 test. c 使用 GCC 命令进行相应的处理，以分别得到预处理后的文件 test. i、汇编代码文件 test. s 和可重定位目标文件 test. o。这些输出文件中，哪些是可显示的文本文件？哪些是不能显示的二进制文件？请给出所有可显示文本文件的输出结果。

```
1  // test. c
2
3  int add(int i, int j ){
4      int x = i + j;
5      return x;
6  }
```

13015 计算机系统原理【第3章考点解析】

考点4 生成机器代码的过程

汇编代码文件 **test.s** 是可显示文本文件

```
.....
add:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    12(%ebp), %eax
        movl    8(%ebp), %edx
        leal    (%edx, %eax), %eax
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

13015 计算机系统原理【第3章考点解析】

考点4 生成机器代码的过程

使用命令 "**objdump -d test.o**" 可以得到以下显示结果

test.o 是可重定位目标文件，因而目标代码从相对地址 0 开始

```
00000000 <add>:
   0:   55                      push   %ebp
   1:   89e5                    mov    %esp, %ebp
   3:   83 ec 10                sub    $0x10,%esp
   6:   8b 45 0c                mov    0xc(%ebp),%eax
   9:   8b 55 08                mov    0x8(%ebp),%edx
   c:   8d 04 02                lea    (%edx,%eax,1),%eax
   f:   89 45 fc                mov    %eax,-0x4(%ebp)
  12:   8b 45 fc                mov    -0x4(%ebp),%eax
  15:   c9                      leave
  16:   c3                      ret
```

13015 计算机系统原理【第3章考点解析】

考点4 生成机器代码的过程

可以用命令 "**gcc -o test main.o test.o**" 来生成可执行文件 **test**

使用命令 "**objdump -d test**" 可以得到以下显示结果

```
080483d4 <add>:
 80483d4:   55                    push   %ebp
 80483d5:   89 e5                 mov    %esp,%ebp
 80483d7:   83 ec 10              sub    $0x10,%esp
 80483da:   8b 45 0c              mov    0xc(%ebp),%eax
 80483dd:   8b 55 08              mov    0x8(%ebp),%edx
 80483e0:   8d 04 02              lea    (%edx,%eax,1),%eax
 80483e3:   89 45 fc              mov    %eax,-0x4(%ebp)
 80483e6:   8b 45 fc              mov    -0x4(%ebp),%eax
 80483e9:   c9                    leave
 80483ea:   c3                    ret
```

13015 计算机系统原理【第3章考点解析】

考点5 Intel和AT&T格式指令区别

特性	Intel 格式	AT&T 格式
指令例子	mov ax [ebp+edx*4+8]	movw 8(%ebp, %edx, 4), %ax
操作数顺序	目标, 源	源, 目标
立即数表示	8	8
寄存器表示	ax	%ax
存储单元表示	[]	()
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)
	[ebp+edx*4+8]	8(%ebp, %edx, 4)
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**"。

例如： "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3章考点解析】

考点6 数据类型及格式

表 3.1　C 语言基本数据类型和 IA-32 操作数类型的对应关系

C 语言声明	Intel 操作数类型	汇编指令长度后缀	存储长度/位
(unsigned) char	整数 / 字节	b	8
(unsigned) short	整数 / 字	w	16
(unsigned) int	整数 / 双字	l	32
(unsigned) long int	整数 / 双字	l	32
(unsigned) long long int	–	–	2×32
char *	整数 / 双字	l	32
float	单精度浮点数	s	32
double	双精度浮点数	l	64
long double	扩展精度浮点数	t	80/96

13015 计算机系统原理【第3章考点解析】

考点7 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

EAX			
		AX	
		AH	AL
		高8位	低8位

助记：2字母16位，3字母32位，看见末尾L、H是8位

同理：

BL、CL、DL → 低 8 位

BH、CH、DH → 高 8 位

BX、CX、DX → 16 位

EBX、ECX、EDX → 32 位

13015 计算机系统原理【第3章考点解析】

考点7 定点寄存器组

定点寄存器组：

1）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）2 个专用寄存器

EIP是指令指针寄存器，EFLAGS是标志寄存器

3）6 个段寄存器

13015 计算机系统原理【第3章考点解析】

考点8 条件标示

1）**OF(Overflow Flag)：溢出标示**，反映**带符号数**的运算结果是否超过相应数值范围。

例如：对于 8 位带符号整数，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，此时 OF=1；否则 OF=0

2）**SF(Sign Flag)：符号标示**，反映**带符号数**运算结果的符号。

负数时，SF=1；否则SF=0　　**助记：0正1负**

3）**ZF(Zero Flag)：零标示**，反映运算结果是否为0。若结果为0，ZF=1；否则ZF=0

4）**CF(Carry Flag)：进/借位标示**，反映**无符号整数**加（减）运算后的进（借）位情况。

有进（借）位则CF=1；否则CF=0

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（1+1=10）

13015 计算机系统原理【第3章考点解析】

考点8 定点算术运算指令

1）加/减运算指令　　例如：ADD(+) SUB(-)

2）增/减运算指令　　例如：INC(++) DEC(--)

3）取负指令　　例如：NEG(-)

4）比较运算指令　　例如：CMP(<,<=,>,>=)

5）乘/除运算指令

例如：

MUL(*)　　无符号整数

IMUL(*)　　带符号整数

DIV(/,%)　　无符号整数

IDIV(/,%)　　带符号整数

13015 计算机系统原理【第3章考点解析】

表 3.5　定点算术运算指令汇总

指令	显式操作数	影响的标志	操作数类型	AT&T 指令助记符	对应 C 运算符
ADD	2 个	OF、ZF、SF、CF	无/带符号整数	addb、addw、addl	+
SUB	2 个	OF、ZF、SF、CF	无/带符号整数	subb、subw、subl	-
INC	1 个	OF、ZF、SF	无/带符号整数	incb、incw、incl	++
DEC	1 个	OF、ZF、SF	无/带符号整数	decb、decw、decl	--
NEG	1 个	OF、ZF、SF、CF	无/带符号整数	negb、negw、negl	-
CMP	2 个	OF、ZF、SF、CF	无/带符号整数	cmpb、cmpw、cmpl	<, <=, >, >=
MUL	1 个	OF、CF	无符号整数	mulb、mulw、mull	*
IMUL	1 个	OF、CF	带符号整数	imulb、imulw、imull	*
IMUL	2 个	OF、CF	带符号整数	imulb、imulw、imull	*
IMUL	3 个	OF、CF	带符号整数	imulb、imulw、imull	*
DIV	1 个	无	无符号整数	divb、divw、divl	/,%
IDIV	1 个	无	带符号整数	idivb、idivw、idivl	/,%

13015 计算机系统原理【第3章考点解析】

考点8 定点算术运算指令

5. 下列不属于 IA-32 定点算术运算指令的是

A. ADD/SUB　　B. IN/OUT　　C. MUL/IMUL　　D. INC/DEC

【答案】：B【2025年10月】

13015 计算机系统原理【第3章考点解析】

考点8 条件标示

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。

	F	F	F	A
+	F	F	F	0
1	F	F	E	A

Intel 指令 "**add ax, bx**" 的功能是 **R[ax]←R[ax]+R[bx]**。

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

1）无符号**整数**解释

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

16 位无符号数最大值为 2^{16}-1=65535，结果溢出，所以：**CF=1**：无符号数相加产生进位（溢出）

结果不为0，所以：**ZF=0**

13015 计算机系统原理【第3章考点解析】

考点8 条件标示

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。

	F	F	F	A
+	F	F	F	0
1	F	F	E	A

2）带符号**整数**解释（补码表示）

R[ax] = FFFAH = -6，R[bx] = FFF0H= -16，相加：-6 + (-16) = -22

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

16 位带符号数最大值为 -2^{15}~2^{15}-1，即：-32767~32768，结果正确，**OF=0**

结果不为0，所以：**ZF=0**

结果是负数，所以：**SF=1**（结果最高位为 1，表示负数）

13015 计算机系统原理【第3章考点解析】

考点8 条件标示

例 3.5 假设 R[eax] = 0000 00B4H，R[ebx] = 0000 0011H，M[0000 00F8H] = 0000 00A0H，请问：

① 执行指令“mulb %bl”后，哪些寄存器的内容会发生变化？与执行“imulb %bl”指令所发生的变化是否一样？为什么？两条指令得到的 CF 和 OF 标志各是什么？请用该例给出的数据验证你的结论。

② 执行指令“imull $-16,(%eax,%ebx,4),%eax”后哪些寄存器和存储单元发生了变化？乘积的机器数和真值各是多少？

13015 计算机系统原理【第3章考点解析】

考点8 条件标示

因为 R[eax]=0000 00B4H，R[ebx]=0000 0011H，所以，R[al]=B4H，R[bl]=11H。【低8位】

① **无符号整数乘法**，指令“**mulb %bl**”中指出的操作数为8位，故指令的功能为“**R[ax]←R[al]×R[bl]**”，因此，改变内容的寄存器是AX。

B4H = 11×16^1+4×16^0=**180**　　11H = 1×16^1+1×16^0=**17**

指令执行后R[ax]=**0BF4H**，即十进制数3060=**180×17**

因为高8位乘积不为全0（即乘积高8位中含有效数位：**0BH**），故CF=1，OF=1。

			B	4
×			1	1
			B	4
		B	4	
	0	B	F	4

带符号整数乘法，执行指令“**imulb %bl**”后，同上

B4H = 1011 0100 = 1100 1011 + 1 = 1100 1100 = -100 1100 = **-76**

11H = 0001 0001 = **17**　　指令执行后R[ax]=**FAF4H**，即十进制数**-1292=-76×17**

因为高9位乘积不为全0或全1（即乘积高8位中含有效数位：**FAH**），故CF=1，OF=1。

13015 计算机系统原理【第3章考点解析】

考点8 条件标示

② 带符号整数乘法，指令“imull $-16, (%eax,%ebx,4), %eax”的功能是“R[eax]←(-16)×M[R[eax]+4×R[ebx]]”

其中，第二个乘数所在的存储单元地址为

R[eax]+4×R[ebx]=0xB4+(0x11<<2)=0xF8=0000 00F8H

0000	0000	0000	0000	0000	1010	0000	0000
1111	1111	1111	1111	1111	0101	1111	1111
							1
1111	1111	1111	1111	1111	0110	0000	0000
F	F	F	F	F	6	0	0

【备注】：(0x11<<2)：左移2位实际是二进制左移

或 R[eax]+4×R[ebx]=B4H+4×11H=B4H+44H=F8H=0000 00F8H，

因为M[0000 00F8H]=0000 00A0H=160，与-16相乘后得到一个负的乘积，因此乘积的符号为负。仅考虑低32位乘积，其数值部分绝对值的机器数为0000 00A0H<<4($16=2^4$)=0000 0A00H，是负数，对其各位取反末位加1，得到机器数为FFFF F600H，

即指令执行后EAX中存放的内容为FFFF F600H，其真值为-2560。

【备注】：二进制左移4位 = 十六进制左移1位

13015 计算机系统原理【第3章考点解析】

考点9 寻址方式

1）立即寻址

2）寄存器寻址

3）存储器寻址

- 实地址模式
- 保护模式

IA-32采用段页式虚拟存储管理方式

13015 计算机系统原理【第3章考点解析】

考点10 浮点寄存器栈和多媒体扩展寄存器组

IA-32的浮点处理架构有两种：

1）浮点协处理器 x87 架构

它是一种栈结构

2）由 MMX 发展而来的 SSE 架构

采用**单指令多数据**（Single Instruction Multi Data, SIMD） 技术

13015 计算机系统原理【第3章考点解析】

考点11 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　　【**带符号**】

例如：**movsbw %al，%ax**：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　　【**无符号**】

例如：**movzwl %ax，%eax**：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

【注意】：**MOVS**和**MOVZ**指令的**目的地址**只能是**寄存器编号**。

XCHG：数据交换指令，将两个寄存器内容互换。

例如：xchg**b**：表示字节交换。

13015 计算机系统原理【第3章考点解析】

考点11 通用数据传送指令

PUSH:

① 先执行 **R[sp]←R[sp]-2** 或 **R[esp]←R[esp]-4**

② 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】

① 将一个字或双字从 SP 或 ESP 指示的栈单元送入指定寄存器

② 再执行 **R[sp] ← R[sp]+2** 或 **R[esp] ← R[esp]+4**

例如：**popl** 表示双字出栈，**popw** 表示字出栈

栈(Stack)是一种采用“先进后出”方式进行访问的一块存储区，在处理过程调用时非常有用。

大多数情况下，栈是从高地址向低地址增长的。

13015 计算机系统原理【第3章考点解析】

考点11 通用数据传送指令

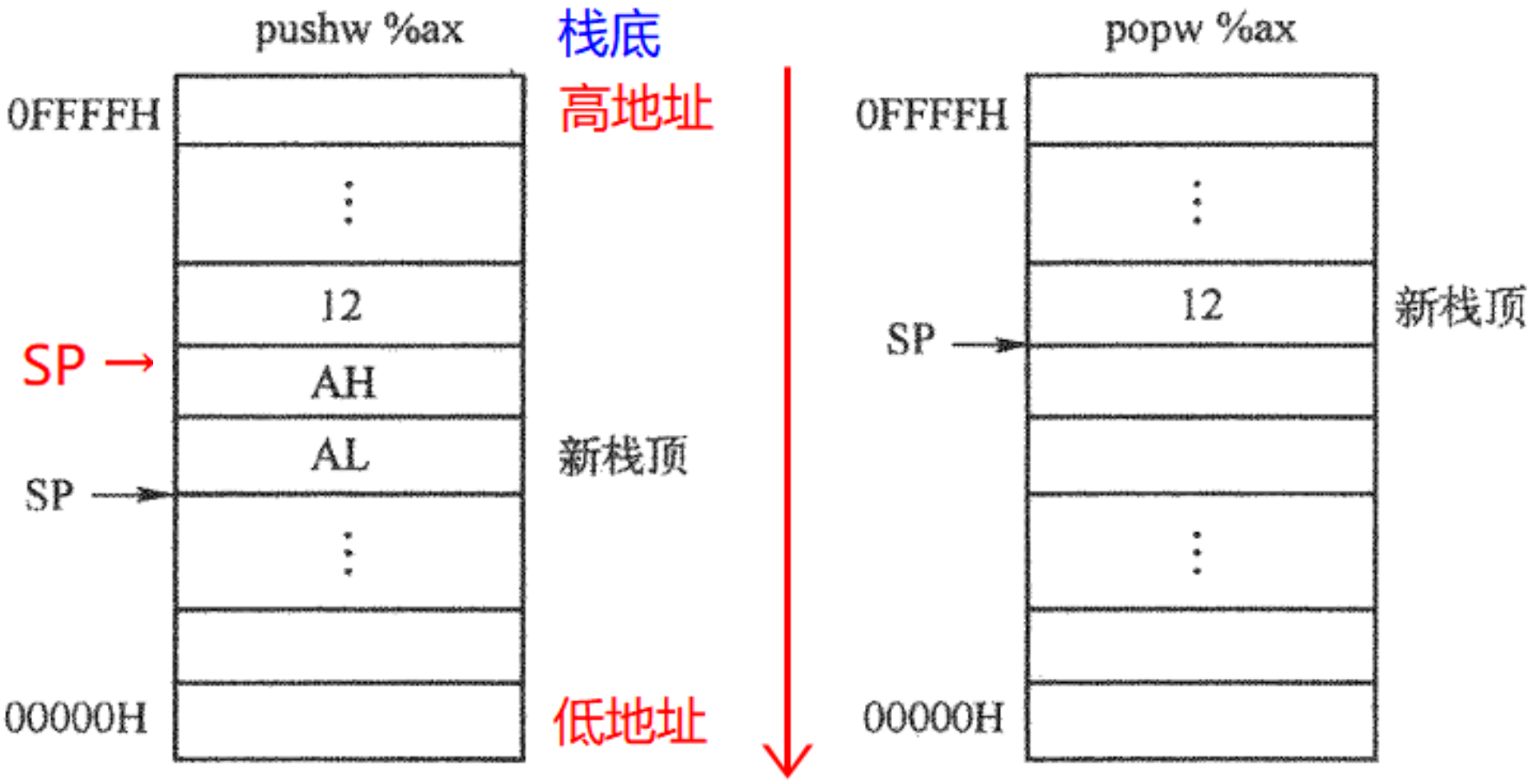


图 3.6 pushw 和 popw 指令的执行

13015 计算机系统原理【第3章考点解析】

考点11 通用数据传送指令

3. 下列说法不正确的是

A. 变址寻址时，有效数据存放在内存中

B. 数据交换指令，将两个寄存器内容互换

C. 堆栈指针 SP 的内容表示当前堆栈内所存储的数据的个数

D. 内存中指令的寻址和数据的寻址是交替进行的

【答案】：C【2025年04月】

15. 栈是一种采用_____方式进行访问的一块存储区，在执行 pushw% ax 指令之后，SP 指向存放有 AX 内容的单元，即当前刚入栈的_____。

【答案】：前进后出、数据【2025年04月】

13015 计算机系统原理【第3章考点解析】

考点11 通用数据传送指令

24. 简述栈在处理过程调用时的作用，并列举 16 位架构下可用的栈相关指令。

【答案】：栈是一种采用 “先进后出” 方式进行访问的一块存储区，在处理过程调用时非常有用。大多数情况下，栈从高地址向低地址增长。16 位架构下有 pushw 和 popw 指令分别表示进栈和出栈。

【2025年04月】

13015 计算机系统原理【第3章考点解析】

考点11 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;
```

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/AX/EAX → 8/16/32位

从寄存器取值，**AT&T** 用 %AL/%AX/%EAX

ptr 存储在 EDX → 32位

* （指针）可以看作是取值操作，从主存取值，

AT&T 用 (%EDX)

13015 计算机系统原理【第3章考点解析】

考点11 通用数据传送指令

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

序号	val_type	contofptr_type	语句 1 对应的指令及操作
1	char	char	movb (%edx), %al #传送
2	int	char	movsbl (%edx), %eax #符号扩展,传送
3	unsigned	int	movl (%edx), %eax #传送
4	int	unsigned char	movzbl (%edx), %eax #零扩展,传送
5	unsigned	unsigned char	movzbl (%edx), %eax #零扩展,传送
6	unsigned short	int	movw (%edx), %ax #截断,传送

非常重要：绝大多数 **AT&T** 指令后缀，就是由「**寄存器**」决定宽度。

【注意】：**MOVS**和**MOVZ**指令的**目的地址**只能是**寄存器编号**。

val 存储在 AL/AX/EAX → 8/16/32位

从寄存器取值，**AT&T** 用 %AL/%AX/%EAX

ptr 存储在 EDX → 32位

* （指针）可以看作是取值操作，从主存取值，**AT&T** 用 (%EDX)

13015 计算机系统原理【第3章考点解析】

考点11 通用数据传送指令

2 *ptr=(contofptr_type) val;

序号	val_type	contofptr_type	语句2对应的指令及操作
1	char	char	movb %al, (%edx) #传送
2	int	char	movb %al, (%edx) #截断,传送
3	unsigned	int	movl %eax, (%edx) #传送
4	int	unsigned char	movb %al, (%edx) #截断,传送
5	unsigned	unsigned char	movb %al, (%edx) #截断,传送
6	unsigned short	int	movzwl %ax, %eax #零扩展 movl %eax, (%edx) #传送

非常重要：绝大多数 **AT&T** 指令后缀，就是由「**寄存器**」决定宽度。

【注意】：**MOVS**和**MOVZ** 指令的**目的地址**只能是**寄存器编号**。

val 存储在 AL/AX/EAX → 8/16/32位

从寄存器取值，**AT&T** 用 %AL/%AX/%EAX

ptr 存储在 EDX → 32位

* （指针）可以看作是取值操作，从主存取值，**AT&T** 用 (%EDX)

13015 计算机系统原理【第3章考点解析】

考点12 地址传送指令

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

例如：**lea**l 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3章考点解析】

考点12 地址传送指令

例 3.2 将以下 Intel 格式的汇编指令转换为 GCC 默认的 AT&T 格式汇编指令。说明每条指令的含义。

1	push	ebp	pushl	%ebp	#R[esp]←R[esp]-4, M[R[esp]]←R[ebp], 双字
2	mov	ebp, esp	movl	%esp, %ebp	#R[ebp]←R[esp], 双字
3	mov	edx,DWORD PTR [ebp+8]	movl	8(%ebp), %edx	#R[edx]←M[R[ebp]+8], 双字
4	mov	bl,255	movb	$255, %bl	#R[bl]←255, 字节
5	mov	ax,WORD PTR [ebp+edx*4+8]	movw	8(%ebp,%edx,4), %ax	#R[ax]←M[R[ebp]+R[edx]×4+8], 字
6	mov	WORD PTR [ebp+20],dx	movw	%dx, 20(%ebp)	#M[R[ebp]+20]←R[dx], 字
7	lea	eax, [ecx+edx*4+8]	leal	8(%ecx,%edx,4), %eax	#R[eax]←R[ecx]+R[edx]×4+8, 双字

说明：1 push ebp

① 预留空间　　高地址 → 地地址　　**ESP寄存器**指向当前栈顶

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）

非常重要：绝大多数 **AT&T** 指令后缀，就是由「**寄存器**」决定宽度。

13015 计算机系统原理【第3章考点解析】

考点13 移位指令

左移:

1）**SHL**：**逻辑左移**，每左移一次，最高位送入 CF，并在低位补 0。 **【无符号数】**

例如：原数：00001010 (十进制 10) → 左移1位 ($\times 2^1$)，低位补 0，最高位（0）送入 CF

结果：00010100 (十进制 20)

标志位变化：CF = 0

2）**SAL**：**算术左移**，每左移一次，最高位送入 CF，并在低位补 0。 **【带符号数】**

如果移位前后符号位发生变化，则 OF=1，表示左移后结果溢出。**这是 SAL 与 SHL 的不同之处**

例如：原数：00001010 (十进制 +10) → 左移1位 ($\times 2^1$)，低位补 0，最高位（0）送入 CF

结果：00010100 (十进制 +20)

标志位变化：CF = 0，OF = 0（移位前符号位是 0，移位后还是 0，无变化→不溢出）

13015 计算机系统原理【第3章考点解析】

考点13 移位指令

右移:

1）**SHR：逻辑右移**，每右移一次，最低位送入 CF，并在高位补 0。 **【无符号数】**

例如：原数：**11010011** (十进制 211) → 右移1位 (/2^1)，高位补 0，最低位（**1**）送入 CF

结果：**01101001** (十进制 105)

标志位变化：CF = **1**

2）**SAR：算术右移**，每右移一次，操作数的最低位送入 CF，并在高位补符号位。 **【带符号数】**

例如：原数：**11010011** (十进制 -45) → 右移1位 (/2^1)，高位补 0，最低位（**1**）送入 CF

结果：**11101001** (十进制 -23)

标志位变化：CF = **1**

【注意】：211/2=105、-45/2 向下取整（符合有符号数除法逻辑）

13015 计算机系统原理【第3章考点解析】

考点13 移位指令

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，则以下汇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
```

显然这里的汇编指令是 GCC 默认的 AT&T 格式，$2 和 $1 分别表示立即数 2 和 1。

```
movw   %ax, %dx      # R[dx] ← R[ax]
salw   $2, %ax       # R[ax] ← R[ax] << 2  (或 R[ax] ← R[ax] × 4)
addw   %dx, %ax      # R[ax] ← R[ax] + R[dx]
sarw   $1, %ax       # R[ax] ← R[ax] >> 1  (算术右移，高位补符号位)
```

13015 计算机系统原理【第3章考点解析】

考点13 移位指令

显然这里的汇编指令是 GCC 默认的 AT&T 格式，$2 和 $1 分别表示立即数 2 和 1。

```
movw   %ax, %dx      # R[dx] ← R[ax]                          R[dx] = x
salw   $2, %ax       # R[ax] ← R[ax] << 2  (或 R[ax] ← R[ax] × 4)   R[ax] = 4x
addw   %dx, %ax      # R[ax] ← R[ax] + R[dx]                  R[ax] = 5x
sarw   $1, %ax       # R[ax] ← R[ax] >> 1  (算术右移，高位补符号位)   R[ax] = 5x/2
```

假设上述代码段执行前 **R[ax] = x**，则执行 ((x<<2)+x)>>1 后，R [ax] = 5x/2。

因为 short 型变量为带符号整数，因而采用算术移位指令 salw，后缀 w 表示操作数的长度为一个字，即 16 位。

13015 计算机系统原理【第3章考点解析】

考点13 移位指令

		1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	FF80H
		1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	FF80H 算术左移2位，低位补0
	+	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	FF80H
	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	
		1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	算术右移1位，高位补符号
			F				E				C				0			
		1	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	
	+																1	
		1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	负数
	=																	-320

算术左移时，AX 中的内容 FF80H 在移位前、后符号未发生变化，故 OF=0，没有溢出。

最终 AX 的内容为 **FEC0H**，解释为 short 型整数时，其值为 -320。

验证：x = -128（**FF80H** - **补码求真值**），5x/2 = -320。经验证，结果正确。

13015 计算机系统原理【第3章考点解析】

考点14 程序执行流控制指令

1）**无条件跳转指令**：JMP的执行结果就是直接跳转到目标地址处执行

例如：jmp * . L8(, %eax, 4)　　# R[eip]←M[. L8+R[eax]×4]

2）**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 AND ZF=0

3）**条件设置指令**：用来将条件标志组合得到的条件值设置到一个8位通用寄存器中

例如：setc %dl，含义：若CF=1，则R[dl]=1；否则R[dl]=0

指　令	跳转条件	说　明
jc label	CF=1	有进位/借位

13015 计算机系统原理【第3章考点解析】

考点14 程序执行流控制指令

4）**条件传送指令**：如果符合条件就进行传送操作

格式：CMOVcc DST，SRC　　【Intel】

格式：CMOVcc SRC，DST　　【AT&T】

例如：cmovc %eax，%edx，含义：若CF=1，则R[edx]←R[eax]；否则什么都不做

指　令	跳转条件	说　明
jc label	CF=1	有进位/借位

5）**调用和返回指令**

① 调用指令 CALL；② 返回指令 RET

6）**陷阱指令**

重要作用之一：系统调用

表 3.6　条件跳转指令

序号	指　令	跳转条件	说　明
1	jc label	CF=1	有进位/借位
2	jnc label	CF=0	无进位/借位
3	je/jz label	ZF=1	相等/等于零
4	jne/jnz label	ZF=0	不相等/不等于零
5	js label	SF=1	是负数
6	jns label	SF=0	是非负数
7	jo label	OF=1	有溢出
8	jno label	OF=0	无溢出
9	ja/jnbe label	CF=0 AND ZF=0	无符号整数 A>B
10	jae/jnb label	CF=0 OR ZF=1	无符号整数 A≥B
11	jb/jnae label	CF=1 AND ZF=0	无符号整数 A<B
12	jbe/jna label	CF=1 OR ZF=1	无符号整数 A≤B
13	jg/jnle label	SF=OF AND ZF=0	带符号整数 A>B
14	jge/jnl label	SF=OF OR ZF=1	带符号整数 A≥B
15	jl/jnge label	SF≠OF AND ZF=0	带符号整数 A<B
16	jle/jng label	SF≠OF OR ZF=1	带符号整数 A≤B

13015 计算机系统原理【第3章考点解析】

考点14 程序执行流控制指令

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。根据以下各组指令序列，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3章考点解析】

考点14 程序执行流控制指令

CMP 指令通过执行减法来设置条件标志位，每组中第二条 SETcc 指令中使用的条件标志都是由 x 和 y 相减后设置的。

```
第一组：cmpl    %eax, %edx    #R[eax]=x,R[edx]=y
        setb    %cl
```

11	jb/jnae label	CF=1 AND ZF=0	无符号整数 A<B

无符号整数小于比较，因此，x 和 y 可能是 **unsigned、unsigned long** 或 **指针型** 数据。

```
第二组：cmpl    %eax, %edx    #R[eax]=x,R[edx]=y
        setne   %cl
```

4	jne/jnz label	ZF=0	不相等/不等于零

两个位串的不相等比较，因此，x 和 y 可能是 **unsigned、int、unsigned long、long** 或 **指针型** 数据。

13015 计算机系统原理【第3章考点解析】

考点14 程序执行流控制指令

CMP 指令通过执行减法来设置条件标志位，每组中第二条 SETcc 指令中使用的条件标志都是由 x 和 y 相减后设置的。

第三组：cmpw %ax, %dx #R[ax]=x,R[dx]=y

setl %cl

15	jl/jnge label	SF≠OF AND ZF=0	带符号整数 A<B

带符号整数小于比较，因此，x 和 y 只能是 **short** 型数据。

第四组：cmpb %al, %dl #R[al]=x,R[dl]=y

setae %cl

10	jae/jnb label	CF=0 OR ZF=1	无符号整数 A≥B

无符号整数大于小于比较，因此，x 和 y 可能是 **unsigned char** 或 **char** 数据。

13015 计算机系统原理【第3章考点解析】

考点14 程序执行流控制指令

例 3.8 以下各组指令序列用于测试变量 x 的某种特性，并将测试结果记录到 CL 寄存器。根据以下各组指令序列，分别判断数据 x 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：testl    %eax, %eax     #R[eax]=x
        sete     %cl
第二组：testl    %eax, %eax     #R[eax]=x
        setge    %cl
第三组：testw    %ax, %ax       #R[ax]=x
        setns    %cl
第四组：testb    %al, $15       #R[al]=x
        setz     %cl
```

13015 计算机系统原理【第3章考点解析】

考点14 程序执行流控制指令

TEST 指令执行后，OF=CF=0，而 ZF 和 SF 则根据两个操作数相 “与” 的结果来设置：若结果为全 0，则 ZF=1；若最高位为 1，则 SF=1。

	0	1	0	1
&	0	1	0	1
	0	1	0	1

第一组：
```
testl   %eax, %eax      #R[eax]=x
sete    %cl
```

3	je/jz label	ZF=1	相等/等于零

x 可能是 **unsigned、int、unsigned long、long 或 指针型** 数据。

第二组：
```
testl   %eax, %eax      #R[eax]=x
setge   %cl
```

14	jge/jnl label	SF=OF OR ZF=1	带符号整数 A≥B

SF=OF=0 OR ZF=1，带符号整数大于等于 0 比较，因此，x 可能是 int 或 long 型数据。

13015 计算机系统原理【第3章考点解析】

考点14 程序执行流控制指令

TEST 指令执行后，OF=CF=0，而 ZF 和 SF 则根据两个操作数相 “与” 的结果来设置：若结果为全 0，则 ZF=1；若最高位为 1，则 SF=1。

第三组：testw %ax, %ax #R[ax]=x

setns %cl

6	jns label	SF=0	是非负数

判断 x 是否大于等于 0，因此，x 只能是 **short** 型数据。

	1	1	1	1	0	1	0	1
&	0	0	0	0	1	1	1	1
	0	0	0	0	0	1	0	1

第四组：testb %al, $15 #R[al]=x

setz %cl

3	je/jz label	ZF=1	相等/等于零

判断 x 的低 4 位是否为 0，因此，x 可能是 **char、signed char** 或 **unsigned char 型** 数据。

13015 计算机系统原理【第3章考点解析】

考点15 过程调用

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

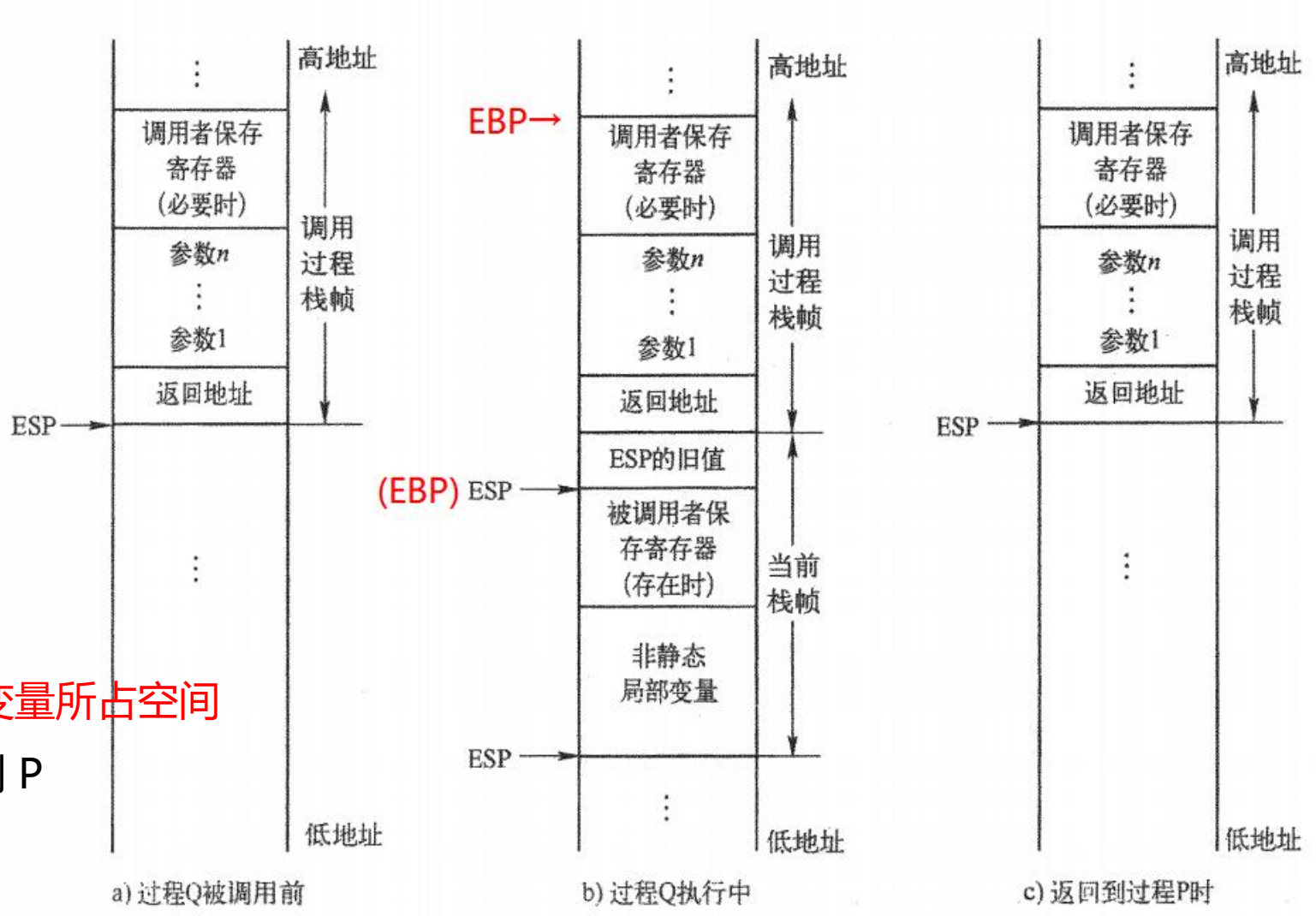


图 3.8 过程调用过程中栈和栈帧的变化

13015 计算机系统原理【第3章考点解析】

考点15 过程调用

下面以一个最简单的例子来说明过程调用的机器级实现。假定有一个函数 add 实现两个数相加，另一个过程 caller 调用 add，以计算 125+80 的值，对应的 C 语言程序如下

```
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  }
```

13015 计算机系统原理【第3章考点解析】

考点15 过程调用

经 GCC 编译后 caller 过程对应的代码如下（# 后面的文字是注释）。

```
1   caller:
2   pushl   %ebp
3   movl    %esp, %ebp
4   subl    $24, %esp
5   movl    $125, -12(%ebp)        #M[R[ebp]-12] ←125, 即 temp1=125
6   movl    $80, -8(%ebp)          #M[R[ebp]-8] ←80, 即 temp2=80
7   movl    -8(%ebp), %eax         #R[eax]←M[R[ebp]-8], 即 R[eax]=temp2
8   movl    %eax, 4(%esp)          #M[R[esp]+4]←R[eax], 即 temp2 入栈
9   movl    -12(%ebp), %eax        #R[eax]←M[R[ebp]-12], 即 R[eax]=temp1
10  movl    %eax, (%esp)           #M[R[esp]]←R[eax], 即 temp1 入栈
11  call    add                    #调用 add, 将返回值保存在 EAX 中
12  movl    %eax, -4(%ebp)         #M[R[ebp]-4]←R[eax], 即 add 返回值送 sum
13  movl    -4(%ebp), %eax         #R[eax]←M[R[ebp]-4], 即 sum 作为 caller 返回值
14  leave
15  ret
```

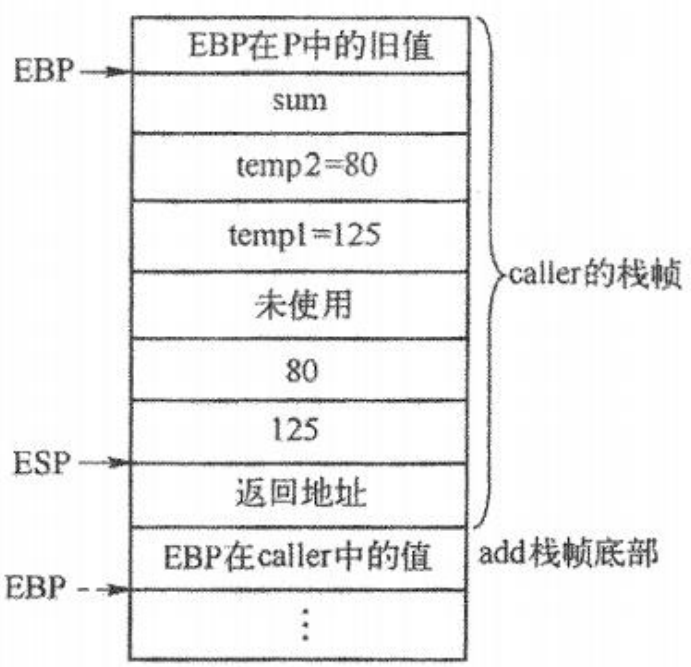


图 3.9 caller 和 add 的栈帧

13015 计算机系统原理【第3章考点解析】

考点16 值传递和地址传递

总体来说分为两种：

1）按值传递

2）按地址传递

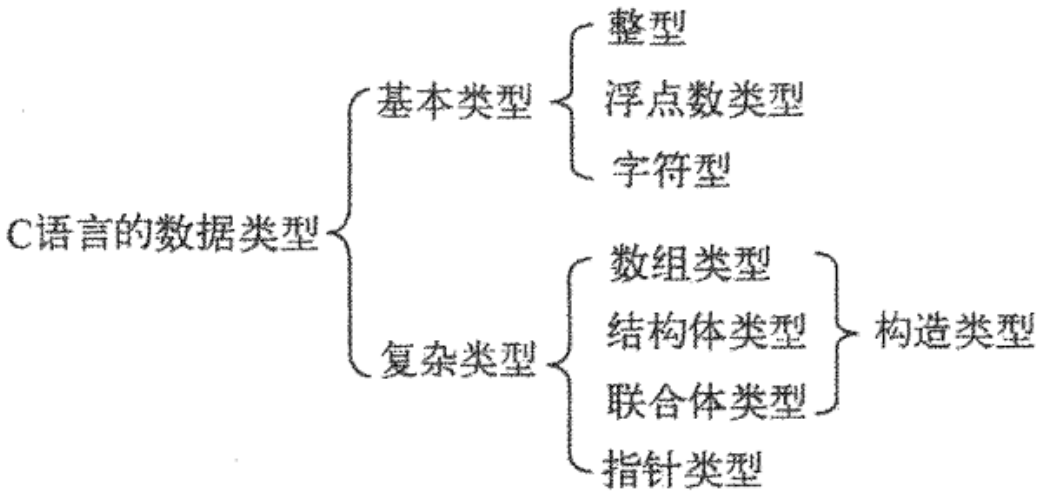


图 3.10　C 语言中的数据类型

13015 计算机系统原理【第3章考点解析】

考点16 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;
}
```

程序一的输出：

a=15　　b=22
a=22　　b=15

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;
}
```

程序二的输出：

a=15　　b=22
a=15　　b=22

图 3.11　按值传递参数和按地址传送参数的程序示例

13015 计算机系统原理【第3章考点解析】

考点16 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3章考点解析】

考点17 选择语句

例 3.9 以下是一个 C 语言函数：

```
1   int get_lowaddr_content(int *p1, int *p2){
2       if ( p1 > p2 )
3           return *p2;
4       else
5           return *p1;
6   }
```

```
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
```

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

13015 计算机系统原理【第3章考点解析】

考点18 循环语句

C 语言中循环结构有三种：

1）do~while 语句

```
do{
      loop_body_statement
} while (cond_expr);
```

2）while 语句

```
while (cond_expr)
      loop_body_statement
```

3）for 语句

```
for (begin_expr; cond_expr; update_expr)
      loop_body_statement
```

13015 计算机系统原理【第3章考点解析】

考点18 循环语句

例 3.10 一个 C 语言函数被 GCC 编译后得到的过程体对应的汇编代码如下。

```
1     movl    8(%ebp), %ebx
2     movl    $0, %eax
3     movl    $0, %ecx
4   .L12:
5     leal    (%eax,%eax), %edx
6     movl    %ebx, %eax
7     andl    $1, %eax
8     orl     %edx, %eax
9     shrl    %ebx
10    addl    $1, %ecx
11    cmpl    $32, %ecx
12    jne     .L12
```

该 C 语言函数的整体框架结构如下。

```
int func_test(unsigned x) {
    int result=0;
    int i;
    for ( ① ; ② ; ③ ) {
        ④
    }
    return result;
}
```

13015 计算机系统原理【第3章考点解析】

考点18 循环语句

例 3.10 一个 C 语言函数被 GCC 编译后得到的过程体对应的汇编代码如下。

```
1     movl    8(%ebp), %ebx          # ebp+8 放到ebx，R[ebx] = x
2     movl    $0, %eax               # R[eax] = 0 → result = 0
3     movl    $0, %ecx               # R[ecx] = i → i = 0 ①
4   .L12:
5     leal    (%eax,%eax), %edx      # R[edx] = result × 2 = result << 1
6     movl    %ebx, %eax             # R[eax] = x
7     andl    $1, %eax               # R[eax] = x & 1
8     orl     %edx, %eax             # R[eax] = (x & 1) | (result << 0x01) ④
9     shrl    %ebx                   # R[ebx] 右移一位 x >> 1 ④
10    addl    $1, %ecx               # R[ecx] = i + 1 = i++ ③
11    cmpl    $32, %ecx              # i != 32 ②
12    jne     .L12
```

result = (result<<1) | (x & 0x01)

教材有误

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

13015 计算机系统原理【第3章考点解析】

考点19 数组

数组可以**定义**为以下四种：

1）静态存储型（static）　　分配在静态数据区中

2）外部存储型（extern）　　分配在静态数据区中

3）自动存储型（auto）　　分配在栈中

4）全局静态区数组　　分配在静态数据区中

15. 数组可以定义为_____、_____、自动存储型或者定义为全局静态区数组。

【答案】：**静态存储型、外部存储型【2025年10月】**

13015 计算机系统原理【第3章考点解析】

考点20 数据的对齐方式

对齐的两条**核心规则**

1）成员对齐：

结构体中每个成员的偏移量（相对于首地址的距离），必须是该成员自身大小的整数倍

例如：short (2 字节)　　首地址必须是 2 的倍数

int (4 字节)　　首地址必须是 4 的倍数

2）整体对齐：

结构体的总大小，必须是其内部最大成员大小的整数倍（保证数组存储时每个元素都对齐）

例如：short (2 字节)

int (4 字节)

首地址必须是 (4 + 4) 的倍数

13015 计算机系统原理【第3章考点解析】

考点20 数据的对齐方式

例 3.11　假定 C 语言程序中定义了以下结构体数组。

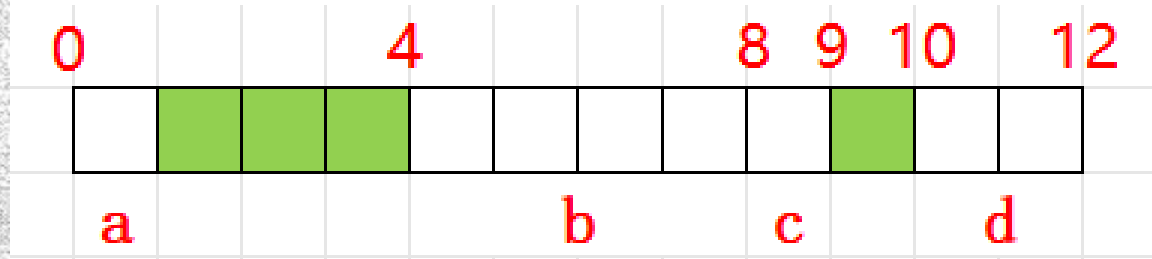
```
1   struct {
2       char   a;
3       int    b;
4       char   c;
5       short  d;
6   } record[100];
```

在对齐方式下该结构体数组 record 占用的存储空间为多少字节？每个成员的偏移量为多少？如何调整成员变量的顺序使得 record 占用空间最少？

13015 计算机系统原理【第3章考点解析】

考点20 数据的对齐方式

```
1  struct {
2      char  a;
3      int   b;
4      char  c;
5      short d;
6  } record[100];
```

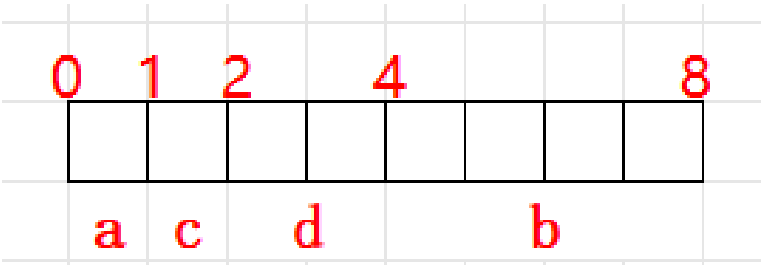



```
1  struct {
2      char  a;
3      char  c;
4      short d;
5      int   b;
6  } record[100];
```

该结构占用的存储空间都为 12 字节，因此，数组 record 共占 100×12=1200 字节。

为了使得 record 占用空间最少，可以按照从短→长（或从长→短）调整成员变量的声明顺序。

从短→长调整后的声明如下：



调整后每个数组元素占 8 字节，数组共占 100× 8=800 字节空间，比原来节省 400 字节。

13015 计算机系统原理【第3章考点解析】

考点21 x86-64的基本特点

1）比 IA-32 具有更多的通用寄存器个数。

2）比 IA-32 具有更长的通用寄存器位数，从 32 位扩展到 64 位。

3）字长从 32 位变为 64 位，因而逻辑地址从 32 位变为 64 位。

4）对于 long double 型数据，虽然还是采用与 IA-32 相同的 80 位扩展精度格式，但是，所分配的存储空间从 IA-32 的 12 字节大小扩展为 16 字节大小。

5）过程调用时，对于整型入口参数只有 6 个以内的情况，用通用寄存器而不是用栈来传递，因而，很多过程可以不访问栈，使得大多数情况下执行时间比 IA-32 代码更短。

6）128 位的 XMM 寄存器从原来的 8 个增加到 16 个，浮点操作采用基于 SSE 的面向 XMM 寄存器的指令集，浮点数存放在 128 位的 XMM 寄存器中。

13015 计算机系统原理【第3章考点解析】

考点22 x86-64的数据传送指令

汇编指令中指令助记符结尾处的 “**q**” 表示操作数长度为**四字（64 位）**。

在 x86-64 中，提供了一些在 IA-32 中没有的数据传送指令，如下：

1）**movabsq** 指令用于将一个 64 位立即数送到一个 64 位通用寄存器中

2）**movq** 指令用于传送一个 64 位的四字

3）**movsbq**、**movswq**、**movslq** 用于将源操作数进行符号扩展并传送到一个 64 位寄存器中

4）**movzbq**、**movzwq**用于将源操作数进行零扩展后传送到一个 64 位寄存器中

5）**leaq** 用于将有效地址加载到 64 位寄存器

6）**pushq** 和 **popq** 分别是四字压栈和四字出栈指令。

【重要】：在 x86-64 中，movl 指令的功能与在 IA-32 中不同，它在传送 32 位寄存器内容的同时，还会将**目的寄存器的高 32 位自动清 0**，因此，在 x86-64 中，movl 指令的功能相当于 movzlq 指令，因而在 x86-64 中不需要 movzlq 指令。【**movl = movzlq**】

13015 计算机系统原理【第3章考点解析】

例 3.12 以下是一个 C 语言函数，其功能是将类型为 source_type 的参数转换为 dest_type 类型的数据并返回。

```
dest_type convert(source_type x) {
    dest_type y = (dest_type) x;
    return y;
}
```

根据过程调用时的参数传递约定可知，x 存放在寄存器 RDI 对应的适合宽度的寄存器（如 RDI、EDI、DI 和 DIL）中，y 存放在 RAX 对应的寄存器（RAX、EAX、AX 或 AL）中，填写表 3.10 中的汇编指令，以实现 convert 函数中的赋值语句。

表 3.10 例 3.12 中 source_type 和 dest_type 的类型

source_type	dest_type	汇编指令
char	long	
int	long	
long	long	
long	int	
unsigned int	unsigned long	
unsigned long	unsigned int	
unsigned char	unsigned long	

13015 计算机系统原理【第3章考点解析】

考点22 x86-64的数据传送指令

根据过程调用时的参数传递约定可知，x 存放在寄存器 RDI 对应的适合宽度的寄存器（如 RDI、EDI、DI 和 DIL）中，y 存放在 RAX 对应的寄存器（RAX、EAX、AX 或 AL）中，

source_type	dest_type	汇编指令
char	long	movsbq %dil, %rax
int	long	movslq %edi, %rax
long	long	movq %rdi, %rax
long	int	movl %edi, %eax
unsigned int	unsigned long	movl %edi, %eax
unsigned long	unsigned int	movl %edi, %eax
unsigned char	unsigned long	movzbq %dil, %rax

movzlq %edi %rax

【重要】：在 x86-64 中，movl 指令的功能与在 IA-32 中不同，它在传送 32 位寄存器内容的同时，还会将**目的寄存器的高 32 位自动清 0**，因此，在 x86-64 中，movl 指令的功能相当于 movzlq 指令，因而在 x86-64 中不需要 movzlq 指令。【**movl = movzlq**】

13015 计算机系统原理【第3章考点解析】

考点23 x86-64的算术逻辑运算指令

例 3.13 以下是 C 语言赋值语句“x=a * b+c * d;”对应的 x86-64 汇编代码，已知 x、a、b、c 和 d 分别在寄存器 RAX、RDI、RSI、RDX 和 RCX 对应宽度的寄存器中。根据以下汇编代码，推测变量 x、a、b、c 和 d 的数据类型。

```
1    movslq   %ecx, %rcx
2    imulq    %rdx, %rcx
3    movsbl   %sil, %esi
4    imull    %edi, %esi
5    movslq   %esi, %rsi
6    leaq     (%rcx, %rsi), %rax
```

1）变量 d 是 int

2）变量 c 是 long

3）变量 b 是 char

4）变量 a 是 int

6）变量 x 是 long

R[RAX] = x

R[RDI] = a　　R[RSI] = b

R[RDX] = d　　R[RCX] = d

13015 计算机系统原理【第3章考点解析】

总结

1）**movabsq** 指令用于将一个 64 位立即数送到一个 64 位通用寄存器中

2）**movq** 指令用于传送一个 64 位的四字

3）**movsbq、movswq、movslq** 用于将源操作数进行符号扩展并传送到一个 64 位寄存器中

4）**movzbq、movzwq**用于将源操作数进行零扩展后传送到一个 64 位寄存器中

5）**leaq** 用于将有效地址加载到 64 位寄存器

6）**pushq** 和 **popq** 分别是四字压栈和四字出栈指令。

【重要】：在 x86-64 中，movl 指令的功能与在 IA-32 中不同，它在传送 32 位寄存器内容的同时，还会将**目的寄存器的高 32 位自动清 0**，因此，在 x86-64 中，movl 指令的功能相当于 movzlq 指令，因而在 x86-64 中不需要 movzlq 指令。【**movl = movzlq**】

谢 谢 大 家