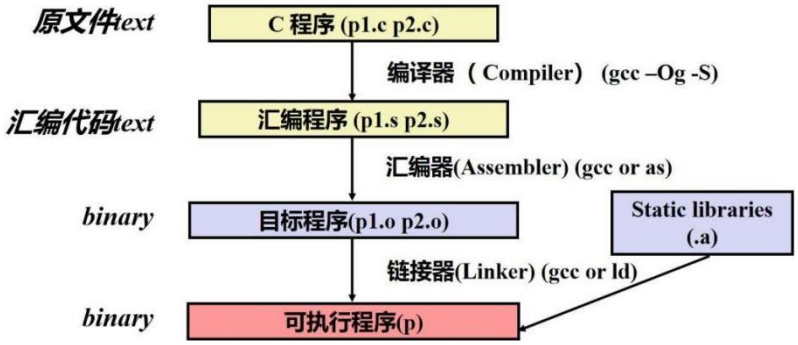
# 第三章速记清单

1. 程序编码：




2. 整数寄存器（教材P120，记住下表）：

| 63 | 31 | 15 | 7 | 0 | |
|---|---|---|---|---|---|
| %rax | %eax | %ax | %al | | Return value |
| %rbx | %ebx | %bx | %bl | | Callee saved |
| %rcx | %ecx | %cx | %cl | | 4th argument |
| %rdx | %edx | %dx | %dl | | 3rd argument |
| %rsi | %esi | %si | %sil | | 2nd argument |
| %rdi | %edi | %di | %dil | | 1st argument |
| %rbp | %ebp | %bp | %bpl | | Callee saved |
| %rsp | %esp | %sp | %spl | | Stack pointer |
| %r8 | %r8d | %r8w | %r8b | | 5th argument |
| %r9 | %r9d | %r9w | %r9b | | 6th argument |
| %r10 | %r10d | %r10w | %r10b | | Caller saved |
| %r11 | %r11d | %r11w | %r11b | | Caller saved |
| %r12 | %r12d | %r12w | %r12b | | Callee saved |
| %r13 | %r13d | %r13w | %r13b | | Callee saved |
| %r14 | %r14d | %r14w | %r14b | | Callee saved |
| %r15 | %r15d | %r15w | %r15b | | Callee saved |

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

**程序计数器（PC/%rip）**：给出将要执行的下一条指令地址。

**条件码寄存器**：保存最近执行指令的状态信息。

16个通用寄存器(如下表所示)：

**x86-64 的整数寄存器**

| | | | |
|---|---|---|---|
| %rax | %eax | %r8 | %r8d |
| %rbx | %ebx | %r9 | %r9d |
| %rcx | %ecx | %r10 | %r10d |
| %rdx | %edx | %r11 | %r11d |
| %rsi | %esi | %r12 | %r12d |
| %rdi | %edi | %r13 | %r13d |
| %rsp | %esp | %r14 | %r14d |
| %rbp | %ebp | %r15 | %r15d |

**64位**：%rax, %rbx, %rcx, %rdx, %rsi, %rdi, %rsp, %rbp, %rN

(这里N代表了8、9、10、11、12、13、14、15)

**32位**：%eax,%ebx,%ecx,%edx,%esi,%edi,%esp,%ebp,%rNd

**16位**： %ax, %bx, %cx, %dx, %si, %di, %sp, %bp, %rNw

**8位**： %al, %bl, %cl, %dl, %sil, %dil, %spl, %bpl, %rNb

**其中比较常见的寄存器功能包括：**

rax 常用作函数返回值；

函数调用时的前六个寄存器参数顺序：rdi，rsi，rdx，rcx，r8，r9；

rbp 用于指示栈帧；

rsp 用于指示栈顶；

其他参数保存在栈上。

3. （默认64位机上，本书规定1字=2字节）数据格式：

| 类型 | 后缀 | | 大小 |
|---|---|---|---|
| **char** | 字节（byte） | **b** | 1 字节 |

| | | | |
|---|---|---|---|
| **short** | 字（word） | **w** | 2 字节 |
| **int** | 双字（long word） | ***l*** | 4 字节 |
| **long** | 四字（quad） | **q** | 8 字节（quad） |
| char*，也包括所有指针类型 | 四字 | **q** | 8 字节 |
| **float** | 单精度 | **s** | 4字节 |
| **double** | 双精度 | ***l*** | 8字节 |

**4. 规定：生成四字节并以寄存器为目的的指令（比如 movl指令）会将高位 4 字节置 0，见教材P123。**

**一条指令不能同时以两个内存数据为操作数。**

5. 指令寻址方式（见教材P121图3-3）

| 类型 | 格式 | 操作数值 | 名称 |
|---|---|---|---|
| 立即数 | $*Imm* | *Imm* | 立即数寻址 |
| 寄存器 | $r_a$ | $R[r_a]$ | 寄存器寻址 |
| 存储器 | *Imm* | $M[Imm]$ | 绝对寻址 |
| 存储器 | $(r_a)$ | $M[R[r_a]]$ | 间接寻址 |
| 存储器 | $Imm(r_b)$ | $M[Imm+R[r_b]]$ | （基址 + 偏移量）寻址 |
| 存储器 | $(r_b, r_i)$ | $M[R[r_b]+R[r_i]]$ | 变址寻址 |
| 存储器 | $Imm(r_b, r_i)$ | $M[Imm+R[r_b]+R[r_i]]$ | 变址寻址 |
| 存储器 | $(,r_i, s)$ | $M[R[r_i] \cdot s]$ | 比例变址寻址 |
| 存储器 | $Imm(,r_i,s)$ | $M[Imm+R[r_i] \cdot s]$ | 比例变址寻址 |
| 存储器 | $(r_b, r_i, s)$ | $M[R[r_b]+R[r_i] \cdot s]$ | 比例变址寻址 |
| 存储器 | $Imm(r_b, r_i, s)$ | $M[Imm+R[r_b]+R[r_i] \cdot s]$ | 比例变址寻址 |

图 3-3 操作数格式。操作数可以表示立即数(常数)值、寄存器值或是来自内存的值。比例因子 $s$ 必须是 1、2、4 或者 8

6. 数据传送指令

**movx S D**：x 为 b、w、l、q，代表操作数的位数，将S的值赋给D，

说明：S（source）是源操作数，D（destination） 是目的操作数。

7. 零扩展传送指令

**movzbw, movzbl, movzwl, movzbq, movzwq S D**

说明：z 代表 0 扩展，b、w、l、q 分别代表操作数位数。

8. 符号扩展传送指令

**movsbw, movsbl, movswl, movsbq, movswq, movslq　S　D**

说明：s 代表符号扩展，b、w、l、q 分别代表操作数位数。

9. 出栈入栈

**popq, pushq　D**

说明：栈底的地址是高地址，栈自底向下扩展。%rsp 指向栈顶，%rbp指向栈帧的基地址。

10. 算数和逻辑指令

指令格式 ： 效果

**leaq　S　D**：加载地址，不改变状态标志，也没有实际引用内存

说明：leaq (load effective address，q代表4字节)。例如： leaq 7(%rdx, %rdx, 4), %rax，如果寄存器%rdx里存的数是x，则指令执行后，寄存器%rax的内容是7+5x）。

**INC　D**：D 自增

**DEC　D**：D 自减

**NEG　D**： D 取补码（按位取反末位加1，即**得到D的相反数**）

**NOT　D**： D 取非（按位取反）

**ADD　S　D**：D + S

**SUB　S　D**：D - S

**IMUL　S　D**：双操作数乘法 D * S

**XOR　S　D**：异或 D ^ S

**OR　S　D**： 或 D | S

**AND　S　D**：与 D & S

**SAL　k　D**：D 算数左移k 位

**SHL　k　D**：D 逻辑左移k 位

**SAR　k　D**：D 算数右移k 位（高位补符号位）

**SHR　k　D**：D 逻辑右移k 位（高位补 0）

NEG：把操作数按位取反加一 （可以用来求一个数的相反数）

NOT：把操作数按位取反

很明显可以看出区别：NEG比NOT指令多了一步“加一”操作

下面我们举个例子来清晰的说明下：

77用二进制为 100 1101B，正数，故在其前面加0，所以原码为：0100 1101

用NEG指令对它按位取反加一，结果为：1011 0011（即-77的补码）

这样，就用NEG求得了一个数的相反数

如果用NOT指令，则所求结果为：1011 0010（只是单纯的按位取反，即-78的补码）

特殊指令：

**imulq S：**单操作数的有符号乘法

**mulq S：**单操作数的无符号乘法

说明：这两个乘法指令以%rax 的值为被乘数，效果是 S * R[%rax] 结果的低 64 位放在%rax 中，高位放在%rdx 中。

**idivq S：**单操作数的有符号除法

**divq S：**单操作数的无符号除法

说明：这两个除法指令以%rdx 的值作为被除数高位，%rax 的值作为被除数的低位，效果是 R[%rdx]:R[%rax]/S 结果的商放在%rax，余数放在%rdx。

注意：以上指令全部加b、w、l、q 作为操作数位数。

11. 条件码（AF、PF不常用）

CF：进位标志，用来检测无符号溢出

OF：溢出标志，检测有符号溢出

ZF：零标志，最近操作结果是否为0

SF：符号标志，最近操作的结果是否为负数

12. 访问条件码

| 指令 | 同义词 | 作用 | 设置条件 |
|---|---|---|---|
| sete D | Setz | D <- ZF | 相等 / 结果为0 |
| setne D | setnz | D <- ~ZF | 不相等 / 结果不为0 |
| sets D | | D <- SF | 结果为负数 |
| setns D | | D <- ~SF | 结果为非负数 |
| setl D | setnge | D <- SF^OF | 小于 (符号数) |
| setle D | setng | D <- (SF^OF)\|ZF | 小于等于 (符号数) |
| setg D | setnle | D <- ~(SF^OF)&~ZF | 大于 (符号数) |
| setge D | setnl | D <- ~(SF^OF) | 大于等于 (符号数) |
| seta D | setnbe | D <- ~CF&~ZF | 大于 (无符号数) |
| setae D | setnb | D <- ~CF | 大于等于 (无符号数) |
| setb D | setnae | D <- CF | 小于 (无符号数) |
| setbe D | setna | D <- CF\|ZF | 小于等于 (无符号数) |

13. 比较和测试指令

**cmpx S D：** 操作等同与 sub，但只设置条件码而不改变寄存器的值。

说明：x为b、w、l、q，分别代表操作数位数。

**testx S D：** 操作等同于and，但只设置条件码而不改变寄存器的值。

说明：x为b、w、l、q，分别代表操作数位数。

14. 跳转指令

- **jX 指令: 根据条件码跳转**

| jX指令 | 条件 | 描述 |
|---|---|---|
| jmp | 1 | 无条件 |
| je | ZF | 相等 / 结果为0 |
| jne | ~ZF | 不相等 / 结果不为0 |
| js | SF | 结果为负数 |
| jns | ~SF | 结果为非负数 |
| jg | ~(SF^OF)&~ZF | 大于 (符号数) |
| jge | ~(SF^OF) | 大于等于 (符号数) |
| jl | (SF^OF) | 小于 (符号数) |
| jle | (SF^OF)\|ZF | 小于等于 (符号数) |
| ja | ~CF&~ZF | 大于 (无符号数) |
| jb | CF | 小于 (无符号数) |

**jX D：** 符合跳转条件就跳到目标代码位置执行。缺点：错误的条件分支预测会导致很高的代价。说明：其中X表示比较条件，例如l表示小于（有符号数比较）、ge表示大于等于（有符号数比较）。

15. 条件传送来实现条件分支

**cmovX S D：** 效果：满足条件才传送，根据状态码决定传送（一般放在cmpx后面，两个指令配合使用）。

说明：其中X表示比较条件，例如l表示小于（有符号数比较）、ge表示大于等于（有符号数比较）。

16. 转移控制指令

**call D：** 返回地址入栈，PC 设为D 的起始地址。

**ret：** 弹出返回地址，把PC 设为返回地址。

17. 溢出和进位的区别

**1）溢出标志OF和进位标志CF是两个意义不同的标志**

**2）进位标志表示无符号数运算结果是否超出范围，运算结果仍然正确；**

**3）溢出标志表示有符号数运算结果是否超出范围，运算结果已经不正确。**