程序的机器级表示：基本操作

Machine-Level Programming : Basic Operations

本章内容

Topic

- x86寄存器

  x86 registers

- 数据移动指令

  Move

- 算术、逻辑运算指令

  Arithmetic & logical operations

x86寄存器

x86 registers

- x86-64 寄存器
  x86-64 registers
  - 每个寄存器的低4/2/1字节都有唯一的标识
    Can reference low-order 4 bytes (also low-order 1 & 2 bytes)

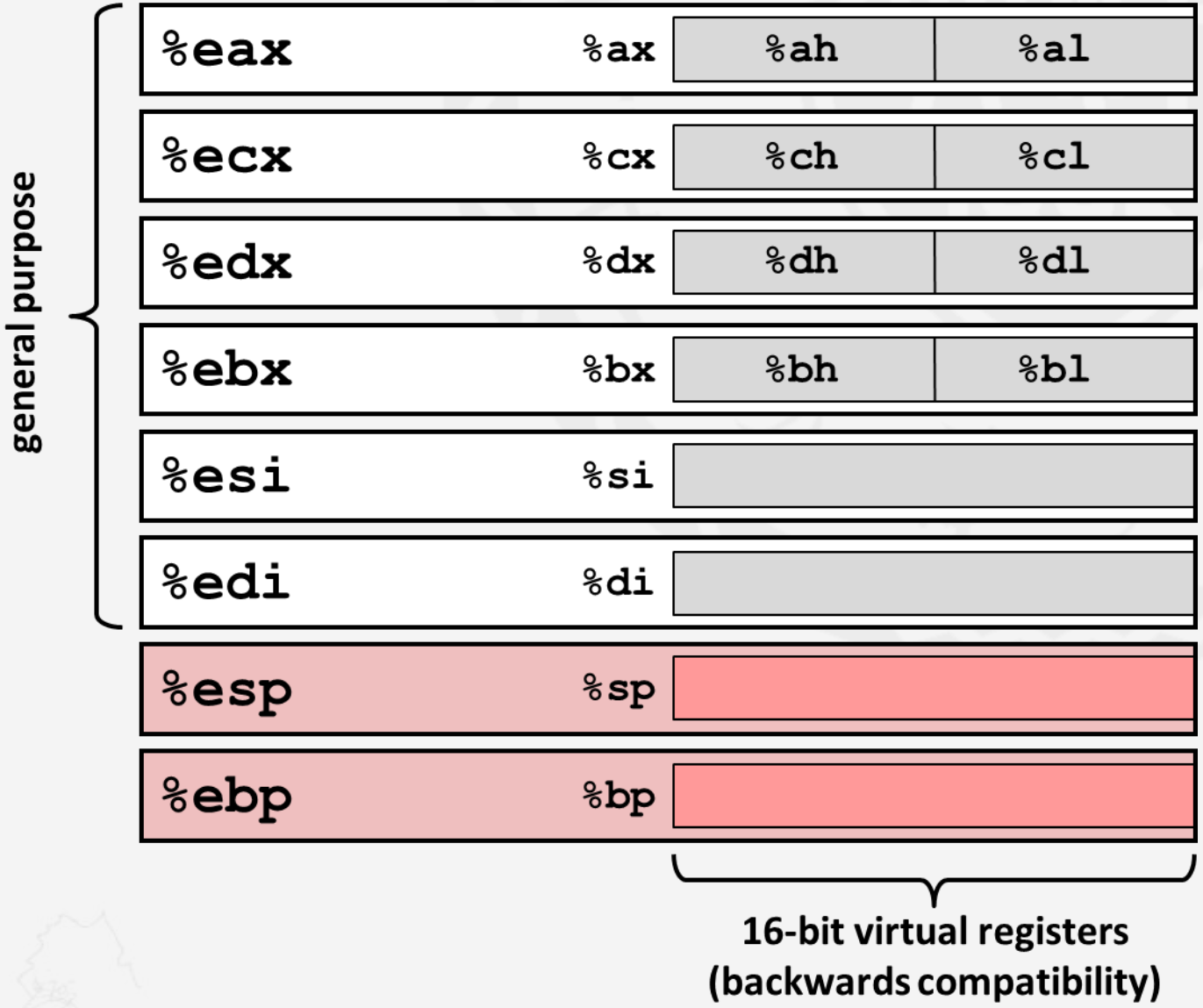
x86寄存器

x86 registers

x86寄存器

x86 registers

- IA32 (x86-32) 寄存器
  IA32 (x86-32) registers



本章内容

Topic

- x86寄存器

  x86 registers

- 数据移动指令

  Move

- 算术、逻辑运算指令

  Arithmetic & logical operations

数据移动指令

Move

汇编语言格式
Assembly Code Format

[label :] [opcode] [operand 1] [, operand 2]

[标号：] [操作码] [操作数1] [，操作数2]

ATT assembly:

```
    l1:  movq  $5  ,  %rax
         addq  $-16,  (%rax)
```

Intel assembly:

```
    l1:  mov  rax , 5
         add  QWORD PTR[rax], -16
```

数据移动指令

Move

数据移动指令
Moving Data

`movq Source, Dest`

- 操作数类型
  Operand Types
  - **立即数：** 整数常量
    Immediate: Constant integer data
    - 例如： **`$0x400, $-533`**
      Example: **`$0x400, $-533`**
    - 和C语言中的常数类似，但需要加前缀 **`$`**
      Like C constant, but prefixed with '**`$`**'
    - 被编码为1、2、4或8个字节
      Encoded with 1, 2, 4 or 8 bytes
  - **寄存器：** 十六个整数寄存器之一
    Register: One of 16 integer registers
    - 例如： **`%rax, %r13`**
      Example: **`%rax, %r13`**
    - **`%rsp`**有特殊用途，通常不使用
      But **`%rsp`** reserved for special use
    - 其他寄存器在一些特殊的指令中也会有特殊用途
      Others have special uses for particular instructions
  - **存储器：** 指向的内存中8个连续字节，由寄存器给出地址
    Memory: 8 consecutive bytes of memory at address given by register
    - 一个简单的例子： **`(%rax)`**
      Simplest example: **`(%rax)`**
    - 有很多其他的"寻址模式"
      Various other "address modes"

数据移动指令

Move

movq 指令操作数的组合
movq Operand Combinations

	源操作数 Src	目标操作数 Dest	Src, Dest	等价C语言 C Analog
movq	Imm	Reg	movq $0x4, %rax	temp = 0x4;
		Mem	movq $-147, (%rax)	*p = -147;
	Reg	Reg	movq %rax, %rdx	temp2 = temp1;
		Mem	movq %rax, (%rdx)	*p = temp;
	Mem	Reg	movq (%rax), %rdx	temp = *p;

数据移动指令

Move

数据格式
Data Formats

C语言类型声明 C declaration	数据类型 Data type	操作码后缀 Opcode suffix	大小 Size(bytes)
char	Byte	b	1
short	Word	w	2
int	Double Word	l	4
long	Quad Word	q	8
char *	Quad Word	q	8
float	Single precision	s	4
Double	Double precision	l	8

数据移动指令

Move

几种简单的存储器寻址模式
Simple Memory Addressing Modes

- 间接寻址 (R) Mem[Reg[R]]
  Normal (R) Mem[Reg[R]]
  - 寄存器 **R** 指向了存储器的地址
    Register **R** specifies memory address
  - 和C语言中的指针作用相同
    Pointer dereferencing in C

```
movq (%rcx),%rax
```

- 基地址+偏移量寻址 D(R) Mem[Reg[R]+D]
  Displacement D(R) Mem[Reg[R]+D]
  - 寄存器 **R** 指定了存储器区域的开始位置
    Register **R** specifies start of memory region
  - 常数 **D** 是偏移量
    Constant displacement D specifies offset

```
movq 8(%rbp),%rdx
```

数据移动指令

Move

举例：简单寻址模式
Example: Simple Memory Addressing

```
void swap (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
   movq    (%rdi), %rax
   movq    (%rsi), %rdx
   movq    %rdx, (%rdi)
   movq    %rax, (%rsi)
   ret
```

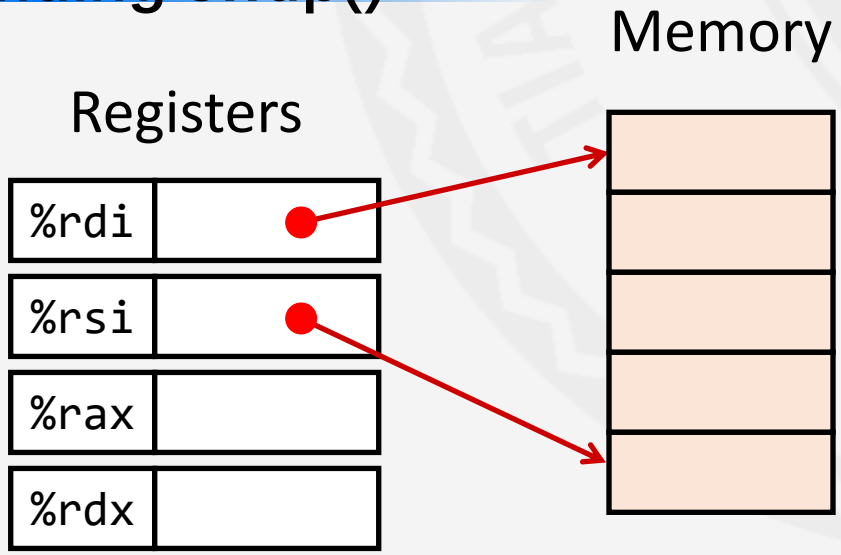
数据移动指令

Move

swap() 分析
Understanding swap()

```
void swap
   (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Value
%rdi	xp
%rsi	yp
%rax	t0
%rdx	t1

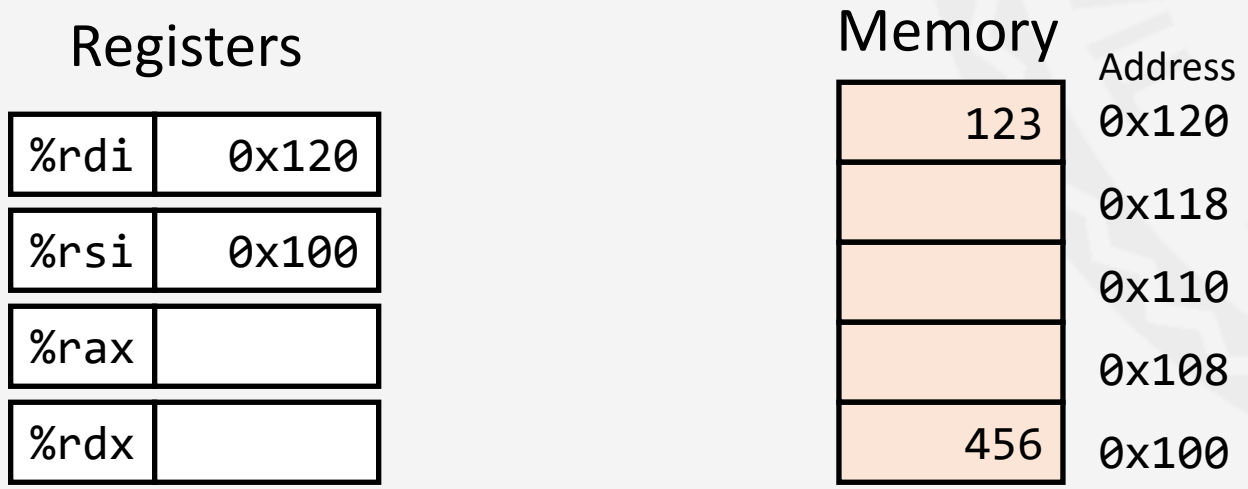



```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

数据移动指令

Move

swap() 分析
Understanding swap()

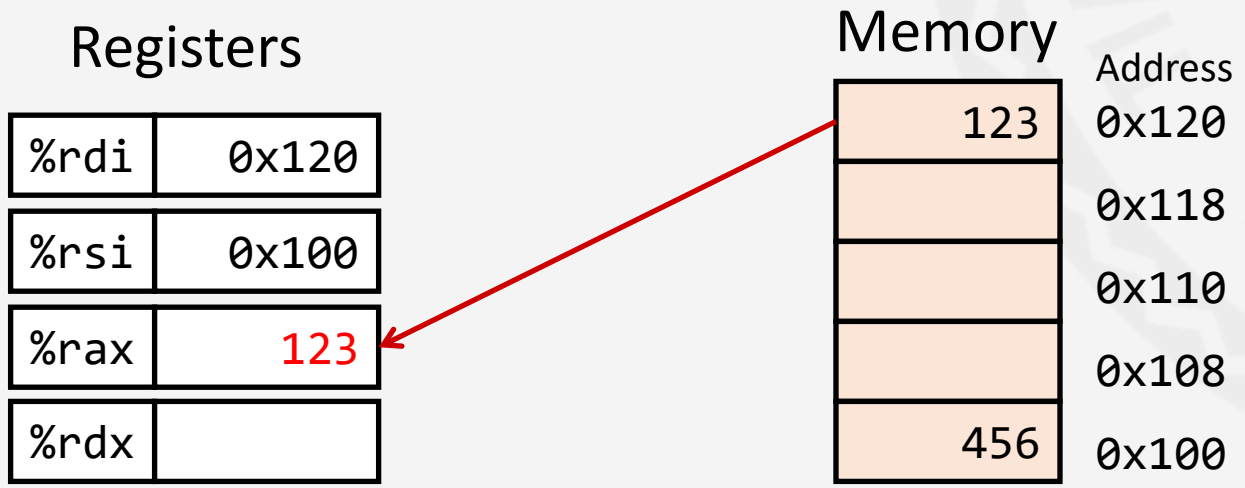



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

数据移动指令

Move

swap() 分析
Understanding swap()

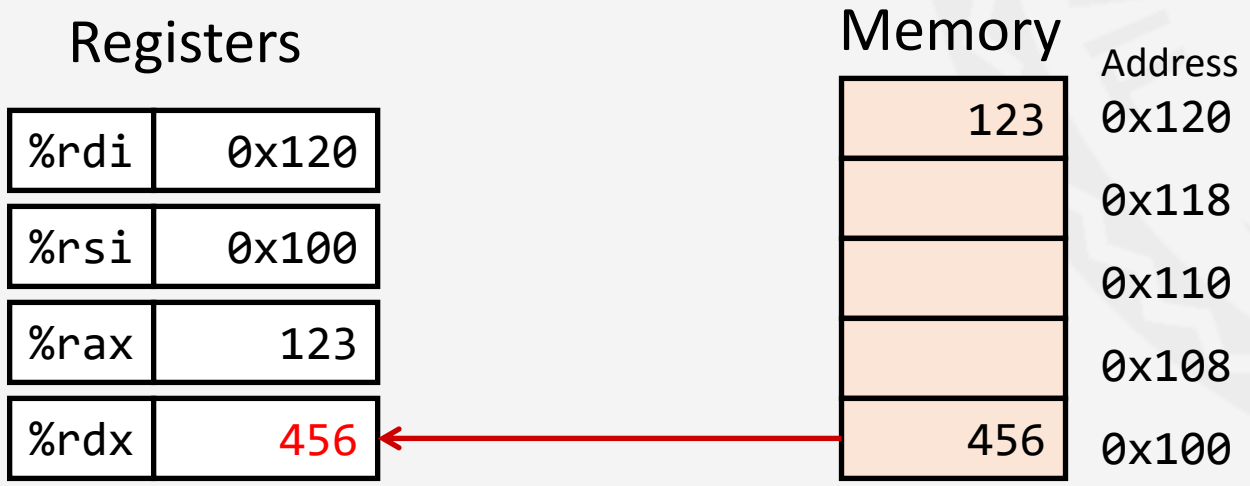



```
swap:
    movq    (%rdi), %rax  # t0 = *xp
    movq    (%rsi), %rdx  # t1 = *yp
    movq    %rdx, (%rdi)  # *xp = t1
    movq    %rax, (%rsi)  # *yp = t0
    ret
```

数据移动指令

Move

swap() 分析
Understanding swap()

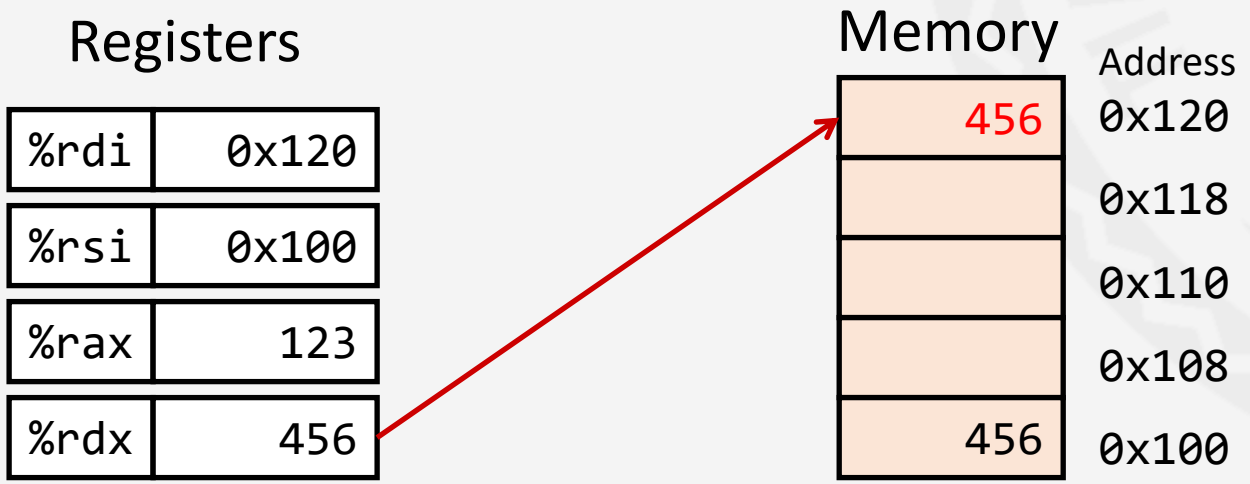



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

数据移动指令

Move

swap() 分析
Understanding swap()

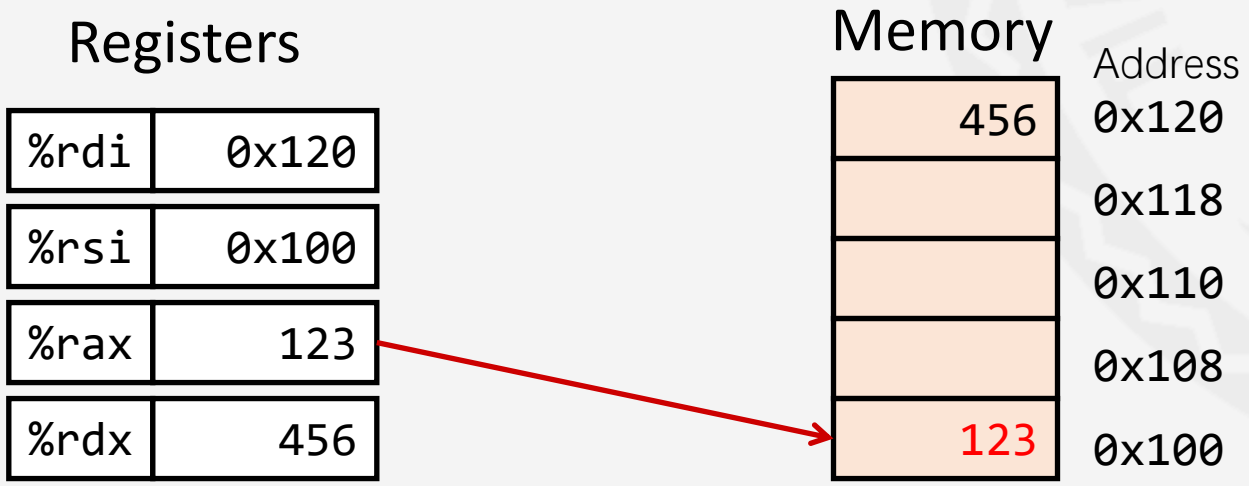



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

数据移动指令

Move

swap() 分析
Understanding swap()




```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

数据移动指令

Move

完整的存储器寻址模式
Complete Memory Addressing Modes

- 一般形式 Most General Form

  $D(R_b, R_i, S)$ $Mem[Reg[R_b] + S * Reg[R_i] + D]$

  - D：常数偏移量，可以为 1,2,4或8字节整数
    D: Constant “displacement” 1, 2, 4 or 8 bytes
  - R_b：基地址寄存器16个寄存器之一
    R_b: Base register: Any of 16 integer registers
  - R_i：变址寄存器，除%rsp外的其他寄存器
    Ri: Index register: Any, except for %rsp
  - S：比例因子：可以为1,2,4或8 （为什么是这些数字？）
    S: Scale: 1, 2, 4, or 8 (why these numbers?)

一些特殊形式

Special Cases

(R_b, R_i)	$Mem[Reg[R_b] + Reg[R_i]]$
$D(R_b, R_i)$	$Mem[Reg[R_b] + Reg[R_i] + D]$
(R_b, R_i, S)	$Mem[Reg[R_b] + S * Reg[R_i]]$

数据移动指令

Move

小练习：地址计算
Exercise: Computing Address

%rdx	0xf000
%rcx	0x0100

Expression	Address Computation	Address
0x8(%rdx)	0xf000 + 0x8	0xf008
(%rdx,%rcx)	0xf000 + 0x100	0xf100
(%rdx,%rcx,4)	0xf000 + 4*0x100	0xf400
0x80(,%rdx,2)	2*0xf000 + 0x80	0x1e080

本章内容

Topic

- x86寄存器

  x86 registers

- 数据移动指令

  Move

- 算术、逻辑运算指令

  Arithmetic & logical operations

算术、逻辑运算指令

Arithmetic & logical operations

地址计算指令
Address Computation Instruction

`leaq Src, Dst`

- Src是寻址模式表达式
  Src is address mode expression
- 将表达式计算的地址写入Dst
  Set Dst to address denoted by expression

- 用途
  Uses
  - 计算地址（计算过程中不需要引用存储器）
    Computing addresses without a memory reference
    - `p = &x[i]`
  - 计算模式为 `x + k*y` 的表达式 （普通的算术运算）
    Computing arithmetic expressions of the form x + k*y
    - `k = 1, 2, 4, or 8`

```
long m12(long x)
{
  return x*12;
}
```

编译后的汇编指令
Converted to ASM by compiler:

```
leaq (%rdi,%rdi,2), %rax # t <- x+x*2
salq $2, %rax            # return t<<2
```

算术、逻辑运算指令

Arithmetic & logical operations

一些算术运算指令
Some Arithmetic Operations

- 两操作数指令（双目运算）
  Two Operands Instructions
- 注意操作数的顺序
  Watch out for argument order!
- 有符号数和无符号数指令没有区别（为什么？）
  No distinction between signed and unsigned int (why?)

Format		Computation	
`addq`	*Src, Dest*	Dest = Dest + Src	
`subq`	*Src, Dest*	Dest = Dest – Src	
`imulq`	*Src, Dest*	Dest = Dest * Src	
`salq`	*Src, Dest*	Dest = Dest << Src	***Also called shlq***
`sarq`	*Src, Dest*	Dest = Dest >> Src	***Arithmetic***
`shrq`	*Src, Dest*	Dest = Dest >> Src	***Logical***
`xorq`	*Src, Dest*	Dest = Dest ^ Src	
`andq`	*Src, Dest*	Dest = Dest & Src	
`orq`	*Src, Dest*	Dest = Dest | Src	

算术、逻辑运算指令

Arithmetic & logical operations

一些算术运算指令
Some Arithmetic Operations

- 单操作数指令（单目运算）
  One Operand Instructions
- 更多的指令介绍见教材
  See book for more instructions

Format	Computation
`incq` *Dest*	*Dest = Dest + 1*
`decq` *Dest*	*Dest = Dest – 1*
`negq` *Dest*	*Dest = – Dest*
`notq` *Dest*	*Dest = ~Dest*

算术、逻辑运算指令

Arithmetic & logical operations

举例：算术运算
Arithmetic Expression Example

```
long arith
(long x, long y, long z)
{
  long t1 = x+y;
  long t2 = z+t1;
  long t3 = x+4;
  long t4 = y * 48;
  long t5 = t3 + t4;
  long rval = t2 * t5;
  return rval;
}
```

```
arith:
  leaq    (%rdi,%rsi), %rax
  addq    %rdx, %rax
  leaq    (%rsi,%rsi,2), %rdx
  salq    $4, %rdx
  leaq    4(%rdi,%rdx), %rcx
  imulq   %rcx, %rax
  ret
```

- 需要关注的指令
  Interesting Instructions
  - **leaq**: 地址计算
    leaq: address computation
  - **salq**: 左移
    salq: shift
  - **imulq**: 乘法
    imulq: multiplication
    - 只出现了一次
      But, only used once

算术、逻辑运算指令

Arithmetic & logical operations

分析：算术运算示例
Understanding Arithmetic Expression Example

```
long arith
(long x, long y, long z)
{
  long t1 = x+y;
  long t2 = z+t1;
  long t3 = x+4;
  long t4 = y * 48;
  long t5 = t3 + t4;
  long rval = t2 * t5;
  return rval;
}
```

```
arith:
   leaq    (%rdi,%rsi), %rax
   addq    %rdx, %rax
   leaq    (%rsi,%rsi,2), %rdx
   salq    $4, %rdx
   leaq    4(%rdi,%rdx), %rcx
   imulq   %rcx, %rax
   ret
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rdx	Argument z
%rax	t1, t2, rval
%rdx	t4
%rcx	t5

- 指令顺序和C语言语句顺序不同
  Instructions in different order from C code
- 一些表达式需要由多条指令组合实现
  Some expressions require multiple instructions
- 一些指令可以实现多个表达式的功能
  Some instructions cover multiple expressions
- (x+y+z)*(x+4+48*y)可以得到相同的汇编代码
  Get exact same code when compile:
  (x+y+z)*(x+4+48*y)

另一个例子
Another Example

```
long logical(long x, long y)
{
    long t1 = x^y;
    long t2 = t1 >> 17;
    long mask = (1<<13) - 7;
    long rval = t2 & mask;
    return rval;
}
```

```
logical:
    movq    %rdi, %rax
    xorq    %rsi, %rax
    sarq    $17,  %rax
    andl    $8185, %eax
    ret
```

$2^{13} = 8192, 2^{13} - 7 = 8185$