



x86

ARITHMETIC AND LOGICAL OPERATIONS

LOADING AN ADDRESS

Load Effective Address (Quad)

`leaq S, D` ($D \leftarrow \&S$)

- ▶ Loads the address of S in D, not the contents
 - ▶ Trivial example:
 - ▶ `leaq (%rax), %rdx`
 - ▶ Equivalent to: `movq %rax, %rdx`
- ▶ Destination must be a register
- ▶ Used to compute addresses without a memory reference
 - ▶ e.g., translation of `p = &x[i];`

LOADING AN ADDRESS

`leaq S, D` ($D \leftarrow \&S$)

- ▶ Commonly used by compiler to do simple arithmetic
 - ▷ If $\%rdx = x$,
 - ▷ `leaq 7(%rdx, %rdx, 4), %rdx` $\rightarrow 5x + 7$
 - ▷ Multiply and add all in one instruction
- ▶ Example:

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



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

PRACTICE PROBLEM 3.6 WALKTHROUGH

`%rax = x`

`%rcx = y`

Expression	Result in %rdx
<code>leaq 6(%rax), %rdx</code>	
<code>leaq (%rax, %rcx), %rdx</code>	
<code>leaq (%rax, %rcx, 4), %rdx</code>	
<code>leaq 7(%rax, %rax, 8), %rdx</code>	
<code>leaq 0xA(, %rcx, 4), %rdx</code>	
<code>leaq 9(%rax, %rcx, 2), %rdx</code>	

PRACTICE PROBLEM 3.6 WALKTHROUGH

`%rax = x`

`%rcx = y`

Expression	Result in %rdx
<code>leaq 6(%rax), %rdx</code>	$x+6$
<code>leaq (%rax, %rcx), %rdx</code>	$x+y$
<code>leaq (%rax, %rcx, 4), %rdx</code>	$x+4y$
<code>leaq 7(%rax, %rax, 8), %rdx</code>	$9x+7$
<code>leaq 0xA(, %rcx, 4), %rdx</code>	$4y+10$
<code>leaq 9(%rax, %rcx, 2), %rdx</code>	$x+2y+9$

TWO OPERAND ARITHMETIC OPERATIONS

Accumulated operation

- ▶ Second operand is both a source and destination
- ▶ A bit like C operators '+=', '-=', etc.
- ▶ Max shift is 64 bits, so k is either an immediate byte, or register (e.g. %cl where %cl is byte 0 of register %rcx)

Format	Computation	Notes
addq S, D	$D = D + S$	
subq S, D	$D = D - S$	
imulq S, D	$D = D * S$	
salq S, D	$D = D \ll S$	Also known as "shlq"
sarq S, D	$D = D \gg S$	Arithmetic Shift, Sign Extend
shrq S, D	$D = D \gg S$	Logical Shift, Zero Fill
xorq S, D	$D = D \wedge S$	
andq S, D	$D = D \& S$	
orq S, D	$D = D S$	

ONE OPERAND ARITHMETIC OPERATIONS

Format	Computation	Notes
incq D	$D = D + 1$	
decq D	$D = D - 1$	
negq D	$D = -D$	Two's Complement Negation
notq D	$D = \sim D$	Bitwise Negation

PRACTICE PROBLEM 3.8

Address	Value
0x100	0xFF
0x108	0xAB
0x110	0x13
0x118	0x11

Register	Value
%rax	0x100
%rcx	0x1
%rdx	0x3

Instruction	Destination	Result
addq %rcx, (%rax)		
subq %rdx, 8(%rax)		
imulq \$16, (%rax, %rdx, 8)		
incq 16(%rax)		
decq %rcx		
subq %rdx, %rax		

PRACTICE PROBLEM 3.8

Address	Value
0x100	0xFF
0x108	0xAB
0x110	0x13
0x118	0x11

Register	Value
%rax	0x100
%rcx	0x1
%rdx	0x3

Instruction	Destination	Result
addq %rcx, (%rax)	0x100	0x100
subq %rdx, 8(%rax)	0x108	0xA8
imulq \$16, (%rax, %rdx, 8)	0x118	0x110
incq 16(%rax)	0x110	0x14
decq %rcx	%rcx	0x0
subq %rdx, %rax	%rax	0xFD

PRACTICE PROBLEM 3.8

```
long shift_left4_rightn(long x, long n)
{
    x <<= 4;
    x >>= n;
    return x;
}
```

`_shift_left4_rightn:` (given that `n` is stored in `%rcx`)

```
movq    %rdi, %rax    ; get x
salq   $4, %rax     ; x <<= 4;
movq    %rsi, %rcx    ; get n
sarq   $cl, %rax    ; x >>= n;
ret
```

ARITHMETIC EXPRESSION EXAMPLE

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

```
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;
}
```

Compiler trick to generate efficient code

```
arith:
    leaq    (%rdi,%rsi), %rax    # t1
    addq    %rdx, %rax          # t2
    leaq    (%rsi,%rsi,2), %rdx
    salq    $4, %rdx            # t4
    leaq    4(%rdi,%rdx), %rcx  # t5
    imulq   %rcx, %rax          # rval
    ret
```

PRACTICE PROBLEM 3.10

What does this instruction do?

```
xorq    %rdx, %rdx
```

```
  00101010  
^ 00101010  
-----  
  00000000
```

Zeros out a
register

How might it be different from this instruction?

```
movq    $0, %rdx
```

3-byte instruction vs 7-byte instruction
Null bytes encoded in instruction

EXAM PRACTICE

Chapter 3 Problems (Part 1)

- ▶ 3.1 x86 operands
- ▶ 3.2, 3.3 instruction operand sizes
- ▶ 3.4 instruction construction
- ▶ 3.5 disassemble to C
- ▶ 3.6 leaq
- ▶ 3.7 leaq disassembly
- ▶ 3.8 operations in x86
- ▶ 3.9 fill in x86 from C
- ▶ 3.10 fill in C from x86
- ▶ 3.11 xorq