



Machine-Level Programming I: Basics - Arithmetic & logical operations

These slides adapted from materials provided by the textbook authors.

Machine Programming I: Basics

- History of Intel processors and architectures
- C, assembly, machine code
- Assembly Basics: Registers, operands, move
- **Arithmetic & logical operations**

Some Arithmetic Operations

■ Two Operand Instructions:

Format	Computation		
<code>addq</code>	Src, Dest	$\text{Dest} = \text{Dest} + \text{Src}$	
<code>subq</code>	Src, Dest	$\text{Dest} = \text{Dest} - \text{Src}$	
<code>imulq</code>	Src, Dest	$\text{Dest} = \text{Dest} * \text{Src}$	
<code>salq</code>	Src, Dest	$\text{Dest} = \text{Dest} \ll \text{Src}$	Also called <code>shlq</code>
<code>sarq</code>	Src, Dest	$\text{Dest} = \text{Dest} \gg \text{Src}$	Arithmetic
<code>shrq</code>	Src, Dest	$\text{Dest} = \text{Dest} \gg \text{Src}$	Logical
<code>xorq</code>	Src, Dest	$\text{Dest} = \text{Dest} \wedge \text{Src}$	
<code>andq</code>	Src, Dest	$\text{Dest} = \text{Dest} \& \text{Src}$	
<code>orq</code>	Src, Dest	$\text{Dest} = \text{Dest} \text{Src}$	

■ Watch out for argument order!

■ No distinction between signed and unsigned int (why?)

Some Arithmetic Operations

■ One Operand Instructions

<code>incq</code>	<code>Dest</code>	$\text{Dest} = \text{Dest} + 1$
<code>decq</code>	<code>Dest</code>	$\text{Dest} = \text{Dest} - 1$
<code>negq</code>	<code>Dest</code>	$\text{Dest} = -\text{Dest}$
<code>notq</code>	<code>Dest</code>	$\text{Dest} = \sim\text{Dest}$

■ See book for more instructions

Address Computation Instruction

■ `leaq Src, Dst`

- Src is address mode expression
- Set Dst to address denoted by expression



**Dumbest Op
Name In
The Universe**

■ Uses

- Computing addresses without a memory reference
 - E.g., translation of `p = &x[i];`
- Computing arithmetic expressions of the form $x + k*y$
 - $k = 1, 2, 4, \text{ or } 8$

■ Example

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

Converted to ASM by compiler:

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

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
```

Interesting Instructions

- **leaq**: address computation
- **salq**: shift
- **imulq**: multiplication
 - But, only used once

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    # 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
```

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

Machine Programming I: Summary

- **History of Intel processors and architectures**
 - Evolutionary design leads to many quirks and artifacts
- **C, assembly, machine code**
 - New forms of visible state: program counter, registers, ...
 - Compiler must transform statements, expressions, procedures into low-level instruction sequences
- **Assembly Basics: Registers, operands, move**
 - The x86-64 move instructions cover wide range of data movement forms
- **Arithmetic**
 - C compiler will figure out different instruction combinations to carry out computation