

Machine-Level Representation of Programs II

Hojoon Lee

Systems Security Lab @ SKKU



What Can be Disassembled?

```

Legend: Modified register | Code | Heap | Stack | String |
xrax: 0x1c
xrbx: 0x0
rcx: 0x00007fffffffd130 + 0x00007fffffffa48f + "LC_TERMINAL_VERSION=3.1.0beta2"
rcsi: 0x0
rsi: 0x00007fffffffd170 + 0x0000000000000001
rsi2: 0x0
rsi3: 0x00007fffffffd728 + 0x0000000000000000
rsi4: 0x00007fffffffd710 + 0x0000555555554000 + 0x00010102464c457f
rsip: 0x0000555555553970 + xor ebp, ebp
rsb: 0x0
rsb2: 0x0
rsb3: 0x0f
rsb4: 0x0
rsi5: 0x0000555555553970 + xor ebp, ebp
rsi6: 0x00007fffffffd170 + 0x0000000000000001
rsi7: 0x0
rsi8: 0x0
rsi9: 0x0
rsi10: 0x0
rsi11: 0x0
rsi12: 0x0000555555553970 + xor ebp, ebp
rsi13: 0x00007fffffffd170 + 0x0000000000000001
rsi14: 0x0
rsi15: 0x0
rsi16: 0x0
rsi17: 0x0
rsi18: 0x0
rsi19: 0x0
rsi20: 0x0
rsi21: 0x0
rsi22: 0x0
rsi23: 0x0
rsi24: 0x0
rsi25: 0x0
rsi26: 0x0
rsi27: 0x0
rsi28: 0x0
rsi29: 0x0
rsi30: 0x0
rsi31: 0x0
rsi32: 0x0
rsi33: 0x0
rsi34: 0x0
rsi35: 0x0
rsi36: 0x0
rsi37: 0x0
rsi38: 0x0
rsi39: 0x0
rsi40: 0x0
rsi41: 0x0
rsi42: 0x0
rsi43: 0x0
rsi44: 0x0
rsi45: 0x0
rsi46: 0x0
rsi47: 0x0
rsi48: 0x0
rsi49: 0x0
rsi50: 0x0
rsi51: 0x0
rsi52: 0x0
rsi53: 0x0
rsi54: 0x0
rsi55: 0x0
rsi56: 0x0
rsi57: 0x0
rsi58: 0x0
rsi59: 0x0
rsi60: 0x0
rsi61: 0x0
rsi62: 0x0
rsi63: 0x0
rsi64: 0x0
rsi65: 0x0
rsi66: 0x0
rsi67: 0x0
rsi68: 0x0
rsi69: 0x0
rsi70: 0x0
rsi71: 0x0
rsi72: 0x0
rsi73: 0x0
rsi74: 0x0
rsi75: 0x0
rsi76: 0x0
rsi77: 0x0
rsi78: 0x0
rsi79: 0x0
rsi80: 0x0
rsi81: 0x0
rsi82: 0x0
rsi83: 0x0
rsi84: 0x0
rsi85: 0x0
rsi86: 0x0
rsi87: 0x0
rsi88: 0x0
rsi89: 0x0
rsi90: 0x0
rsi91: 0x0
rsi92: 0x0
rsi93: 0x0
rsi94: 0x0
rsi95: 0x0
rsi96: 0x0
rsi97: 0x0
rsi98: 0x0
rsi99: 0x0
rsi100: 0x0
rsi101: 0x0
rsi102: 0x0
rsi103: 0x0
rsi104: 0x0
rsi105: 0x0
rsi106: 0x0
rsi107: 0x0
rsi108: 0x0
rsi109: 0x0
rsi110: 0x0
rsi111: 0x0
rsi112: 0x0
rsi113: 0x0
rsi114: 0x0
rsi115: 0x0
rsi116: 0x0
rsi117: 0x0
rsi118: 0x0
rsi119: 0x0
rsi120: 0x0
rsi121: 0x0
rsi122: 0x0
rsi123: 0x0
rsi124: 0x0
rsi125: 0x0
rsi126: 0x0
rsi127: 0x0
rsi128: 0x0
rsi129: 0x0
rsi130: 0x0
rsi131: 0x0
rsi132: 0x0
rsi133: 0x0
rsi134: 0x0
rsi135: 0x0
rsi136: 0x0
rsi137: 0x0
rsi138: 0x0
rsi139: 0x0
rsi140: 0x0
rsi141: 0x0
rsi142: 0x0
rsi143: 0x0
rsi144: 0x0
rsi145: 0x0
rsi146: 0x0
rsi147: 0x0
rsi148: 0x0
rsi149: 0x0
rsi150: 0x0
rsi151: 0x0
rsi152: 0x0
rsi153: 0x0
rsi154: 0x0
rsi155: 0x0
rsi156: 0x0
rsi157: 0x0
rsi158: 0x0
rsi159: 0x0
rsi160: 0x0
rsi161: 0x0
rsi162: 0x0
rsi163: 0x0
rsi164: 0x0
rsi165: 0x0
rsi166: 0x0
rsi167: 0x0
rsi168: 0x0
rsi169: 0x0
rsi170: 0x0
rsi171: 0x0
rsi172: 0x0
rsi173: 0x0
rsi174: 0x0
rsi175: 0x0
rsi176: 0x0
rsi177: 0x0
rsi178: 0x0
rsi179: 0x0
rsi180: 0x0
rsi181: 0x0
rsi182: 0x0
rsi183: 0x0
rsi184: 0x0
rsi185: 0x0
rsi186: 0x0
rsi187: 0x0
rsi188: 0x0
rsi189: 0x0
rsi190: 0x0
rsi191: 0x0
rsi192: 0x0
rsi193: 0x0
rsi194: 0x0
rsi195: 0x0
rsi196: 0x0
rsi197: 0x0
rsi198: 0x0
rsi199: 0x0
rsi200: 0x0
rsi201: 0x0
rsi202: 0x0
rsi203: 0x0
rsi204: 0x0
rsi205: 0x0
rsi206: 0x0
rsi207: 0x0
rsi208: 0x0
rsi209: 0x0
rsi210: 0x0
rsi211: 0x0
rsi212: 0x0
rsi213: 0x0
rsi214: 0x0
rsi215: 0x0
rsi216: 0x0
rsi217: 0x0
rsi218: 0x0
rsi219: 0x0
rsi220: 0x0
rsi221: 0x0
rsi222: 0x0
rsi223: 0x0
rsi224: 0x0
rsi225: 0x0
rsi226: 0x0
rsi227: 0x0
rsi228: 0x0
rsi229: 0x0
rsi230: 0x0
rsi231: 0x0
rsi232: 0x0
rsi233: 0x0
rsi234: 0x0
rsi235: 0x0
rsi236: 0x0
rsi237: 0x0
rsi238: 0x0
rsi239: 0x0
rsi240: 0x0
rsi241: 0x0
rsi242: 0x0
rsi243: 0x0
rsi244: 0x0
rsi245: 0x0
rsi246: 0x0
rsi247: 0x0
rsi248: 0x0
rsi249: 0x0
rsi250: 0x0
rsi251: 0x0
rsi252: 0x0
rsi253: 0x0
rsi254: 0x0
rsi255: 0x0
rsi256: 0x0
rsi257: 0x0
rsi258: 0x0
rsi259: 0x0
rsi260: 0x0
rsi261: 0x0
rsi262: 0x0
rsi263: 0x0
rsi264: 0x0
rsi265: 0x0
rsi266: 0x0
rsi267: 0x0
rsi268: 0x0
rsi269: 0x0
rsi270: 0x0
rsi271: 0x0
rsi272: 0x0
rsi273: 0x0
rsi274: 0x0
rsi275: 0x0
rsi276: 0x0
rsi277: 0x0
rsi278: 0x0
rsi279: 0x0
rsi280: 0x0
rsi281: 0x0
rsi282: 0x0
rsi283: 0x0
rsi284: 0x0
rsi285: 0x0
rsi286: 0x0
rsi287: 0x0
rsi288: 0x0
rsi289: 0x0
rsi290: 0x0
rsi291: 0x0
rsi292: 0x0
rsi293: 0x0
rsi294: 0x0
rsi295: 0x0
rsi296: 0x0
rsi297: 0x0
rsi298: 0x0
rsi299: 0x0
rsi300: 0x0
rsi301: 0x0
rsi302: 0x0
rsi303: 0x0
rsi304: 0x0
rsi305: 0x0
rsi306: 0x0
rsi307: 0x0
rsi308: 0x0
rsi309: 0x0
rsi310: 0x0
rsi311: 0x0
rsi312: 0x0
rsi313: 0x0
rsi314: 0x0
rsi315: 0x0
rsi316: 0x0
rsi317: 0x0
rsi318: 0x0
rsi319: 0x0
rsi320: 0x0
rsi321: 0x0
rsi322: 0x0
rsi323: 0x0
rsi324: 0x0
rsi325: 0x0
rsi326: 0x0
rsi327: 0x0
rsi328: 0x0
rsi329: 0x0
rsi330: 0x0
rsi331: 0x0
rsi332: 0x0
rsi333: 0x0
rsi334: 0x0
rsi335: 0x0
rsi336: 0x0
rsi337: 0x0
rsi338: 0x0
rsi339: 0x0
rsi340: 0x0
rsi341: 0x0
rsi342: 0x0
rsi343: 0x0
rsi344: 0x0
rsi345: 0x0
rsi346: 0x0
rsi347: 0x0
rsi348: 0x0
rsi349: 0x0
rsi350: 0
```

- ▶ Anything that can be interpreted as executable code
- ▶ Disassembler examines bytes and reconstructs assembly source

Today: Machine Programming I: Basics

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

x86-64 Integer Registers

%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

Can reference low-order 4 bytes (also low-order 1 & 2 bytes)

Some History: IA32 Registers

Origin
(mostly obsolete)

general purpose	%eax	%ax	%ah	%al	<i>accumulate</i>
	%ecx	%cx	%ch	%cl	<i>counter</i>
	%edx	%dx	%dh	%dl	<i>data</i>
	%ebx	%bx	%bh	%bl	<i>base</i>
	%esi	%si			<i>source index</i>
	%edi	%di			<i>destination index</i>
	%esp	%sp			<i>stack pointer</i>
	%ebp	%bp			<i>base pointer</i>

16-bit virtual registers
(backwards compatibility)

Moving Data

- ▶ Moving Data

movq *Source, Dest:*

- ▶ Operand Types

- **Immediate:** Constant integer data
 - Example: **\$0x400**, **\$-533**
 - Like C constant, but prefixed with **`\$'**
 - Encoded with 1, 2, or 4 bytes
- **Register:** One of 16 integer registers
 - Example: **%rax**, **%r13**
 - But **%rsp** reserved for special use
 - Others have special uses for particular instructions
- **Memory:** 8 consecutive bytes of memory at address given by register
 - Simplest example: **(%rax)**
 - Various other “address modes”

%rax
%rcx
%rdx
%rbx
%rsi
%rdi
%rsp
%rbp
%rN

movq Operand Combinations

	Source	Dest	Src, Dest	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;

Cannot do memory-memory transfer with a single instruction

Simple Memory Addressing Modes

► Normal (R) Mem[Reg[R]]

- Register R specifies memory address
- Aha! Pointer dereferencing in C

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

► Displacement D(R) Mem[Reg[R]+D]

- Register R specifies start of memory region
- Constant displacement D specifies offset

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

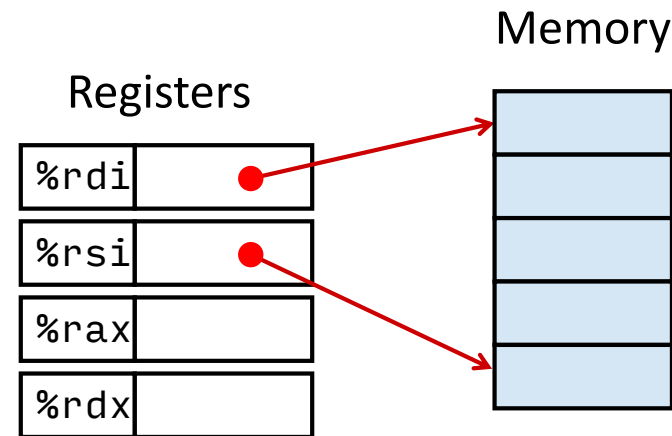

Example of Simple Addressing Modes

```
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
```

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
```

Understanding `Swap()`

Registers

%rdi	0x120
%rsi	0x100
%rax	
%rdx	

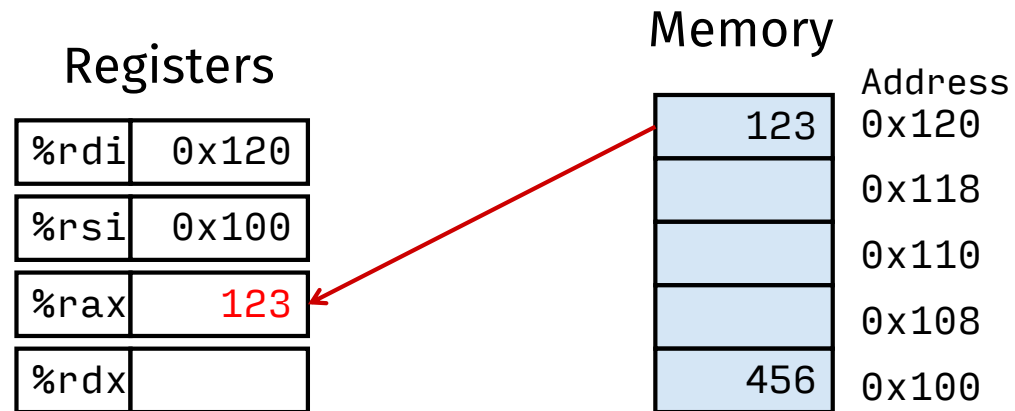
Memory

	Address
123	0x120
	0x118
	0x110
	0x108
456	0x100

`swap:`

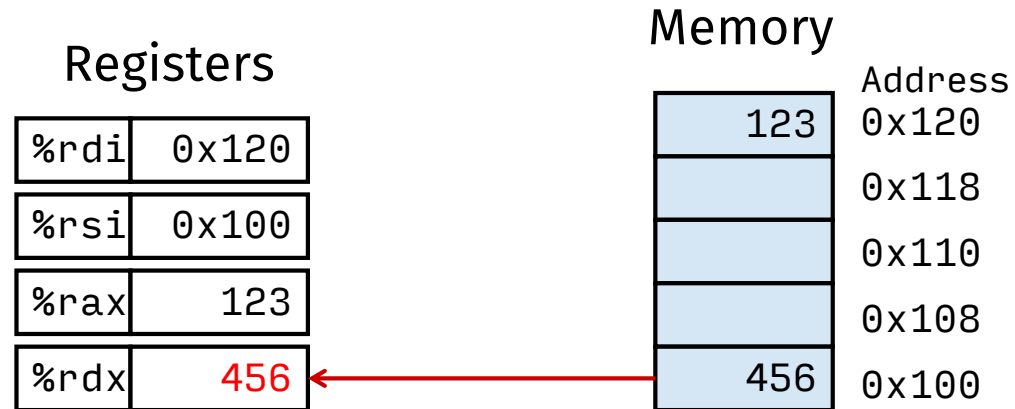
```
    movq    (%rdi), %rax    # t0 = *xp
    movq    (%rsi), %rdx    # t1 = *yp
    movq    %rdx, (%rdi)    # *xp = t1
    movq    %rax, (%rsi)    # *yp = t0
    ret
```

Understanding `Swap()`



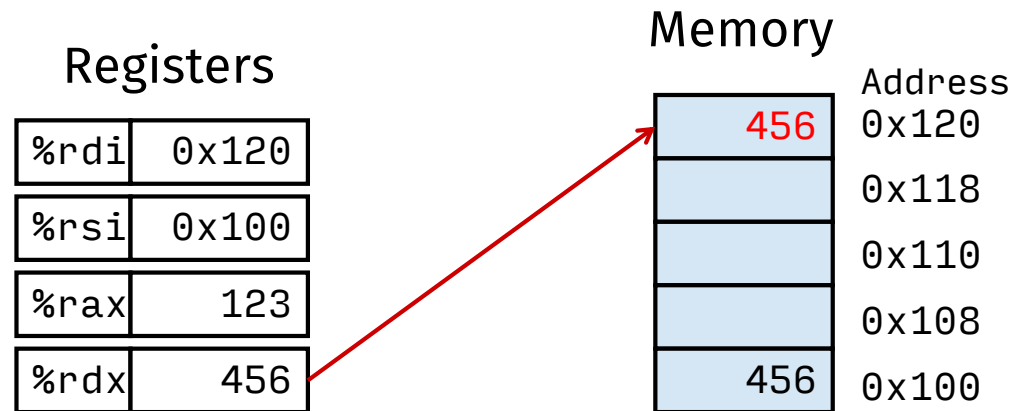
```
swap:
    movq    (%rdi), %rax    # t0 = *xp
    movq    (%rsi), %rdx    # t1 = *yp
    movq    %rdx, (%rdi)    # *xp = t1
    movq    %rax, (%rsi)    # *yp = t0
    ret
```

Understanding `Swap()`



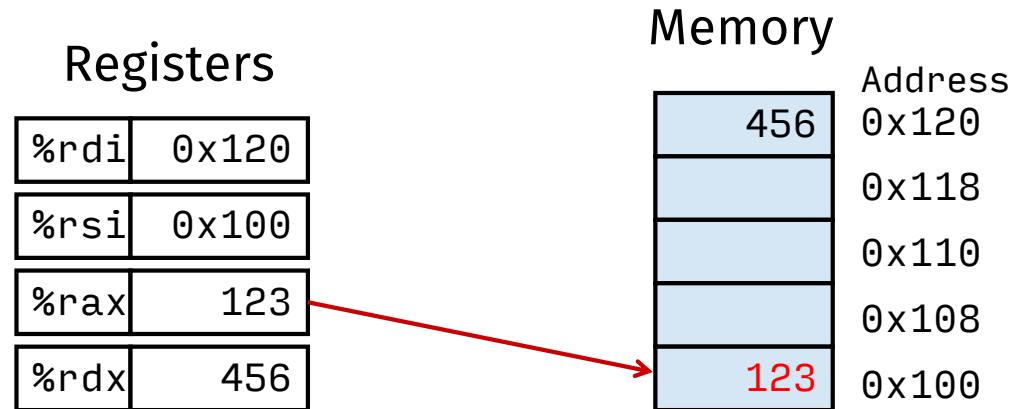
```
swap:
    movq    (%rdi), %rax    # t0 = *xp
    movq    (%rsi), %rdx    # t1 = *yp
    movq    %rdx, (%rdi)    # *xp = t1
    movq    %rax, (%rsi)    # *yp = t0
    ret
```

Understanding `Swap()`



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

Understanding `Swap()`



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

Simple Memory Addressing Modes

► Normal (R) Mem[Reg[R]]

- Register R specifies memory address
- Aha! Pointer dereferencing in C

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

► Displacement D(R) Mem[Reg[R]+D]

- Register R specifies start of memory region
- Constant displacement D specifies offset

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


Complete Memory Addressing Modes

► Most General Form

$D(Rb, Ri, S)$ $Mem[Reg[Rb] + S * Reg[Ri] + D]$

- D: Constant “displacement” 1, 2, or 4 bytes
- Rb: Base register: Any of 16 integer registers
- Ri: Index register: Any, except for **%rsp**
- S: Scale: 1, 2, 4, or 8 (*why these numbers?*)

► Special Cases

(Rb, Ri) $Mem[Reg[Rb] + Reg[Ri]]$

$D(Rb, Ri)$ $Mem[Reg[Rb] + Reg[Ri] + D]$

(Rb, Ri, S) $Mem[Reg[Rb] + S * Reg[Ri]]$

Address Computation Examples

%rdx	0xf000
%rcx	0x0100

Expression	Address Computation	Address
0x8(%rdx)		
(%rdx,%rcx)		
(%rdx,%rcx,4)		
0x80(,%rdx,2)		

Address Computation Examples

%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

Today: Machine Programming I: Basics

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

Address Computation Instruction

- ▶ **leaq Src, Dst**
 - **Src** is address mode expression
 - Set **Dst** to address denoted by expression
- ▶ 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
```

Some Arithmetic Operations

- ▶ Two Operand Instructions:

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

`incq Dest Dest = Dest + 1`

`decq Dest Dest = Dest - 1`

`negq Dest Dest = - Dest`

`notq Dest Dest = ~Dest`

- ▶ See book for more instructions

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