MACHINE LEVEL PROGRAMMING 1: BASICS

CS 045

Computer Organization and Architecture

Prof. Donald J. Patterson

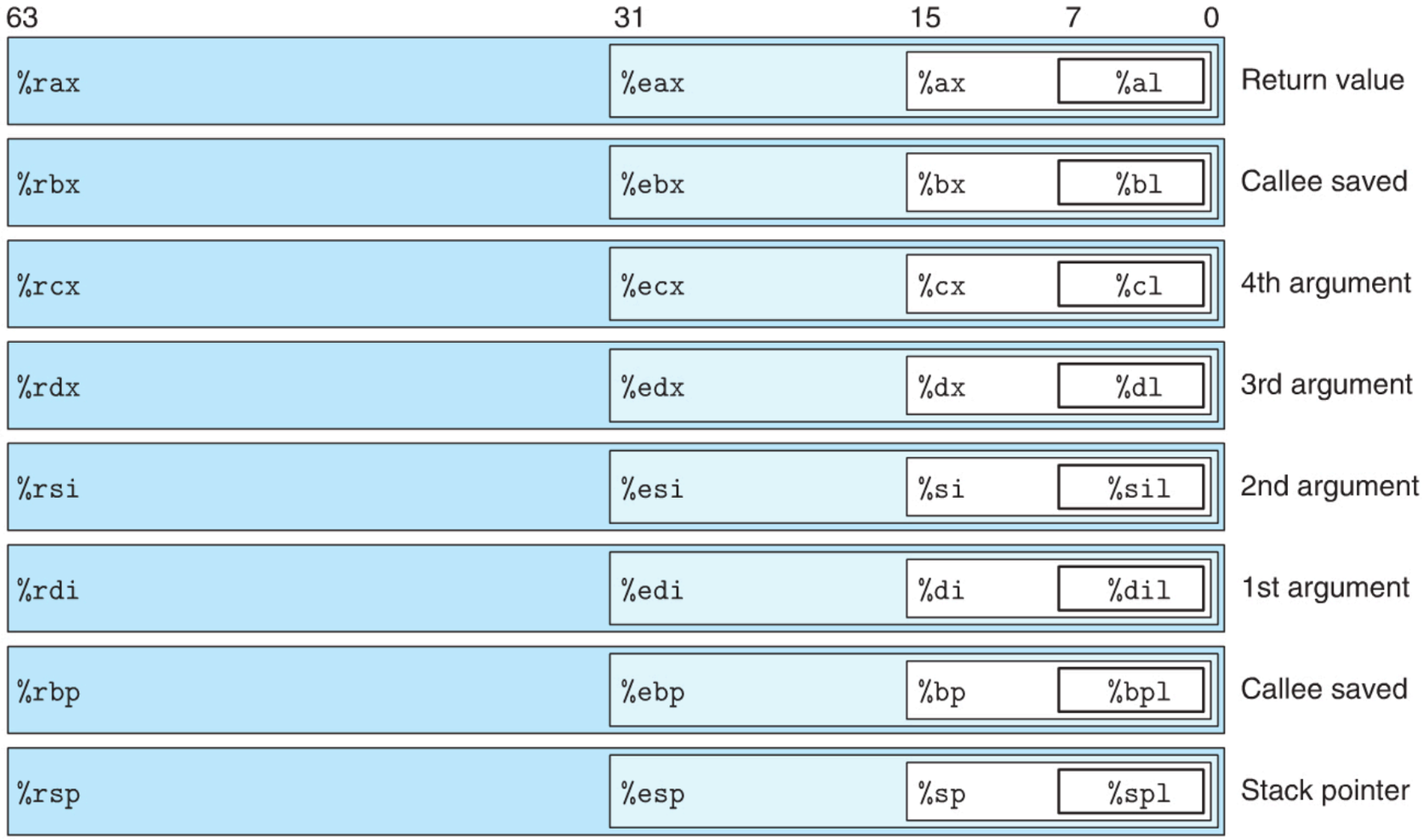
Adapted from Bryant and O'Hallaron,
Computer Systems:
A Programmer's Perspective, Third Edition

- HISTORY OF INTEL PROCESSORS AND ARCHITECTURES
- C, ASSEMBLY, MACHINE CODE
- ASSEMBLY BASICS: REGISTERS, OPERANDS, MOVE
- ARITHMETIC & LOGICAL OPERATIONS

- HISTORY OF INTEL PROCESSORS AND ARCHITECTURES
- C, ASSEMBLY, MACHINE CODE
- ASSEMBLY BASICS: REGISTERS, OPERANDS, MOVE
- ARITHMETIC & LOGICAL OPERATIONS

C, ASSEMBLY, MACHINE CODE

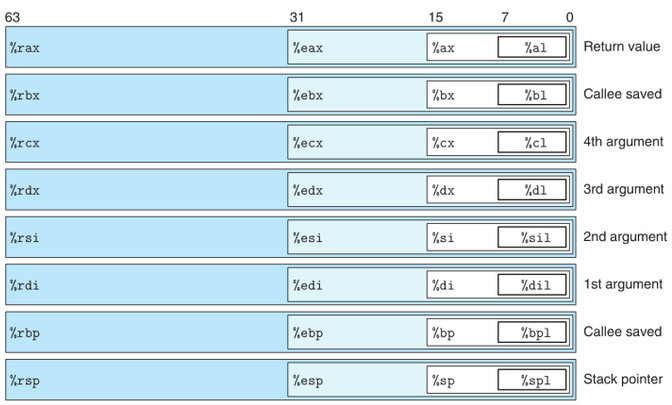
X86-64 INTEGER REGISTERS

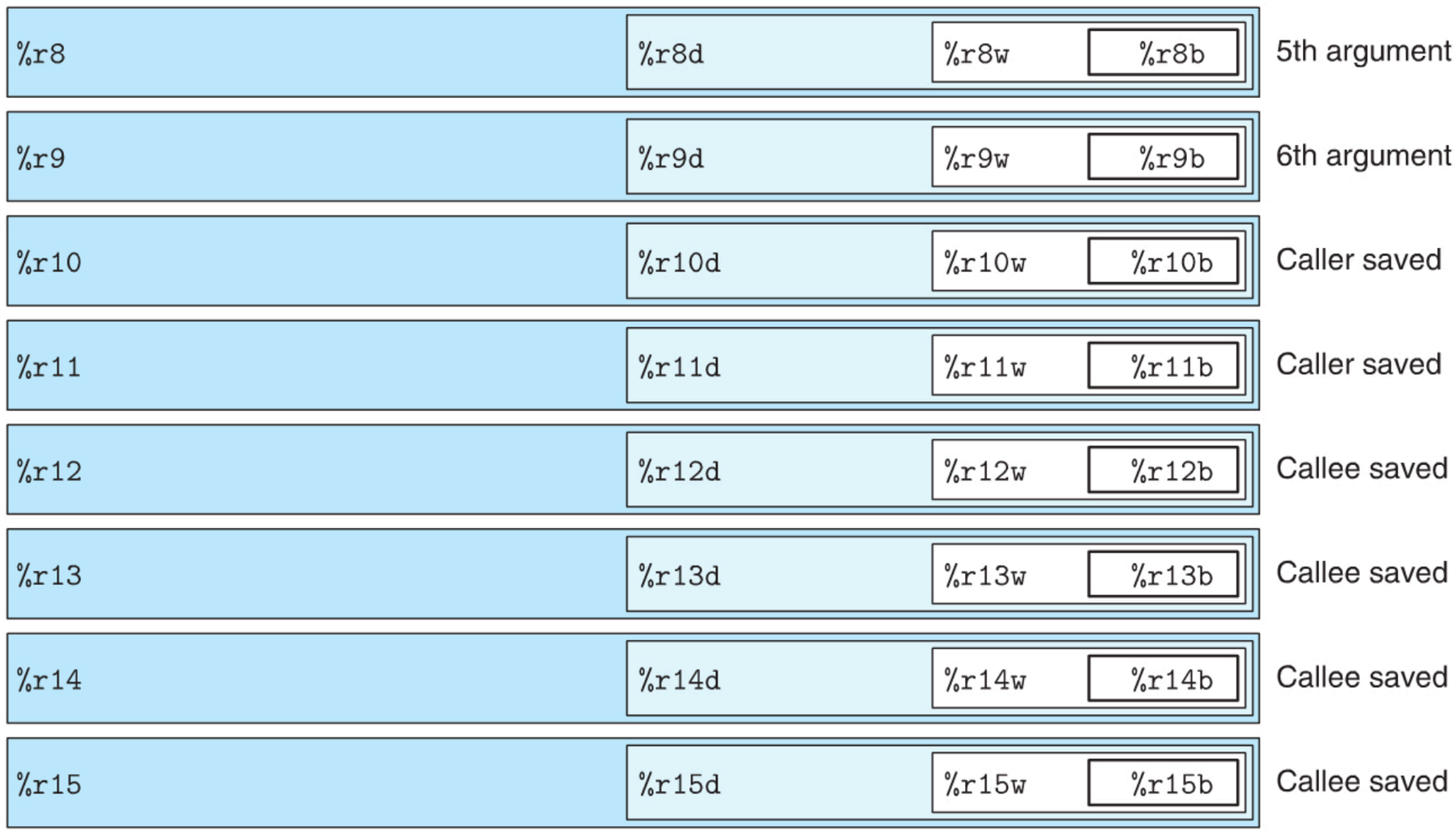


- 16 total
  - Can reference low-order 4 bytes (also low-order 1 & 2 bytes)

C, ASSEMBLY, MACHINE CODE

X86-64 INTEGER REGISTERS



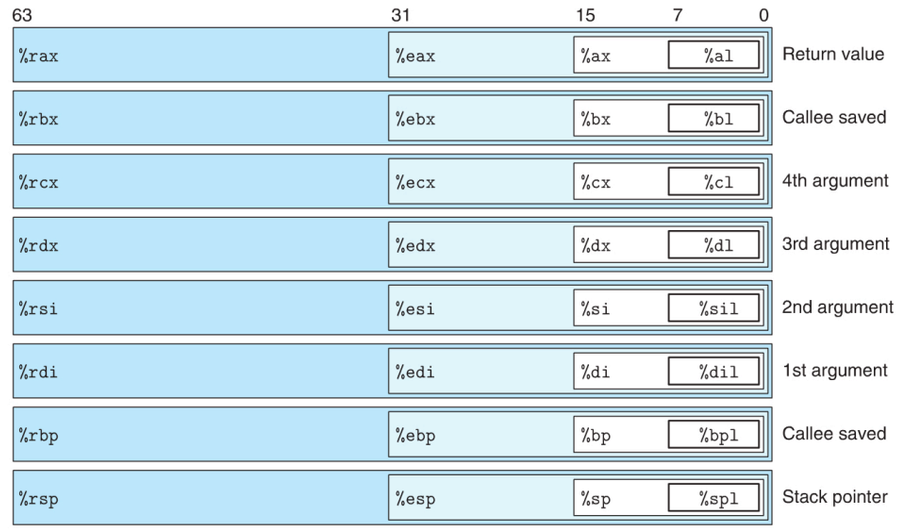


- 16 total
  - Can reference low-order 4 bytes (also low-order 1 & 2 bytes)

C, ASSEMBLY, MACHINE CODE

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 (a pointer): `(%rax)`
    - Various other "address modes"



C, ASSEMBLY, MACHINE CODE

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

C, ASSEMBLY, MACHINE CODE

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
```

C, ASSEMBLY, MACHINE CODE

EXAMPLE OF SIMPLE ADDRESSING MODES

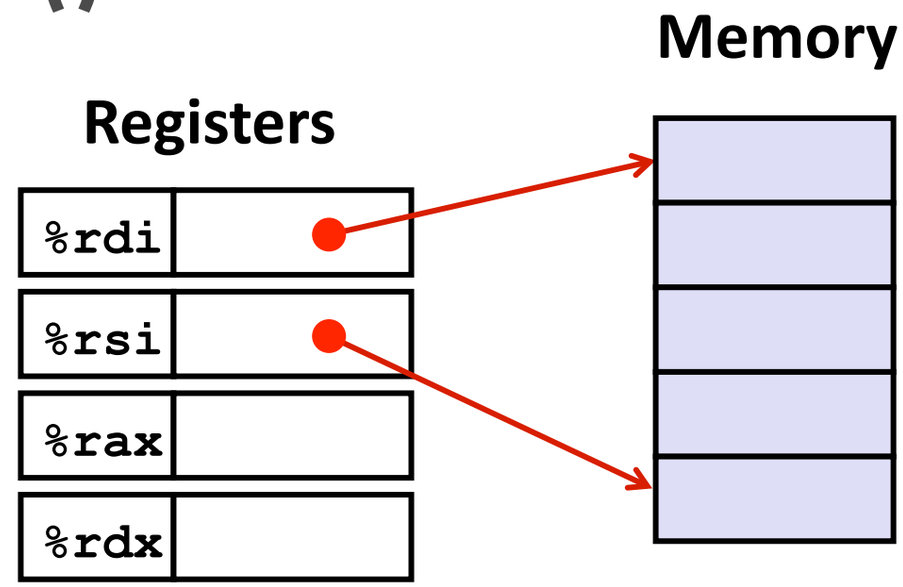
```
void swap
    (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING 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
```

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x00000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x00000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

1_{10}

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x00000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

1_{10}	0x0000000000000001	0x120
		0x118
		0x110
		0x108
57346_{10}	0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

1_{10}	0x0000000000000001	0x120
		0x118
		0x110
		0x108
57346_{10}	0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

1_{10}

0x0000000000000001	0x120
	0x118
	0x110
	0x108
57346_{10} 0x000000000000e002	0x100

8 bytes

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x00000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

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

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x00000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

0x0000000000000001	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

0x000000000000e002	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

0x000000000000e002	0x120
	0x118
	0x110
	0x108
0x000000000000e002	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

0x000000000000e002	0x120
	0x118
	0x110
	0x108
	0x100

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

C, ASSEMBLY, MACHINE CODE

DECONSTRUCTING SWAP()

Registers

%rdi	0x120
%rsi	0x100
%rax	0x0000000000000001
%rdx	0x000000000000e002

Memory

0x000000000000e002	0x120
	0x118
	0x110
	0x108
0x0000000000000001	0x100

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

C, ASSEMBLY, MACHINE CODE

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
```

SIMPLE 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]]**

C, ASSEMBLY, MACHINE CODE

ADDRESS COMPUTATION EXAMPLES

%rdx	0xf000
%rcx	0x0100

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

(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]]**

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

- HISTORY OF INTEL PROCESSORS AND ARCHITECTURES
- C, ASSEMBLY, MACHINE CODE
- ASSEMBLY BASICS: REGISTERS, OPERANDS, MOVE
- ARITHMETIC & LOGICAL OPERATIONS

- HISTORY OF INTEL PROCESSORS AND ARCHITECTURES
- C, ASSEMBLY, MACHINE CODE
- ASSEMBLY BASICS: REGISTERS, OPERANDS, MOVE
- ARITHMETIC & LOGICAL OPERATIONS

ADDRESS COMPUTATION EXAMPLES

- `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, or 8
- Example

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

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

ADDRESS COMPUTATION EXAMPLES

- `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, or 8
- Example

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

Converted to assembly by compiler

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

C, ASSEMBLY, MACHINE CODE

SOME ARITHMETIC OPERATIONS

- Two Operand Instructions:
  - 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`
- Watch out for argument order!
  - No distinction between signed and unsigned int (why?)

C, ASSEMBLY, MACHINE CODE

SOME ARITHMETIC OPERATIONS

- One Operand Instructions
  - Format Computation
  - `incq  dest` `dest = dest + 1`
  - `decq  dest` `dest = dest  1`
  - `negq  dest` `dest = -dest`
  - `notq  dest` `dest = ~dest`

- See book for more instructions

WORK IT OUT

C, ASSEMBLY, MACHINE CODE

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

C, ASSEMBLY, MACHINE CODE

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

SUMMARY

SUMMARY

- History of Intel processors and architectures
  - Evolutionary design leads to many quirks and artifacts

SUMMARY

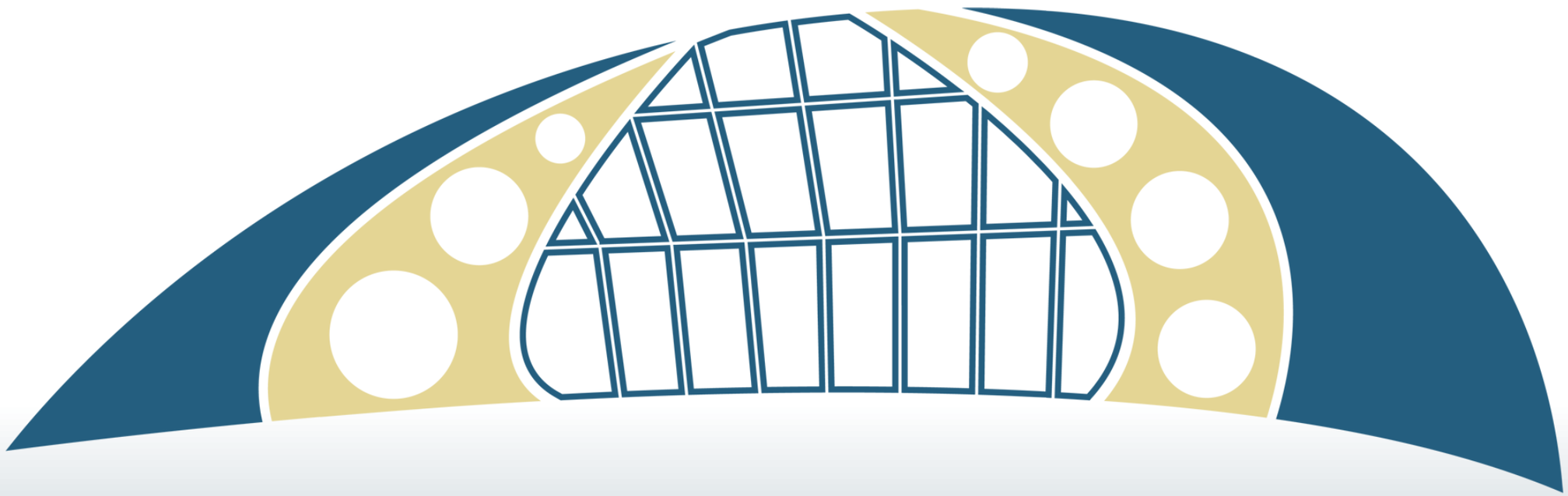
- History of Intel processors and architectures
  - Evolutionary design leads to many quirks and artifacts
- C, assembly, machine code
  - New forms of visible state: program counter, registers, …
  - Compiler must transform statements, expressions, procedures into low-level instruction sequences

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

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



WESTMONT INSPIRED

COMPUTING LAB