CSO-1

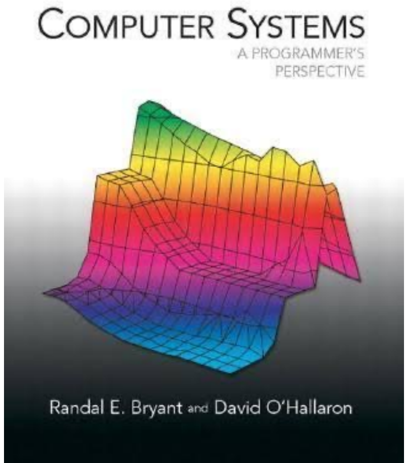
X86 Assembly

Daniel G. Graham PhD

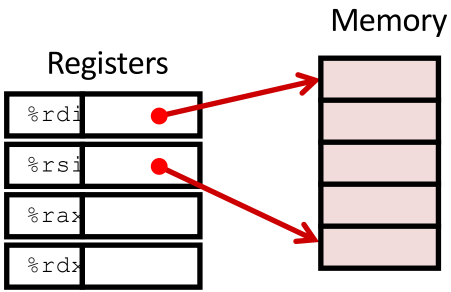
University of Virginia | ENGINEERING

Contents

1. Mov vs Lea (instructions)
2. Jump tables (Switch Statements)
3. References: computer systems a programmer perceptive



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




```
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

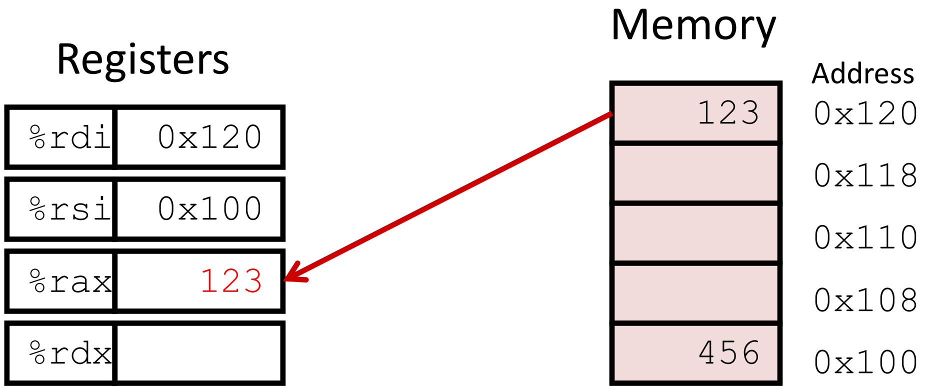
%rdi	0x120
%rsi	0x100
%rax	
%rdx	

Memory

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

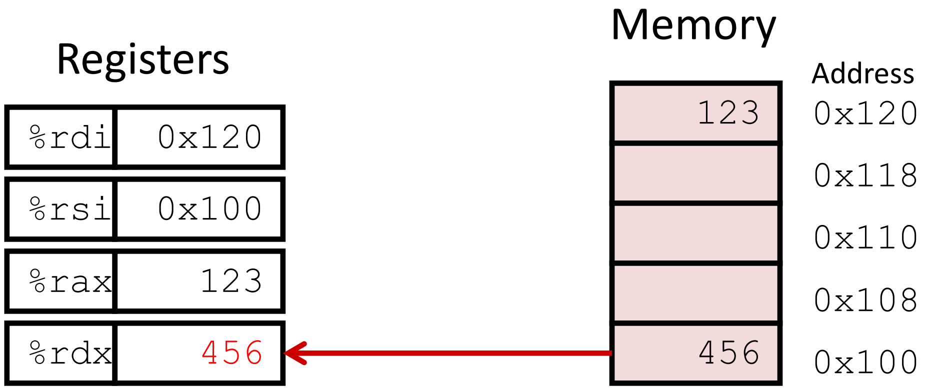
```
swap:
   movq    (%rdi), %rax   # t0 = *xp
   movq    (%rsi), %rdx   # t1 = *yp
   movq    %rdx, (%rdi)   # *xp = t1
   movq    %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()




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

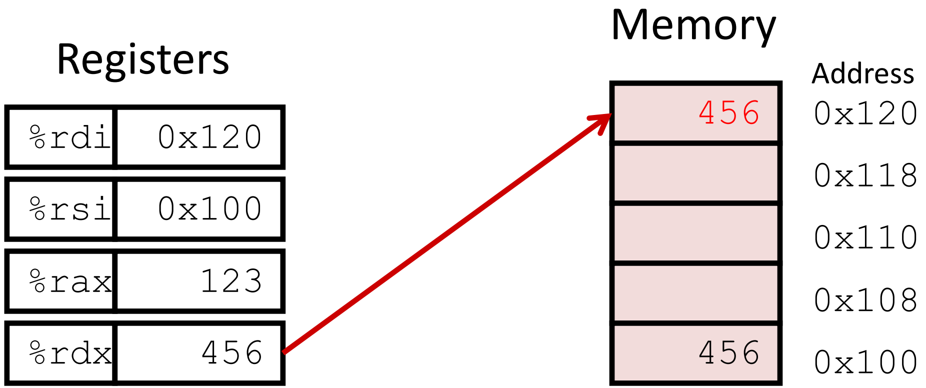
Understanding Swap()




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

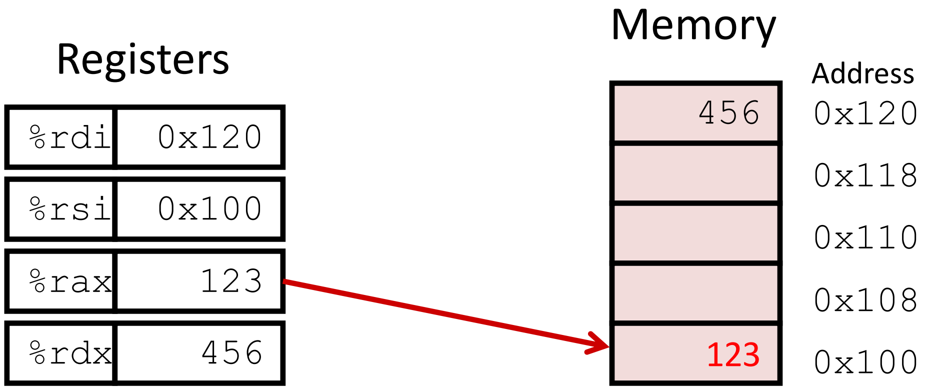
University of Virginia ENGINEERING

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
```

LOAD EFFECTIVE ADDRESS

leaq vs. movq example

Registers

%rax	
%rbx	
%rcx	0x4
%rdx	0x100
%rdi	
%rsi	

Memory

	Address
0x400	0x120
0xf	0x118
0x8	0x110
0x10	0x108
0x1	0x100

```
leaq  (%rdx,%rcx,4),  %rax
movq  (%rdx,%rcx,4),  %rbx
leaq  (%rdx), %rdi
movq  (%rdx), %rsi
```

leaq vs. movq example

Registers

%rax	0x110
%rbx	
%rcx	0x4
%rdx	0x100
%rdi	
%rsi	

Memory

	Address
0x400	0x120
0xf	0x118
0x8	0x110
0x10	0x108
0x1	0x100

```
leaq  (%rdx,%rcx,4),   %rax
movq  (%rdx,%rcx,4),   %rbx
leaq  (%rdx), %rdi
movq  (%rdx), %rsi
```

```
%rdx + %rcx * 4 -> %rax
```

0x100 + (0x4 * 4) = 0x110

leaq vs. movq example

Registers

%rax	0x110
%rbx	0x8
%rcx	0x4
%rdx	0x100
%rdi	
%rsi	

Memory

	Address
0x400	0x120
0xf	0x118
0x8	0x110
0x10	0x108
0x1	0x100

```
Leaq  (%rdx,%rcx,4),   %rax
Movq  (%rdx,%rcx,4),   %rbx
leaq  (%rdx), %rdi
movq  (%rdx), %rsi
```

`%rdx + %rcx * 4 -> %rbx`

0x100 + (0x4 * 4) = 0x110

leaq vs. movq example

Registers

%rax	0x110
%rbx	0x8
%rcx	0x4
%rdx	0x100
%rdi	0x100
%rsi	

Memory

	Address
0x400	0x120
0xf	0x118
0x8	0x110
0x10	0x108
0x1	0x100

```
Leaq  (%rdx,%rcx,4),   %rax
Movq  (%rdx,%rcx,4),   %rbx
leaq  (%rdx), %rdi
movq  (%rdx), %rsi
```

leaq vs. movq example

Registers

%rax	0x110
%rbx	0x8
%rcx	0x4
%rdx	0x100
%rdi	0x100
%rsi	0x1

Memory

	Address
0x400	0x120
0xf	0x118
0x8	0x110
0x10	0x108
0x1	0x100

```
Leaq  (%rdx,%rcx,4),  %rax
Movq  (%rdx,%rcx,4),  %rbx
leaq  (%rdx), %rdi
movq  (%rdx), %rsi
```

LEA tricks

```
leaq (%rax,%rax,4), %rax
```

rax ← rax × 5

rax ← `address-of(memory[rax + rax * 4])`

```
leaq (%rbx,%rcx), %rdx
```

rdx ← rbx + rcx

rdx ← `address-of(memory[rbx + rcx])`

SWITCH STATEMENT AND JUMP TABLES

```c
long switch_eg(long x, long y, long z){
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
        /* Fall Through */
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

SWITCH STATEMENT

- Fall through cases
  - Here: 2
- Multiple case labels
  - Here: 5 & 6
- Missing cases
  - Here: 4

Switch Form

```
switch(x) {
  case val_0:
    Block 0
  case val_1:
    Block 1
    • • •
  case val_n-1:
    Block n–1
}
```

Jump Table

```
jtab: Targ0
      Targ1
      Targ2
      •
      •
      •
      Targn-1
```

Jump Targets

`Targ0:` Code Block 0

`Targ1:` Code Block 1

`Targ2:` Code Block 2

`Targn-1:` Code Block n–1

```c
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
      . . .
    }
    return w;
}
```

Register	Use(s)
`%rdi`	Argument **`x`**
`%rsi`	Argument **`y`**
`%rdx`	Argument **`z`**
`%rax`	Return value

Setup:

What range of values takes default?

```
switch_eg:
   --SNIP--
    cmpq    $6, %rdi    # x:6
    ja      .L8
    jmp     *.L4(,%rdi,8)
```

Note that **`w`** not initialized here

UNIVERSITY of VIRGINIA | ENGINEERING

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
      . . .
    }
    return w;
}
```

Jump table

```
.section    .rodata
  .align 8
.L4:
  .quad     .L8   # x = 0
  .quad     .L3   # x = 1
  .quad     .L5   # x = 2
  .quad     .L9   # x = 3
  .quad     .L8   # x = 4
  .quad     .L7   # x = 5
  .quad     .L7   # x = 6
```

Setup:

```
switch_eg:
    movq    %rdx, %rcx
    cmpq    $6, %rdi          # x:6
    ja      .L8               # Use default
    jmp     *.L4(,%rdi,8)     # goto *JTab[x]
```

Indirect jump

- Table Structure
  - Each target requires 8 bytes
  - Base address at **`.L4`**
- Jumping
  - Direct: **`jmp .L8`**
  - Jump target is denoted by label **`.L8`**
  - Indirect: **`jmp *.L4(,%rdi,8)`**
  - Start of jump table: **`.L4`**

Jump table

```
.section    .rodata
  .align 8
.L4:
  .quad     .L8   # x = 0
  .quad     .L3   # x = 1
  .quad     .L5   # x = 2
  .quad     .L9   # x = 3
  .quad     .L8   # x = 4
  .quad     .L7   # x = 5
  .quad     .L7   # x = 6
```

Jump table

```
.section    .rodata
  .align 8
.L4:
  .quad     .L8   # x = 0
  .quad     .L3   # x = 1
  .quad     .L5   # x = 2
  .quad     .L9   # x = 3
  .quad     .L8   # x = 4
  .quad     .L7   # x = 5
  .quad     .L7   # x = 6
```

```
switch(x) {
case 1:       // .L3
    w = y*z;
    break;
case 2:       // .L5
    w = y/z;
    /* Fall Through */
case 3:       // .L9
    w += z;
    break;
case 5:
case 6:       // .L7
    w -= z;
    break;
default:      // .L8
    w = 2;
}
```

```c
  switch(x) {
  case 1:    // .L3
        w = y*z;
        break;
   . . .
}
```

```
.L3:
   movq    %rsi, %rax  # y
   imulq   %rdx, %rax  # y*z
   ret
```

Register	Use(s)
`%rdi`	Argument **x**
`%rsi`	Argument **y**
`%rdx`	Argument **z**
`%rax`	Return value

```
long w = 1;
. . .
switch(x) {
. . .
case 2:
     w = y/z;
     /* Fall Through */
case 3:
     w += z;
     break;
. . .
}
```

```
case 2:
     w = y/z;
     goto merge;
```

```
case 3:
     w = 1;
```

```
merge:
     w += z;
```

```
long w = 1;
. . .
switch(x) {
. . .
case 2:
    w = y/z;
    /* Fall Through */
case 3:
    w += z;
    break;
. . .
}
```

```
.L5:                     # Case 2
    movq   %rsi, %rax
    cqto
    idivq  %rcx          #  y/z
    jmp    .L6           #  goto merge
.L9:                     # Case 3
    movl   $1, %eax      #  w = 1
.L6:                     # merge:
    addq   %rcx, %rax    #  w += z
    ret
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rdx	Argument **z**
%rax	Return value

```
  switch(x) {
    . . .
    case 5:  // .L7
    case 6:  // .L7
        w -= z;
        break;
    default: // .L8
        w = 2;
}
```

```
.L7:                      # Case 5,6
  movl   $1, %eax        #  w = 1
  subq   %rdx, %rax      #  w -= z
  ret
.L8:                      # Default:
  movl   $2, %eax        #  2
  ret
```

Register	Use(s)
`%rdi`	Argument **x**
`%rsi`	Argument **y**
`%rdx`	Argument **z**
`%rax`	Return value

SPRING 2023 EXAM 2

5. [24 points] Assume the first eight registers and the given segment of memory have the following values before the next few instructions.

Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

Which program registers are modified, and to what values, by the following instructions? Leave spaces blank if fewer registers change than there are lines. If no registers are changed, write "none" in the first register box with no new value. *Each instruction below is independent; do not use the result of one as input for the next.* (4 points each)

```
movl 0x8(%rbp), %edx
```

Register	New Value

```
leaq 0x8(%rbp), %rdx
```

Register	New Value

5. [24 points] Assume the first eight registers and the given segment of memory have the following values before the next few instructions.

Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

```
testq %rdx, %rdi
```

Register	New Value

```
andl -0x10(%rsp,%rdx,2), %ecx
```

Register	New Value

5. [24 points] Assume the first eight registers and the given segment of memory have the following values before the next few instructions.

Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

`popw %ax`

Register	New Value

`callq foo`

Register	New Value

NEXT TIME

Synthisis:

1. Writing a recursive function in C
2. Compiling it
3. Then looking at it behaviour in the lldb debugger.

University of Virginia | ENGINEERING