CSO-1

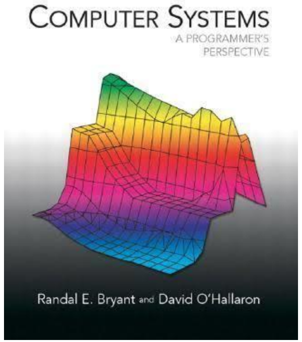
X86 Assembly

Daniel G. Graham PhD

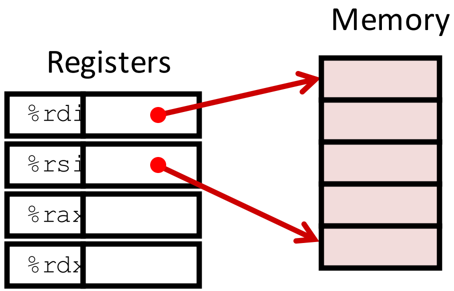
University of Virginia | ENGINEERING

Contents

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

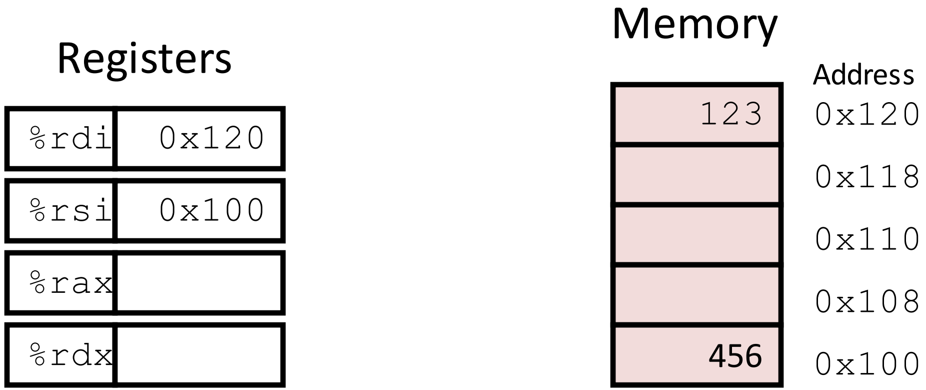


```
swap:
   movq    (%rdi), %rax
   movq    (%rsi), %rdx
   movq    %rdx, (%rdi)
   movq    %rax, (%rsi)
   ret
```



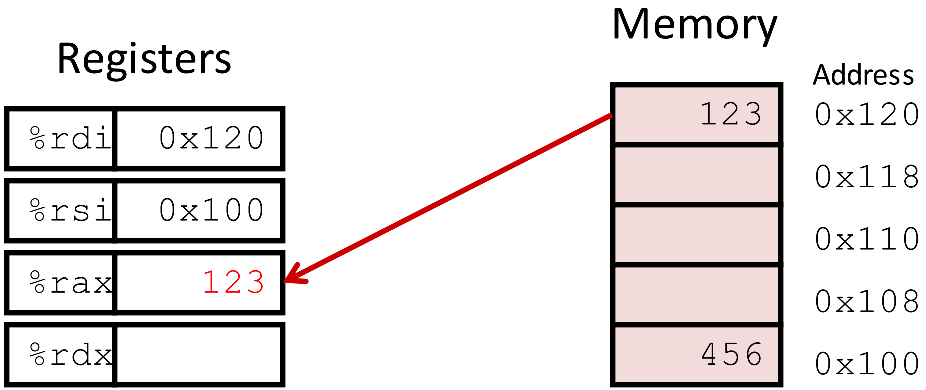

```
swap:
   movq    (%rdi), %rax   # t0 = *xp
   movq    (%rsi), %rdx   # t1 = *yp
   movq    %rdx, (%rdi)   # *xp = t1
   movq    %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()



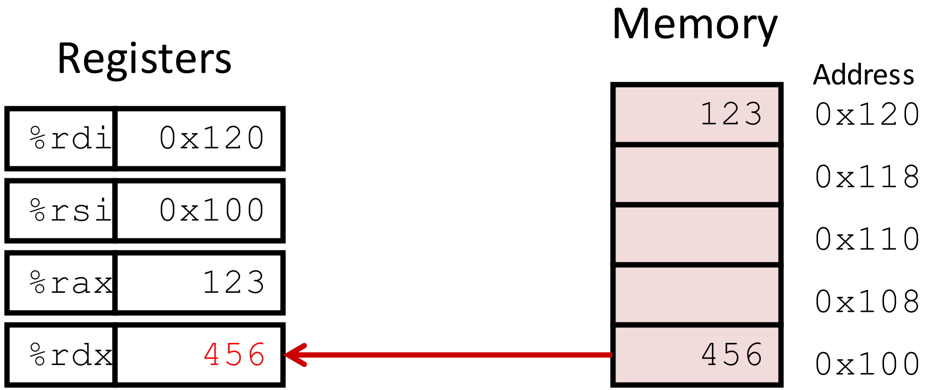

```
swap:
   movq    (%rdi), %rax   # t0 = *xp
   movq    (%rsi), %rdx   # t1 = *yp
   movq    %rdx, (%rdi)   # *xp = t1
   movq    %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()



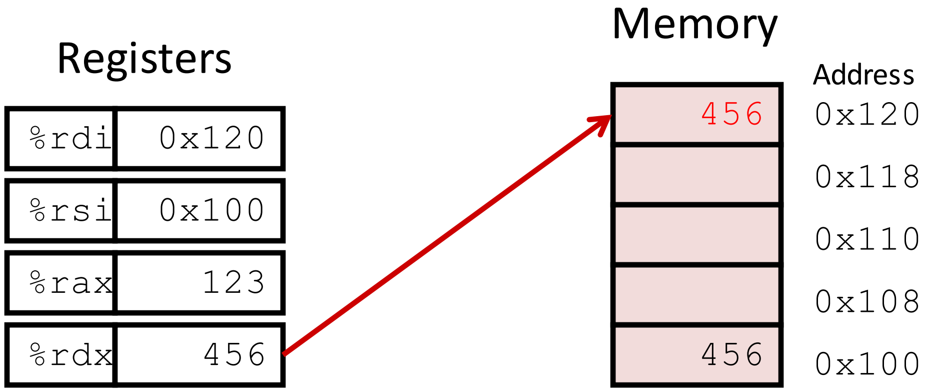

```
swap:
    movq     (%rdi), %rax   # t0 = *xp
    movq     (%rsi), %rdx   # t1 = *yp
    movq     %rdx, (%rdi)   # *xp = t1
    movq     %rax, (%rsi)   # *yp = t0
    ret
```

Understanding `Swap()`



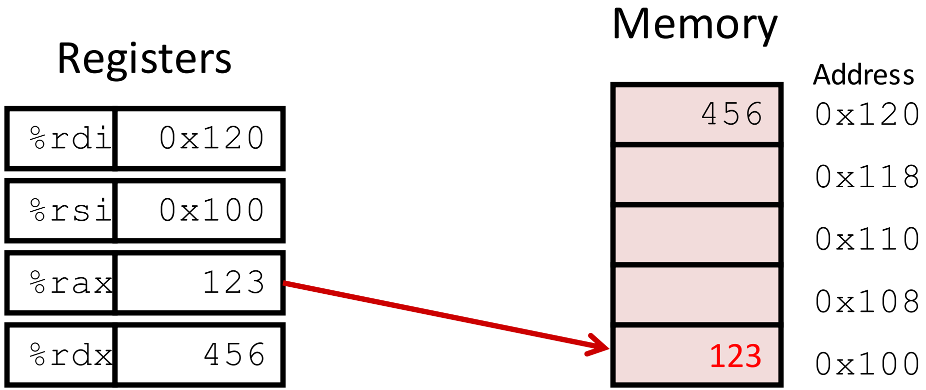

```
swap:
    movq     (%rdi), %rax   # t0 = *xp
    movq     (%rsi), %rdx   # t1 = *yp
    movq     %rdx, (%rdi)   # *xp = t1
    movq     %rax, (%rsi)   # *yp = t0
    ret
```

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
```

LOAD EFFECTIVE ADDRESS

leaq vs. movq example

Registers

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

Memory

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

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

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

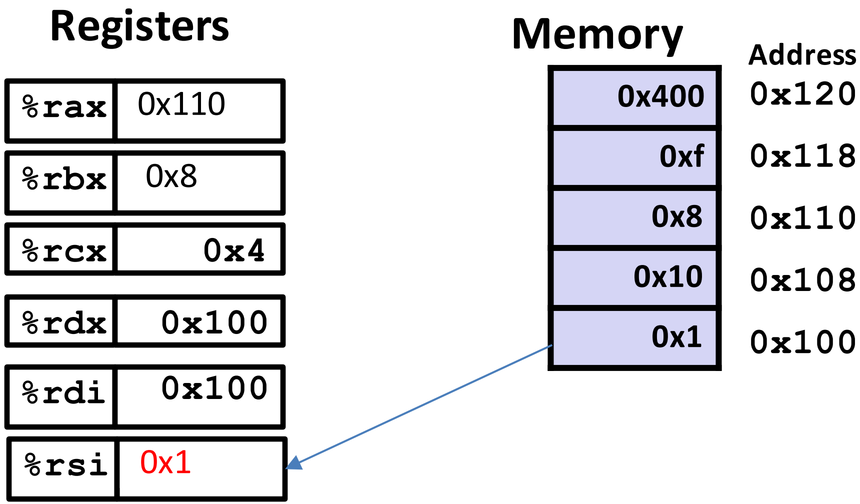
%rax	0x110
%rbx	0x8
%rcx	0x4
%rdx	0x100
%rdi	0x100
%rsi	

Memory

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




```
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])`

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

Synthesis:

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

UNIVERSITY of VIRGINIA | ENGINEERING