CSE 2421

X86-64Assembly Language – Part 2:
Stack, registers, assembler directives, and
data movement instructions

Section 1 Programs and Addressing

Assembler directives (“pseudo-ops”)

- .file
  - Allows a name to be assigned to the assembly language source code file.
- .section
  - This makes the specified section the current section.
  - .rodata
    - Specifies that the following data is to be placed in the read only memory portion of the executable
- .string
  - Specifies that the characters enclosed in quotation marks are to be stored in memory, terminated by a null byte
- .data
  - Changes or sets the current section to the data section
- .text
  - Changes or sets the current section to the text (or code) section

Assembler directives (continued)

- .globl
  - A directive needed by the linker for symbol resolution: followed by name of function
- .type
  - Needed by the linker to identify the label as one associated with a function, as opposed to data
- .size
  - Needed by the linker to identify the size of the text for the program

- **Note**: labels (for functions or data) in assembly language source code are followed by a colon.

Data size assembler directives

- .quad *value*
  - Places the given value, (0x prefix for hex, no prefix for decimal) in memory, encoded in 8 bytes
- .long *value*
  - Places the given value, (0x prefix for hex, no prefix for decimal) in memory, encoded in 4 bytes
- .word *value*
  - Places the given value, (0x prefix for hex, no prefix for decimal) in memory, encoded in 2 bytes
- .byte *value*
  - Places the given value, (0x prefix for hex, no prefix for decimal) in memory, encoded in 1 byte

Run X86 program

```
.file “first.s”
.section .rodata
.data
    .align 8
Array:
    .quad 0x6f
    .quad 0x84
```

```
.text
.globl main
    .type main, @function
main:
    pushq %rbp
    movq %rsp, %rbp

    movq $55,%rdx
    movq %rdx, %rbx
    movq $Array, %rax
    movq %rbx,8(%rax)
    movq (%rax),%rcx

    leave
    ret
.size main, .-main
```

Run X86 program

```
.file “second.s”
.section .rodata
.data
.align 8
Array:
.quad 0x6f
.quad 0x84
.quad 0x55
.quad 0x44
.globl main
    .type main, @function
.text
```

```
main:
pushq %rbp
movq %rsp, %rbp

movq $55,%rdx
movq %rdx, %rbx
movq $0x33, %r8
movq $Array, %rax
movq %rbx,8(%rax)
movq %r8, 24(%rax)
movq %rax,(%rax)
movq (%rax),%rcx

leave
ret
.size main, .-main
```

Assembly Syntax

- Immediate values are preceded by $
  $ -> decimal value
  $0x –> hex value

- Registers are prefixed with %

- Moves and ALU operations are source, destination:
  movq $5, %rax
  movq $0x30, %rbx

- Effective address *DISPLACEMENT(BASE)*

movq $0x30, 8(%rbx)

Review

- What is the size of a memory address on stdlinux???
- So what the only suffix should we be using when we are calculating/moving addresses?
- What size registers should we be using when we are calculating addresses?
- Is there ever an exception to this?

Review

- What is the size of a memory address on stdlinux???

  8 bytes = 64 bits

- So what is the only suffix should we be using when we are calculating/moving addresses?

- What size registers should we be using when we are calculating addresses?

- Is there ever an exception to this?

Review

- What is the size of a memory address on stdlinux???

  8 bytes = 64 bits

- So what is the only suffix should we be using when we are calculating/moving addresses?

  q

- What size registers should we be using when we are calculating addresses?

- Is there ever an exception to this?

Review

- What is the size of a memory address on stdlinux???

  8 bytes = 64 bits

- So what is the only suffix should we be using when we are calculating/moving addresses?

  q

- What size registers should we be using when we are calculating addresses?

  %rax,%rbx, %rcx, %rdx, %r12, etc.

- Is there ever an exception to this?

Review

- What is the size of a memory address on stdlinux???

  8 bytes = 64 bits
- So what is the only suffix should we be using when we are calculating addresses?

  q
- What size registers should we be using when we are calculating/moving addresses?

  %rax,%rbx, %rcx, %rdx, %r12, etc.
- Is there ever an exception to this?

  Not ever!

  (as long as we are working on a 64 bit processor.)

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

- Normal (R) Mem[Reg[R]]
  - Register R specifies memory address

    `movq (%rcx),%rax`
  - Are any of these a valid instruction on stdlinux?

    `movq (%ecx),%rax`

    `movl (%ecx),%eax`

    `movb (%rax),%al`

Simple Memory Addressing Modes

- Normal (R) Mem[Reg[R]]
  - Register R specifies memory address

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

  - Are any of these a valid instruction on stdlinux?

```
movq (%ecx),%rax    #No. must use %rcx
movl (%ecx),%eax
movb (%rax),%al
```

Simple Memory Addressing Modes

- Normal (R) Mem[Reg[R]]
  - Register R specifies memory address

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

  - Are any of these a valid instruction on stdlinux?

```
movq (%ecx),%rax     #No. must use %rcx
movl (%ecx),%eax     #No. must use %rcx
                     # l suffix and dest
                     # of %eax is OK

movb (%rax),%al
```

Simple Memory Addressing Modes

- Normal (R) Mem[Reg[R]]
  - Register R specifies memory address

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

  - Are any of these a valid instruction on stdlinux?

```
movq (%ecx),%rax     #No. must use %rcx
movl (%ecx),%eax     #No. must use %rcx
                     # l suffix and dest
                     # of %eax is OK
movb (%rax),%al      #Yes! Address is 8
                     #byte reg, suffix          #and dest
are 1 byte
```

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
```

Complete Memory Addressing Modes

See Figure 3.3 page 181

- Most General Form

  Imm(Rb,Ri,S) Mem[Imm+ Reg[Rb]+S*Reg[Ri]]
  - Imm: Constant "displacement"
    - It's often a "displacement" of 1, 2, 4 or 8 bytes, but can be any constant value
  - 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?*)
  - This form is seen often when referencing elements of arrays

- Special Cases

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

Complete Memory Addressing Modes

- Examples:

`movq 24(%rax,%rcx,8), %rdx`

means read 8 bytes from this address: (%rax + 8*%rcx + 24)
and store it in %rdx

`movl 24(%rax,%rcx,4), %edx`

means read 4 bytes from this address: (%rax + 4*%rcx + 24)
and store it in %edx

`movw 24(%rax,%rcx,2), %dx`

means read 2 bytes from this address: (%rax + 2*%rcx + 24)
and store it in %dx

`movb 24(%rax,%rcx,1), %dl`

means read 1 byte from this address: (%rax + 1*%rcx + 24)
and store it in %dl

Note that suffix and destination register size match. The change in scale is only so that the example is sensible.

Complete Memory Addressing Modes

- Examples:

`movq %rdx, 24(%rax,%rcx,8)`

means write 8 bytes to this address: (%rax + 8*%rcx + 24) from %rdx

`movl %edx, 24(%rax,%rcx,4)`

means write 4 bytes to this address: (%rax + 4*%rcx + 24) from %edx

`movw %dx, 24(%rax,%rcx,2)`

means write 2 bytes to this address: (%rax + 2*%rcx + 24) from %dx

`movb %dl, 24(%rax,%rcx,1)`

means write 1 byte to this address: (%rax + 1*%rcx + 24) from %dl

Note that suffix and destination register size match. The change in scale is only so that the example is sensible.

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)		
(%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)		
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)		

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