CSO-Recitation 08

CSCI-UA 0201-007

R08: Assessment 06 & Assembly

Today’s Topics

- Assessment 06
- Assembly
- Some exercises
    - give some senses about lab3

Assembly

Assembly programming

Important Instructions

Instruction	What it does
mov src, dest	dest = src
add src, dest	dest = dest + src
sub src, dest	dest = dest - src
imul src, dest	dest = dest * src
inc dest	dest = dest + 1

Moving data - Instruction operands

- src and dest can be one of three things
  1. An **immediate**
     - A constant value, prefaced with $
     - Eg. $0, or $0xabcdabcd
     - dest cannot be an immediate
  2. A **register**
     - One of the 16 general purpose registers
     - Eg. %eax, %rax, %rsi..
  3. A location in **memory**
     - (Register): Consider registers as pointers, and get the value at an address in memory with various "addressing modes"
     - Eg. (%rax), 10(%rax), 10(%rax, rbx, 4)
     - You cannot perform a mov from memory into memory
     - How big is what you are getting from memory, in bytes?

Instruction Suffixes

Suffix	Name	Size (Bytes)
b	byte	1
w	word	2
l	long	4
q	quadword	8

Memory Addressing Modes

- Direct addressing
  - Given a register, use the value located at the memory address contained in the register
  - Register name in parens
  - Eg. mov (%rax), %rbx
- With displacement
  - Use the value in memory located at the register value plus a constant displacement
  - Have the constant appear before the parens
  - Eg. mov 10(%rax), %rbx

Memory Addressing Modes

- Complete
  - We have a constant displacement, a starting point, an offset, and constant to scale the offset by…
  - **D(Rb, Ri, S)**
    - The address at Rb + Ri*S + D, where S and D are constant and Rb and Ri are registers
  - Eg. movq 10(%rax, %rbx, 4), %rcx
  - If the displacement is 0 or the scale is 1, you may leave them out

Lea src, dest

- Lea: Load Effective Address
- Take the address expression from src, and save it to dest
- Do not access memory, just compute the address from the offsets, index, base, and scale, and then save the computed address in dest
- Can also be used to quickly add registers and store the result in a third register

RFLAGS

- A special purpose register that stores some status about the executed instructions
- Different bits tell us different things
- Instructions may set those bits depending on what has happened
  - These include arithmetic instructions like add or sub, as well as instructions like cmp

RFLAGS

Flag	Meaning
ZF	Result was 0
SF	The most significant bit of the result
CF	Set if the result borrowed from or carried out of the most significant bit
OF	Overflow for signed arithmetic

- The CPU doesn’t know if operands are signed or unsigned
- So, it calculates both the signed overflow (OF) and the unsigned overflow (CF) for each instruction
  - That is, OF is set assuming both are signed
  - CF is set assuming both are unsigned

How to decide whether there is overflow?

- Unsigned int:
  - you can look at whether there is a carry/borrow of the MSB
- Signed int:
  - for machine:
    - if there is carry-in but no carry-out of MSB
    - or, there is no carry-in but there’s carry out of MSB
  - for human:
    - if two positive numbers add to a negative number
    - or, two negative numbers add to a positive number

Instructions set/read RFLAGS

- Instructions that set RFLAGS
    - Regular arithmetic instructions
        - add, sub, imul, inc
    - Special flag-setting instructions
        - cmp, test
- Instructions that read RFLAGS
    - Instructions that read RFLAGS to set register values
        - Set
    - Instructions that read RFLAGs to set %rip
        - jmp

cmp

- Same as sub (*dest-src*), except it doesn’t store the result in dest
- It does, however, still change the RFLAGS I just mentioned
- This makes it useful for comparisons and conditions

jmp

- jmp label
  - Continues executing from the label, unconditionally
  - label is where to jump to
  - It acts like goto in C

Conditional Jumps

- je label
  - Jump if ZF is set
- jne label
  - Jump if ZF is not set
- jg label
  - Jump if ZF is not set and SF and OF are the same
- jl label
  - Jump if SF and OF are not the same
- ja label
  - Jump if CF and ZF are both not set

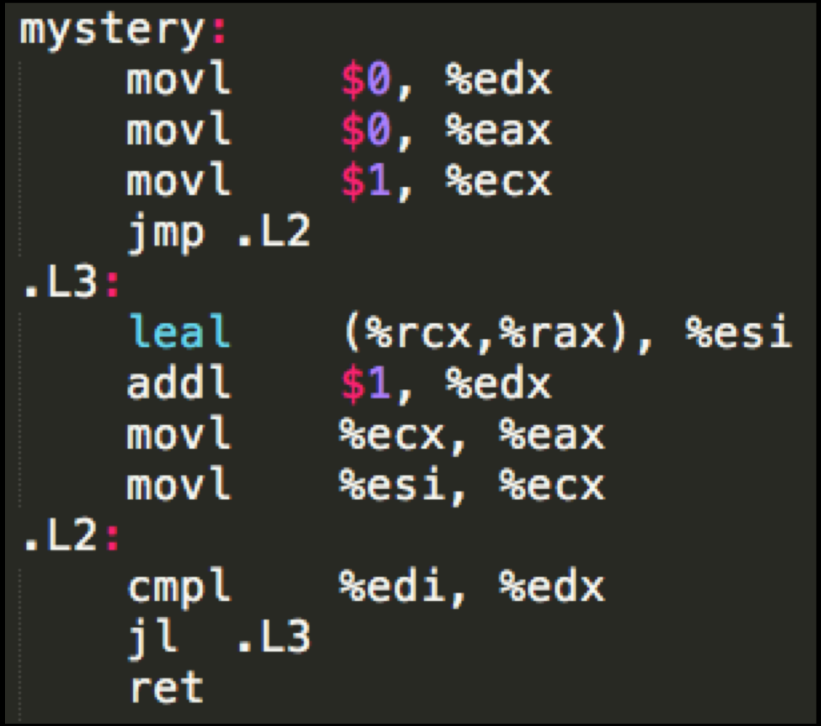
Exercises

Lab3 -- Uncover the mystery

- Very much like a puzzle game
- In this lab, we give you 5 object files, ex1_sol.o, ex2_sol.o, ..., ex5-sol.o, and withhold their corresponding C sources
- Each object file implements a particular mystery function (e.g. ex1_sol.o implements the function ex1)
- We ask you to deduce what these mystery functions do based on their x86-64 assembly code
- Write the corresponding C function that accomplishes the same thing

Exercise

- Try to figure out what the assembly code does, and write C code that does the same thing in `main.c`.



```
mystery:
    movl    $0, %edx
    movl    $0, %eax
    movl    $1, %ecx
    jmp .L2
.L3:
    leal    (%rcx,%rax), %esi
    addl    $1, %edx
    movl    %ecx, %eax
    movl    %esi, %ecx
.L2:
    cmpl    %edi, %edx
    jl  .L3
    ret
```