CS061: Machine Organization & Assembly Language Lab 9

Agenda

1. Presentation:
    a. x86-64 Crash Course
    b. Compiler Optimizations
    c. Out-of-order Execution
    d. Godbolt Demo
    e. Lab Descriptions

2. Work Time / Questions / Demos

Lab Overview

- Use Godbolt (an online compiler) to view the x86-64 assembly code of C & C++ programs.

- Explore how assembly code changes between:
  - Compiler Optimization Levels
  - C and C++
  - Loop Unrolling

```c
#include <stdio.h>

int main() {
    for (int i = 0; i < 3; i++) {
        printf("Hello, World!\n");
    }
}
```

Output... Filter... Libraries + Add new...

```asm
.LC0:
        .string "Hello, World!"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 0
.L3:
        cmp     DWORD PTR [rbp-4], 2
        jg      .L2
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        add     DWORD PTR [rbp-4], 1
        jmp     .L3
.L2:
        mov     eax, 0
        leave
        ret
```

x86-64 Crash Course

- x86 is a separate assembly language with a separate ISA
  - **ISA**: Instruction Set Architecture
  - **CISC**: Complex Instruction Set Computer

- x86 has 16 registers that hold 64 bits per register!
  - LC-3 has 8 registers that hold 16 bits per register.

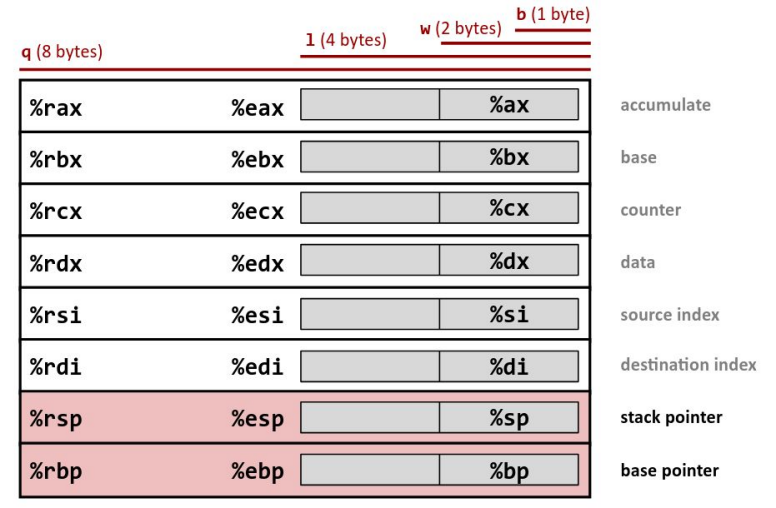
- x86 has a lot of instructions (> 1000 instructions)
  - LC-3 only has 15 instructions.

- x86 instructions are not fixed-length!
  - LC-3 instructions are always 16-bits wide.

- x86 has a built-in stack!
  - Had to create our own stack in LC-3.

x86-64 Registers

- x86 has 16 registers each 64-bits wide
- Some x86 registers are named!
  - E.g. %rax, %rcx, %rdx, %rbx
  - E.g. %r8, %r9, %r10
- Can access portions of a register too!
  - *%rax* is the entire 8-byte (64-bit) register.
  - *%eax* access the first 4-bytes (0-3) of the *%rax* register!
  - *%ax* access the first 2 bytes of the *%rax* register!
  - *%al* access the first byte of the %rax register



x86 Instructions

- x86 has a lot of instructions (> 1000)!

- Example Instruction format:
  - Instruction <Destination Register> <Source Register>
  - Same as LC-3!
  - Not the only syntax: Intel vs AT&T

- Example instructions:
  - add <dest reg> <src reg>
    - Add the destination register with the source register.
    - add rax, rcx : rax += rcx

  - mov <dest reg> <src reg>
    - Move source register value to destination register.
    - mov rbp, rsp : rbp = rsp

  - push rbp : Push rbp register onto the stack

x86 Control Flow

- Control flags that are used by control flow instructions.
  - SF (Sign Flag): Set if result is negative (similar to n register in LC-3)
  - ZF (Zero Flag): Set if result is zero.
  - OF (Overflow Flag): Set if 2's complement overflow has occurred.
  - CF (Carry Flag): Unsigned overflow.

- Instructions that set flags:
  - cmp <reg 1> <reg 2>
    - Performs Reg 1 - Reg 2 and sets control flags.
  - test <reg 1> <reg 2>
    - Performs reg 1 & reg 2 and sets control flags.
    - Useful for testing if reg 1 == reg 2.

x86 Control Flow

- Lots of jump instructions!

- Examples:
  - jmp label: Jump to the label.
  - je/jz label: Jump if result is equal/zero (ZF = 1).
  - jg/jnle label: Jump if result is greater (signed >).

- Combined:
  - test rax rbp
  - je label
  - Jumps to label if rax == rbp.

Compiler Optimizations

- Optimization Levels:
  - How hard should the compiler try to optimize your code?
  - No optimization: -O0
  - Optimization Level 1 (-O1): Minor optimizations.
  - Levels 1 to 3

- Loop Unrolling:
  - Say we have a for loop that runs 3 times.
    - Each time it executes an add instruction.
  - Unrolling the loop means we just have 3 add instructions!
  - Why do this?

```
for (i = 0; i < 3; i ++):
    x = x  + 10
```

↓

```
x = x + 10
x = x + 10
x = x + 10
```

All is not as it seems!

EXTRA INFO - Not on the labs / exams!

- Myth: Code executes sequentially.
  - Sort-of!
  - Compilers can reorder your code!
  - CPU can execute instructions *out-of-order*!

- How can CPU handle if-statements and loops if instructions out of order?
  - CPU tries to predict what will happen! (*Branch Prediction)*
  - Right Prediction: Awesome!
  - Wrong Prediction: Costly!
    - Has to trash the instructions it tried to process out of order.

Exercise Overview

- **Exercise 1:**
  - Comparing optimization levels -O1 to -O3 for a C snippet.
- **Exercise 2:**
  - Comparing number of lines generated by C and C++.
  - Demangling Identifiers.
- **Exercise 3:**
  - Comparing normal generated code versus code with loops unrolled.

Demo Info

- **Lab Grade Breakdown:**
  - 3 points for attendance.
  - 7 points for demoing (+1 bonus point demo'd before/during Friday).
  - 3 point penalty if lab is demo'd during the next lab session.

- **Tips before you demo:**
  - ***Understand your code!*** (Know what each line does & the input/output)
  - ***Test your code!*** (Check for correct output and that there are no errors)