CMPT 295

Unit - Machine-Level Programming

Lecture 11 – Assembly language basics: Practice and DEMO
-> `leaq` and arithmetic & logical instructions and memory addressing modes

Why did the programmer quit their job?

A never got arrays!

Summary

- `leaq` - load effective address instruction
- Using data as operand to an instruction:
  - Immediate (constant integral value)
  - Register (16 registers)
  - Memory (various memory addressing modes)
    - General Syntax: **Imm(r_b, r_i, s)**
- Arithmetic & logical operations
  - Arithmetic instructions: **add*, sub*, imul* inc*, dec*, neg*, not***
  - Logical instructions: **and*, or*, xor***
  - Shift instructions: **sal*, sar*, shr***

1. **Absolute**
2. **Indirect**
3. **"Base + displacement"**
4. **2 indexed**
5. **4 scaled indexed**

Today's Menu

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, `move` operation
  - Memory addressing modes
- Operation `leaq` and Arithmetic & logical operations
- Conditional Statement – Condition Code + `cmov*`
- Loops
- Function call – Stack
- Array
- Buffer Overflow
- Floating-point operations

Practice and DEMO!

Demo

1. `gcc` uses `leaq` for addition -> `sum_store.c`
2. Writing our own assembly code (`arith.s`) using arithmetic instructions of x86-64 assembly language
3. `makefile`
   - when we compile our own `*.s` files with `*.c` files
   - when we compile only `*.c` files
4. How would `gcc` compile our `arith.c` into `arith.s`?

Summary

- Demo
  - Observation: C compiler will figure out different instruction combinations to carry out the computations in our C code

Next lecture

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, `move` operation
  - Memory addressing modes
- Operation `leaq` and Arithmetic & logical operations
- Conditional Statement – Condition Code + `cmov*`
- Loops
- Function call – Stack
- Array
- Buffer Overflow
- Floating-point operations