CMPT 295

Unit - Instruction Set Architecture

Lecture 22 – Introduction to Instruction Set Architecture (ISA) + Design
Last Lecture

➤ What is a buffer overflow
  ➤ When function writes more data than the space allocated to array can hold
  ➤ Effect: data kept on the stack (value of other local variables and registers, return address) may be corrupted

  ➤ Stack smashing

➤ Why buffer overflow spells trouble -> it creates vulnerability
  ➤ Hacker attacks

➤ How to protect system against such attacks
  ➤ Avoid overflow vulnerabilities by always checking array bounds
    ➤ Use “safe” library routines
  ➤ Employ system-level protections
    ➤ Randomized initial stack pointer
    ➤ Non-executable code segments
  ➤ Use compiler (like gcc) that makes use of stack “canary” value
Today’s Menu

- Instruction Set Architecture (ISA)
  - Definition of ISA
  - ISA design
  - ISA evaluation
    - Strategies to improve our ISA
- Execution of machine instructions
  - Intro to logic design + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions
    - Hazards
So far ... The Big Picture

C program (main.c) -> C Compiler
Assembly (main.s) -> Assembler
Object file (main.o) -> Linker
Executable program (a.out) -> CPU
Instruction Set Architecture (ISA) - 1

A formal specification of ...

- What is visible (accessible) to programmer/compiler
  - Registers
    - How many
    - Size
    - Purpose
  - Memory (Memory model)
    - Word size
    - Memory Size $\rightarrow 2^m \times n$
      - $2^m$ distinct addressable locations in main memory
      - each of these addressable locations has $n$ bits

- Set of instructions
  - Description (semantic)
  - Format (template and Operand model)
  - Syntax of these instructions
    - Their binary encoding as a distinct bit pattern, i.e., machine instructions
  - Memory addressing modes

How CPU stores and accesses data

How CPU operates on this data
A formal specification of ... (cont’d)

- Set of data types
- Conventions
  - How control flow and data are passed during function calls
  - How registers are preserved during function calls
    - callee and caller saved registers
- Model of computation
  - Sequential
  - CPU executes the following simple loop
    - DO FOREVER:
      - Fetch next instruction from memory into CPU
      - Decode the instruction
      - Update the program counter
      - Execute the instruction
Goals when designing an ISA

1. Each instruction of IS must have an unambiguous encoding
   - CPU can unambiguously decode and execute them

2. IS is functionally complete -> i.e., it is “Turing complete”
   1. Data transfer instructions
   2. Data manipulation instructions
   3. Program control instructions

3. Size of IS -> **CISC** versus **RISC**
   - Complex Instruction Set Computing
   - Large # of instructions including special purpose instructions
   - Reduced Instruction Set Computing
   - Small # of general purpose instructions
   - smaller machine instruction set
   - simpler CPU design
1. “Each instruction must have an unambiguous encoding”

- Symbolic representation of machine instruction
- Advantage: human readable, i.e., it makes a program easier to read by using ...
  - Mnemonics: abbreviation of operation name
    - Example: mov, add, ret
  - Labels to represent addresses
    - Example: call sum, jmp loop

As an example, let’s use the ISA that defines x86-64

- Each assembly instruction has a corresponding machine instruction
- Expressed as bit pattern (binary encoding)
- Templates:
  - opcode operand, operand
  - opcode operand
  - opcode

As an example, let’s use the ISA that defines x86-64
What is an **opcode**? What is an **operand**?

- **Operation Code**
- **Opcode**: an instruction that can be executed by the CPU
  - It is a bit pattern
  - In machine language, this bit pattern can be represented either as a binary or hexadecimal value
  - In assembly language, this bit pattern is represented using a mnemonic
  - In the output of the `objdump` tool, we can see both representations of opcodes: hexadecimal value and mnemonic
- **Example**: `addq $8, %rsp`
  - `addq` is the opcode
  - `$8` and `%rsp` are the operands
- **Operand(s)**: required by the opcode in order to successfully carry out the instruction
  - They are bit patterns
Let’s try –
Let’s design an instruction format for an ISA

- ISA design principles
  1. Assign a unique opcode to each instruction
  2. When creating formats (templates) to encode instructions, create as few of them as possible
  3. Fields in different formats that have the same purpose must appear in the same position in the format
Let’s try – Let’s design an instruction format for an ISA

- Registers and Memory model
  - # of registers ->
  - Size -> $2^m \times n = 2^{12} \times 16$
  - Size of memory address (# of bits) ->
  - Word size ->

- Set of Instructions
  - Our first assembly instruction: ADD a, b, c
    - Semantic: $M[c] \leftarrow M[a] + M[b]$
    - Format (template): opcode src, src, dest

- Size of opcode ->
- Length of instruction ->
- Maximum # of instructions in the set ->
- Order of operands ->
- Memory addressing mode -> Direct
- Set of data types ->
Let’s try – Let’s design an instruction format for an ISA

- ISA with “3 Operand” model – Memory only
- Let’s add more instructions to our very basic set of instructions:
  - Assembly instruction: \texttt{SUB} a, b, c
    - Semantic: $M[c] \leftarrow M[a] - M[b]$
    - Format:
  - Assembly instruction: \texttt{MUL} a, b, c
    - Semantic: $M[c] \leftarrow M[a] \times M[b]$
    - Format:
Summary

- Instruction Set Architecture (ISA)
  - A formal specification of ...
    - Registers and Memory model
    - Set of instructions
    - Set of data types
    - Conventions
    - etc...

- ISA design
  - Each instruction must have an unambiguous encoding
  - Functionally complete (Turing complete)
  - Instruction sets may be large (CISC) or small (RISC)
  - Design principles when creating set of instructions
Next Lecture

- Instruction Set Architecture (ISA)
  - Definition of ISA
  - ISA design
  - ISA evaluation
    - Strategies to improve our ISA
- Execution of machine instructions
  - Intro to logic design + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions
    - Hazards