CMPT 295

Unit - Instruction Set Architecture

Lecture 23 – Introduction to Instruction Set Architecture (ISA)

+ ISA Design + ~~MIPS~~

Last Lecture - 1

- What is a buffer overflow
  - When function writes more data in array than array can hold on stack
  - 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
  - Allowing hacker attacks
- How to protect system against such attacks
  1. Avoid creating overflow vulnerabilities in the code that we write
     - By always checking bounds and calling "safe" library functions that consider size of array
  2. Employ system-level protections
     - Randomized initial stack pointer and non-executable code segments
  3. Use compiler (like `gcc`) security features:
     - Stack "canary" value and **`endbr64`** instruction

s/w developer

system

compiler

Last Lecture - 2

Brief look at …

- Floating point data and operations
  - Data held and manipulated in XMM registers
  - Assembly language instructions similar to *integer* assembly language instructions we have seen so far

- Optional: Storing Data in Various Segments of Memory
  - Global variables => data segment
  - Local variables => stack segment
  - How their values are represented in an assembly program

Today's Menu

- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design guidelines
  - Example of an instruction set: MIPS
  - Create our own instruction sets
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards

Reference

- Computer Organization and Design, 5th Edition, by David A. Patterson and John L. Hennessy 2014
  - See Resources for a link to an online version
  - Chapter 2 – Instructions: Language of the Computer
  - Chapter 4 – The processor

- Chapter 4 of our textbook ?

  → will not make use of this chapter very much!

The Big Picture – Above the hood

C code

```
short abs( short aNumber ) {

    short result = 0;

    if ( aNumber > 0 ) result = aNumber;
    else result = -aNumber;

    return result;
}
```

Assembly code

```
    .globl  abs
abs:
    movl    %edi, %eax
    testw   %di, %di
    jle     .L3
.L2:
    ret
.L3:
    negl    %eax
    jmp     .L2
```

Machine code

```
1111100010001001
11111111100001010110110
0000001001111110
1100001111110011
1101100011110111
1111101011101011
```

C program (.c) → sum_store.c

Preprocessed Source → sum_store.i

C Preprocessor

C Compiler

Assembly program (.s) → sum_store.s

Assembler

Object (.o) → sum_store.o

Executable → ss

Linker

Loader

ISA - Instruction Set Architecture

An agreement establishing how software communicates with CPU.

Computer executes it

CPU

Memory

The Big Picture - Under the hood!

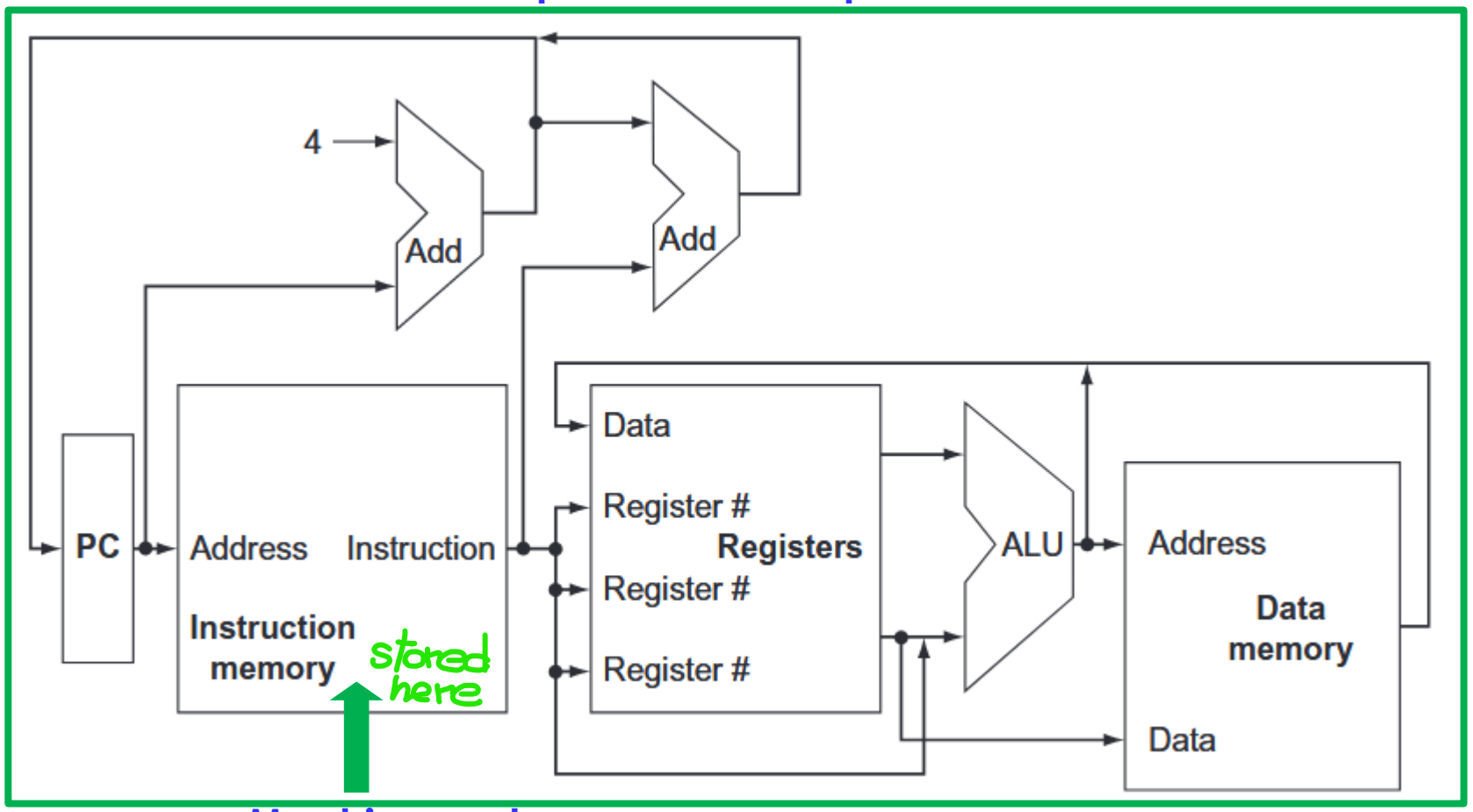
Wikipedia says:

A **datapath** is a collection of functional units such as

- ALU (perform data processing operations),
- registers, and
- buses (allow data to flow between them).

Along with the control unit, the **datapath** composes the central processing unit (CPU/microprocessor).

Microprocessor datapath



Machine code

```
1111100010001001
1111111110000101011001l0
0000001001111110
1100001111110011
1101100011110111
1111101011101011
```

Instruction Set Architecture (ISA)

- **Instruction set architecture** (ISA): defines the machine code (i.e., instruction set) that a microprocessor reads and acts upon as well as the memory model.

  Adapted from https://en.wikipedia.org/wiki/Computer_architecture#History

- **Instruction Set**: it is all the commands understood by a given computer architecture.

  Source: Computer Organization and Design, 5th Edition, by David A. Patterson and John L. Hennessy

- We say that a microprocessor implements an ISA.

Instruction Set Architecture (ISA)

An ISA is a formal specification of …

- Memory and Registers
  - Memory
    - Word size
    - Memory size -> 2^m x **n**
      - 2^m distinct addressable locations in memory
      - each of these addressable locations has **n** bits
  - Registers
    - Number
    - Size
    - Data type
    - Purpose
- Instruction Set
  - Format
  - Syntax
  - Description (semantic)

  } of assembly instructions and their corresponding machine instructions
  - Operand model: number, order and meaning of operands
  - Memory addressing modes

Instruction Set Architecture (ISA) cont'd

An ISA is a formal specification of … (cont'd)

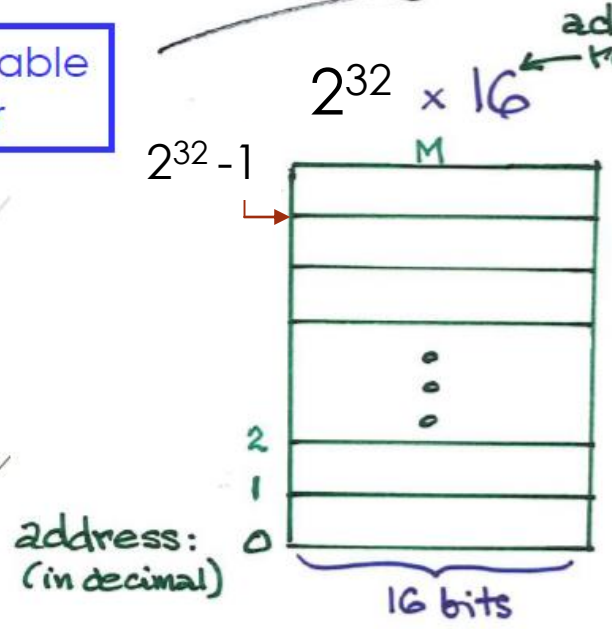
- Conventions
  - How control flow and data are passed during function calls
  - How registers are preserved during function calls
    - Any **callee** and **caller** saved registers?
- Model of computation - sequential
  - Microprocessor executes our C program in such a way that it produces the expected result
    - We get the illusion that the microprocessor executes each C statement sequentially – but as we shall see in our next unit (Chapter 5) this is not what actually happens at the CPU level.

through the fetch-decode-execute loop & PC++ to next instruction

Memory Models:

Model 1

word-addressable computer



Examples of word-addressable ISAs:

- Data General Nova minicomputer
- Texas Instruments TMS9900
- National Semiconductor IMP-16

in this model, n = wordsize

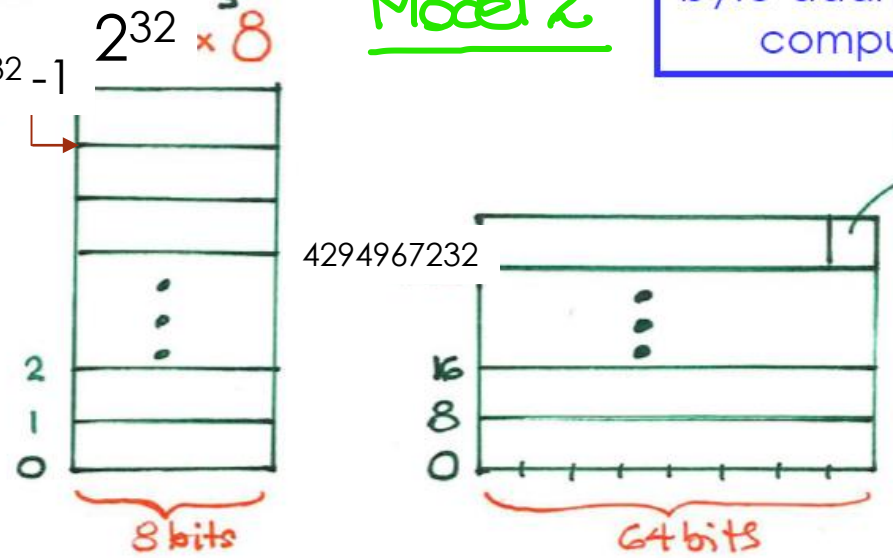
wordsize : 16 bits

Model 3

target machine → m = 64 (however, only 48 bits are used)
→ n = 8 bits

Model 2

byte-addressable computer



* uncompressed view of memory.

* compressed view of memory

in this model, n ≠ wordsize

wordsize : 64 bits.

Examples of byte-addressable ISAs:

- Intel Core i7 and i9
- AMD64 Epyc
- MIPS64
- RISC-V

address resolution: smallest addressable memory "chunk"

Example of an ISA: x86

- Memory model
  - Word size: 64 bits
  - Memory size -> 2^{m} x **n**
    - **m** = 64 bits even though only 48 bits are used
    - **n** = 8 bits (byte-addressable)
- Registers
  - 16 integers registers of 64 bits (8/16/32/64 bits can be accessed)
    - Purpose: stack pointer, return value, callee-saved, caller-saved, arguments
  - 16 floating point registers of 128 bits
- Instruction set
  - Lots of them: https://en.wikipedia.org/wiki/X86_instruction_listings
  - Operand model: two operands (of different sizes)
  - Memory addressing modes: Supports various addressing modes including immediate (direct), indirect, base+displacement, indexed, and scaled

Instruction set (**IS**) design guidelines

1. Each instruction of **IS** must have an unambiguous **binary encoding**, so CPU can unambiguously decode and execute it -> let's assign a unique opcode to each instruction
2. **IS** is functionally complete -> i.e., it is "Turing complete"
    1. Data transfer instructions — Memory reference
    2. Data manipulation instructions — Arithmetic and logical
    3. Program control instructions — Branch and jump

    3 classes of instructions
3. In terms of machine instruction format:
    a. Create as few of them as possible
    b. Have them all of the same length **and same format!**
    c. If we have different machine instruction formats, then position the **fields** that have the **same purpose** in the **same location** in the format

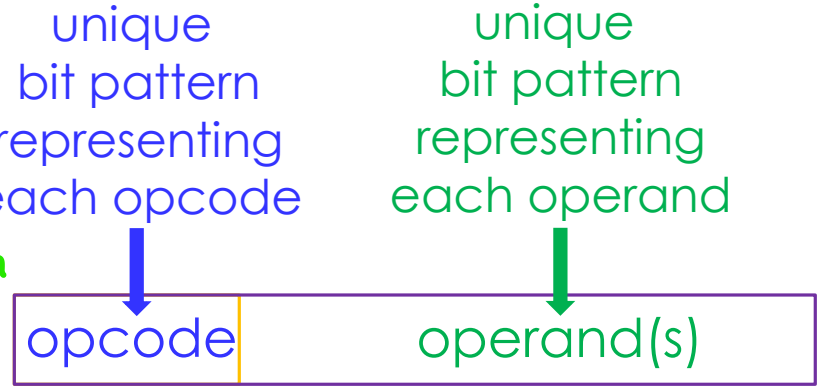
1. "Each instruction of **IS** must have an unambiguous **binary encoding** …"

Assembly instruction → compiles *(more specifically: "assembles")* into → Machine instruction

- Symbolic representation of a machine instruction
  - **Mnemonics**: abbreviation of operation name
    - Example: `movq, addw, ret`
  - **Labels** to represent addresses
    - Example:
```
call sum
jmp loop
```
- Advantage: human readable, i.e., program easier to read and write than a series of 0's and 1's,
- Made easier through the use of mnemonics and labels

- Each assembly instruction has a corresponding machine instruction
- Machine instruction expressed as bit pattern (binary encoding) –> 0's and 1's

Example of format of binary encoding (common)



What is an **opcode**? What is an **operand**?

- **Opcode**: Operation Code
- **Opcode**: operation that can be executed by the CPU
  - Expressed as bit pattern (binary encoding) –> 0's and 1's
- **Operand(s)**: required by the opcode in order for CPU to successfully carry out the instruction
  - They are also expressed as bit patterns –> 0's and 1's
- In the output of the **`objdump`** tool (disassembler), we can see opcodes and operands expressed as hexadecimal values

Example using x86-64

```
00000000004004b7 <someFcn>:
  4004b7:   4c 03 00      add     (%rax),%r8
  4004ba:   7e fb         jle     4004b7 <someFcn>
  4004bc:   c3            retq
```

```
000000000000001101001100
1111101101111110
11000011
```

∴ diff. length

Types of instruction sets

CISC

- Complex Instruction Set Computing
- Large # of instructions including special purpose instructions
- Usually "**register-memory**" architecture
- Examples: VAX, **x86**, MC68000

means: any instruction may access memory
eg: addq 8(%rsp), %rax

RISC

- Reduced Instruction Set Computing
- Small # of general purpose instructions
  - smaller machine instruction set
  - simpler microprocessor design
- "**load/store**" architecture
- Examples: SPARC, **MIPS**, Alpha AXP, PowerPC

means: load & store are only instr. to access memory

Summary

- Assembler (part of the compilation process):
  - Transforms assembly code (`movl %edi, %eax`) into machine code (`0xf889 -> 1111100010001001`)
- Instruction Set Architecture (ISA)

  A formal specification (or agreement) of …
  - Registers and memory model, set of instructions (assembly-machine)
  - Conventions, model of computation
  - etc…
- Design principles when creating instruction set (IS)
  1. Each instruction must have an unambiguous encoding
  2. Functionally complete (Turing complete)
  3. Machine instruction format: 1) as few of them as possible 2) of the same length 3) **fields** that have the **same purpose positioned** in the **same location** in the format
- Types of instruction sets: CISC and RISC

Next lecture

- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design guidelines
  - Example of an instruction set: MIPS
  - Create our own instruction sets
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards