Intro to CS—Honors I
(Instructor: Antonio R. Nicolosi)

Guest Lecturer: Zakir Akram
zakram@stevens.edu
Computer Architecture

- Programs consists of instructions & data
- Instructions & data stored in memory, loaded into registers (via memory bus)
- Instructions perform simple task, e.g.,
  - Reading data from memory into a register
  - Computing (in the ALU, “Arithmetic Logic Unit”) on registers, placing results into another register
  - Storing a register’s value back to memory
Data Movement

CPU
- ALU
- Registers

DRAM
- Instructions & Data

I/O Controller

Memory bus

I/O Devices

I/O bus
Data Movement

Instruction execution

CPU

ALU

Registers

DRAM

Instructions & Data

I/O Controller

Memory bus

I/O Devices

I/O bus
Data Movement

- CPU
  - ALU
  - Registers

- DRAM
  - Instructions & Data

- I/O Controller

- Memory bus
  - I/O data transfer

- I/O Devices

- I/O bus
Memory

- DRAM—Dynamic Random Access Memory
  - Organized as a sequence of words
- On “N-bit machine,” word size is N bits
  - Typical values are multiples of 8, e.g., 32, 64
Memory

- DRAM—Dynamic Random Access Memory
  - Organized as a sequence of words
- On “N-bit machine,” word size is N bits
  - Typical values are multiples of 8, e.g., 32, 64

![Diagram of byte and bit]

- 32-bit word
  - byte (8 bits)
Memory Addressability

- Each memory location identified by unique numerical address

- **Byte- vs. Word-addressable memory**
  - Byte-addressable: Location may be any byte
  - Word-addressable: Whole words only (i.e., intra-word bytes **not** addressable)

- Byte-addressability is the most common
  - Smaller memory units, e.g., nibble (= 4 bits) & individual bit, never directly addressable
Memory Addressability (Cont’d)

- **Word-addressable memory**

- **Byte-addressable memory**

Registers

- Special locations residing on the CPU
  - Typically word-sized
- Just a couple of dozen per chip; very fast read/write access

- Memory hierarchy
  - Registers: very few, very fast
  - DRAM memory: much larger, much slower
  - Disk storage: even larger, even slower

- Hierarchy augmentation possible (caching)
Load-Store Architecture

- ALU executes machine’s instructions using data in registers

- **Load-Store** architecture
  - Load data from memory into registers
  - Carry out arithmetic/logical operations (e.g., add, bitwise and, conditional jump)
  - Store result back to DRAM

- Very few distinct kinds of machine instructions (RISC—Reduced Instruction Set Computer)
Computer Programs

- Program can exist in different formats
  - source code (e.g., Java, C++, assembly)
  - executable (i.e., machine instructions)
  - intermediate (compilation) formats

- Compilation: Translation process, converting source into executable
  - object code: machine instructions module, not yet linked into an executable
  - bytecode (Java peculiarity): “higher-level” machine instructions for an idealized computer—the Java Virtual Machine (JVM)
Traditional Compile/Link Process

- Source 1
  - compile
  - Object 1
  - link
  - Executable
- Source 2
  - compile
  - Object 2
  - link
  - Executable
Java Compile/Link Process

Source 1

compile

Bytecode 1

Object 1

link

Executable

Source 2

compile

Bytecode 2

Object 2
Java Compile/Link Process

Source 1

```
compile
```

Bytecode 1

```
"Interpretation" or "just-in-time" (JIT) compilation
```

Object 1

```
link
```

Bytecode 2

```
compile
```

Source 2

Object 2

Executable
Executable Programs

- A program in executable format is a sequence of instructions + data
  - (Active) instructions say what to do to (passive) data
- During execution, instructions & data stored in distinct portions of memory
  - Instruction: Text segment
  - Data: Data segment (Stack, BSS, Heap)
Program Execution

- Program execution occurs according to rules that are hardwired to each specific architecture (microcode).
- Details vary, but all hardware executes programs via Fetch-Execute loop.
  1. Fetch next instruction from text segment.
  2. Execute the instruction (i.e., perform the operation on the ALU, DRAM, or I/O subsystem specified by the instruction).
  3. Repeat from step #1.
Executing Instructions

- What do instructions say to do?
  - **Load** instructions say, *e.g.*, copy word(s) from memory location at address $X$ into register $R_1$
  - **Register** instructions say, *e.g.*, take operands from registers $R_1$ and $R_2$, compute result (*e.g.*, add operands), place result in register $R_3$
  - **Store** instructions say, *e.g.*, copy result from register $R_3$ back to memory location $Y$
Memory vs. Registers

Q: Why copy from memory into registers first?

A: Because register-to-register operations can be made very fast

- There are very few registers (in the tens) . . .
- but very many memory words (in the billions)

The RISC modus operandi

Copy small parts of the program data into registers for brief periods; operate on it; replace with other parts once done with that data
Q: Why copy from memory into registers first?
A: Because register-to-register operations can be made very fast
  - There are very few registers (in the tens) . . .
  - but very many memory words (in the billions)

The RISC modus operandi
Copy small parts of the program data into registers for brief periods; operate on it; replace with other parts once done with that data
Sample MIPS Instructions

`lw r6,20(sp)` — load word into register 6; the word is located 20 bytes beyond the address stored in the “sp” register

`jr r12` — the next instruction to be executed is at the address stored in register 12 (jump to register)

`div r4,r9` — divide the number in register 4 by the number in register 9; leave quotient in register 4 (remainder goes to special register)
Data Representation I: Binary/Octal/Hexadecimal Notation