

# ECE 131 – Programming Fundamentals

## Module 1, Lecture 3: Background–Computer Hardware and Software

Dr. Daryl Lee

University of New Mexico

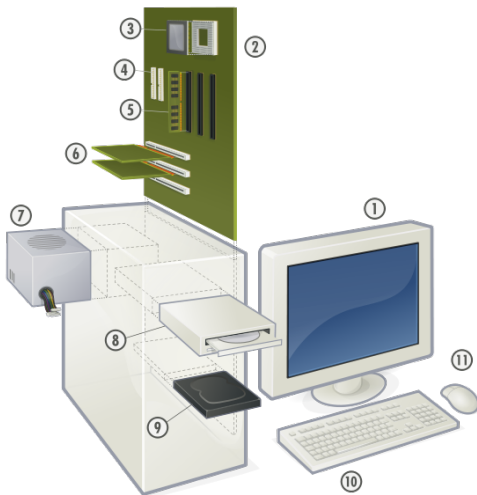


# Overview

In this lecture we'll provide a brief introduction to computer hardware and software. The goal is to make sure everyone is familiar with:

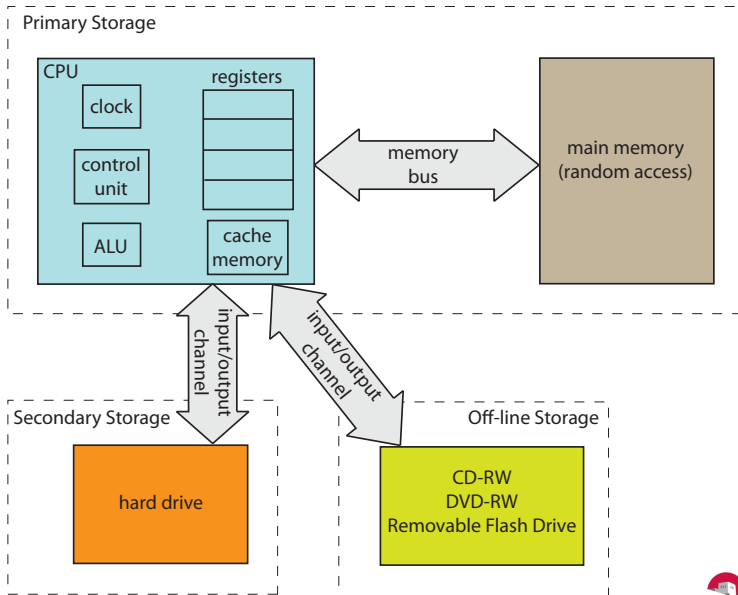
- The common terminology used in this area.
- The purpose/functionality associated with various computing components.
- How programs are executed within a computing environment.

# Hardware Components of a Modern Computer



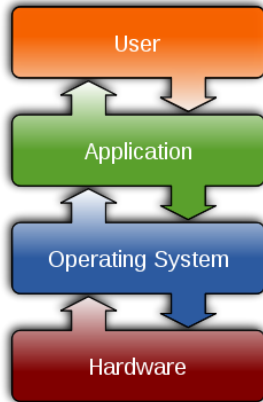
- ① Monitor
- ② Motherboard
- ③ Microprocessor (CPU)
- ④ ATA Slots – for storage devices.
- ⑤ Main memory
- ⑥ Expansion Cards
- ⑦ Power Supply
- ⑧ Optical Disk Drive
- ⑨ Hard Disk Drive
- ⑩ Keyboard
- ⑪ Mouse

# Data Storage



# Software Components of a Modern Compute

- **Operating System (OS)** – Controls the entire system and isolates application programs from the computer hardware.
  - Manages the users of the computer and their privileges.
  - Schedules the processes (programs) associated with the OS.
  - Manages memory and the file system.
  - Controls peripheral devices such as printers, monitors, etc.
- **Application Programs** – Run within the environment provided by the operating system.
  - Users interact with application programs in order to make the computer do something useful.



# The Operating System – Memory Management

- The OS is responsible for allocating memory to application programs as they request it, and reclaiming this memory when it is no longer needed.
- Knowledge of how this works can help you to write more efficient programs.
- The OS uses heuristics in an effort to place the most frequently accessed data/instructions in the fastest memory.  
**Ex:** Whenever a location in main memory is accessed, copy its contents (and those around it) to cache memory.
  - **Cache hit** – a subsequent memory access finds the item it's looking for in cache.
  - **Cache miss** – a subsequent memory access does not find the item it's looking for in cache.

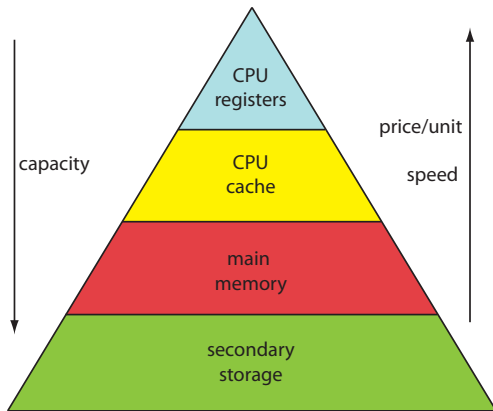
A high percentage of cache hits can significantly speed up the running time of a program.

# The Operating System – Memory Management

**Virtual memory** – a memory management technique that allows programs to have more addressable memory than actually exists in main memory.

- The addressable memory space is partitioned into **pages**.
- Those pages that will not fit in main memory are placed in secondary storage.
- Similar to cache, the memory manager attempts to keep those pages that are accessed most frequently in main memory.
- A **page fault** occurs if a memory access occurs to a page that is not in main memory.
- Too many page faults will lead to a drastic degradation in the performance of a program – an order of magnitude more costly than a cache miss.
- This is why having a large amount of main memory (RAM) can make your computer applications run a lot faster – it will lead to fewer page faults.

# The Memory Hierarchy



**Goal:** The cost of memory in a machine is dominated by the cheapest memory, but memory management by the OS leads to performance similar to that of the fastest memory.



# The Operating System – Interface

Two common ways used to interact directly with the OS:

- **Command line interpreter** (command line shell) – an application program that reads lines of text entered by a user and interprets them in the context of a given operating system.  
**Ex:** Windows NT – cmd.exe  
Vista – Windows Powershell  
Mac OS – Terminal.app  
Linux – bash shell
- **Windowing system** – a graphical user interface that provides an environment for interacting with a given operating system.  
**Ex:** Windows NT – MFC Framework  
Mac OS X – Cocoa Framework  
Linux – Gnome

# Program Execution

There are two primary ways for a program to be executed:

- **Compiler** – A program that transforms source code into the machine readable code that a CPU can execute.

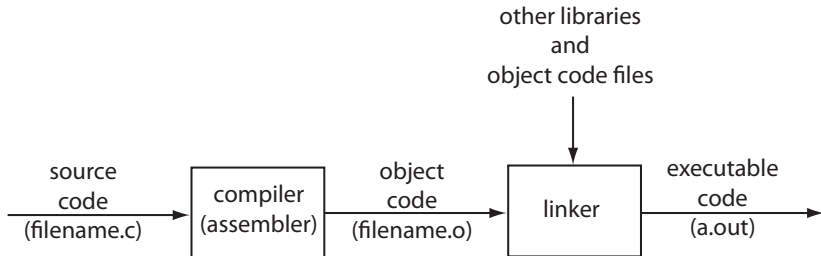
**Ex:** Programs written in C, C++, Fortran and Pascal are typically executed using a compile/link/execute cycle.

- **Interpreter** – A program that executes source code directly, line-by-line, as each line of source code is encountered.

**Ex:** Programs written in Matlab, Ruby, Perl and Python are typically executed using an interpreter.

# Compilers

Typical steps involved in creating an executable program using a compiler:



**Compiler** – translation of source code to object code.

**Linker** – translation of object code to executable code.

Your first C program:

```
#include <stdio.h>
main() {
    printf("\n\n Hello World!  \n\n");
}
```

- Using an editor, create the program above.
- Save the program in a file called `hello.c`.

# Compilers

- To compile the program, at the command prompt type:

```
gcc hello.c
```

This invokes the GNU C Compiler.

- Type:

```
ls -l
```

to list all of the files in the current directory.

- You should see a file named a.out.

To execute this file, type:

```
./a.out
```

- You should see:

```
Hello World!
```

in the terminal window.

# Compilers

- When you invoked the `gcc` program in the previous example, it performed every step in the compile/link process necessary to create an executable file.
- If you want `gcc` to only compile a source file in order to create an object code file, invoke it using:  

```
gcc -c hello.c
```

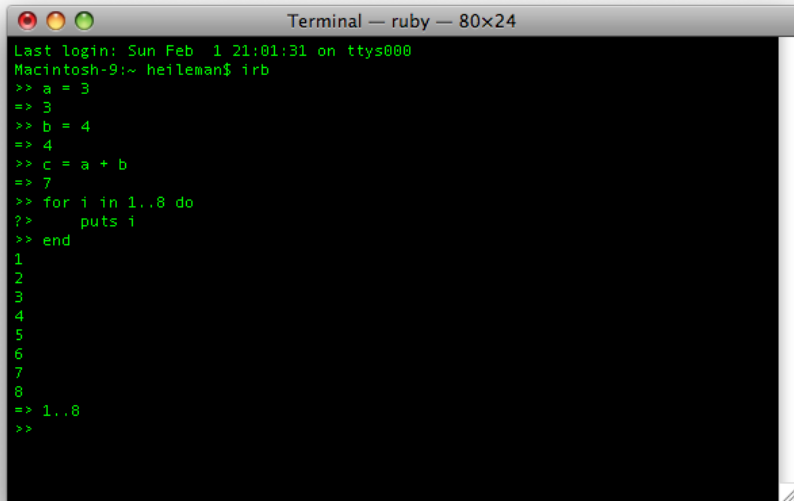
This will produce the object file `hello.o` (assuming there are no errors in your source code).
- The `-c` used above when invoking `gcc` is known as a **compiler flag**. This is a **command line option** that is supplied by the user to the `gcc` program telling it to change its default behavior.
- Using this compiler flag, we can create multiple standalone object code files (each might implement some useful functionality) and use the linker to pull them into executable programs as necessary — this is one way of **reusing** code.

# Interpreters

- An interpreter will typically translate HLL source code into some intermediate form, and then execute it immediately, without writing an object code file.
- It's often easier to develop a program using an interpreter, because you see the results of each line of code immediately, and catch errors as you go.
- Interpreters are [interactive](#). It's common to interact with an interpreter via a command line during development, and then to place all of the source code in source code file once everything is working. You can then invoke the interpreter on the source code file, and it will execute the program.

# Interpreters

## Interactive Ruby:

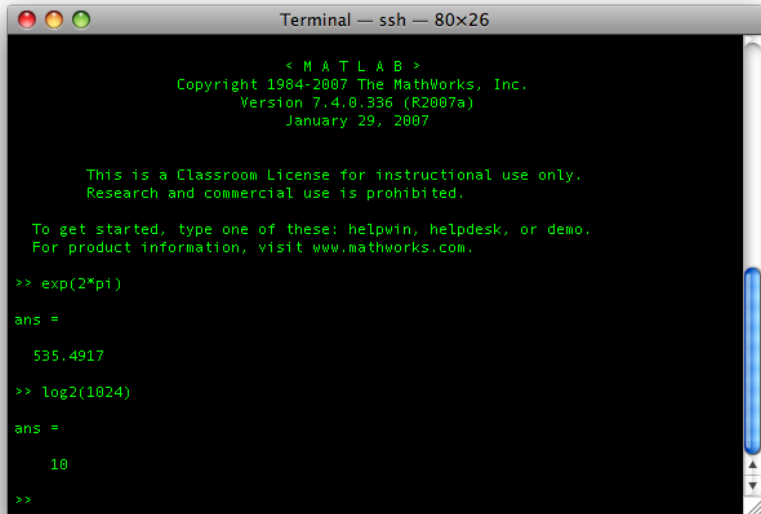
A screenshot of a macOS Terminal window titled "Terminal — ruby — 80x24". The window has a dark background with green text. It shows the output of the 'irb' command, including login information and a series of Ruby commands and their results. The commands include variable assignments, arithmetic, and a loop that prints numbers 1 through 8.

```
Terminal — ruby — 80x24
Last login: Sun Feb  1 21:01:31 on ttys000
Macintosh-9:~ heileman$ irb
>> a = 3
=> 3
>> b = 4
=> 4
>> c = a + b
=> 7
>> for i in 1..8 do
?>     puts i
>> end
1
2
3
4
5
6
7
8
=> 1..8
>>
```



# Interpreters

Interacting with Matlab from the command window:

A screenshot of a terminal window titled "Terminal — ssh — 80x26". The window has a dark background with green text. It displays the MATLAB startup sequence, including the logo, copyright information (1984-2007 The MathWorks, Inc.), version (7.4.0.336 (R2007a)), and date (January 29, 2007). It also shows a classroom license notice and instructions on how to get started. Finally, it shows two MATLAB commands being executed: `exp(2*pi)` and `log2(1024)`, with their respective outputs displayed.

```
Terminal — ssh — 80x26

< M A T L A B >
Copyright 1984-2007 The MathWorks, Inc.
Version 7.4.0.336 (R2007a)
January 29, 2007

This is a Classroom License for instructional use only.
Research and commercial use is prohibited.

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> exp(2*pi)

ans =

    535.4917

>> log2(1024)

ans =

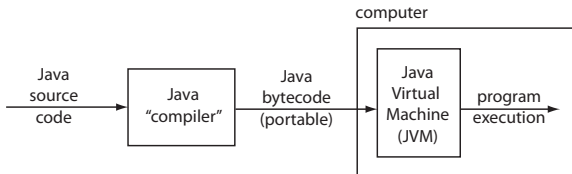
    10

>>
```

# Compilers and Interpreters

- There are actually a spectrum of possibilities between compiling and interpreting.

Ex:



- Java code is “compiled” ahead of time and stored as machine independent code called Java bytecode.
- This code is then linked at run-time (i.e., when the program is to be executed) by an interpreter contained in a Java Virtual Machine (JVM) that translates the Java bytecode into machine code.
- As long as the target computer contains a JVM, the program will execute.