Master of International Business and Entrepreneurship

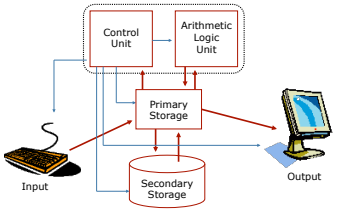
Software Core Concepts

Information Systems for Managers

Learning Objectives

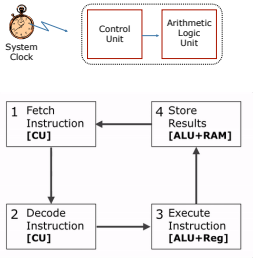
- Be able to define what software is, and be familiar with the different classes of software in use today.
- Be able to define what an interface is, and be familiar with the different types of interfaces in use today.
- Be familiar with the process of software creation.
- Be able to define what an algorithm is and be familiar with the different types of programming language in use today.
- Understand the underlying logical structure of modern software applications.

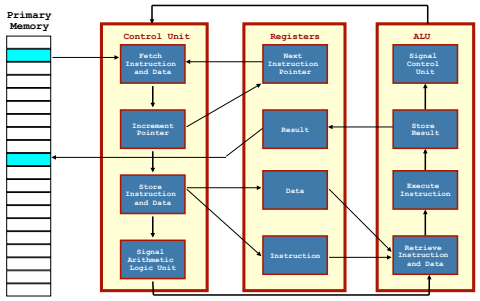
The Stored Program Concept



> The instructions which govern this operation must be given to the device in absolutely exhaustive detail. [...] Once these instructions are given to the device, it must be able to carry them out completely without any need for further intelligent human intervention.

The Processing Cycle



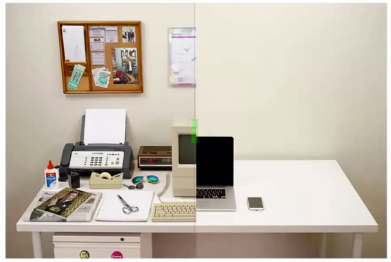


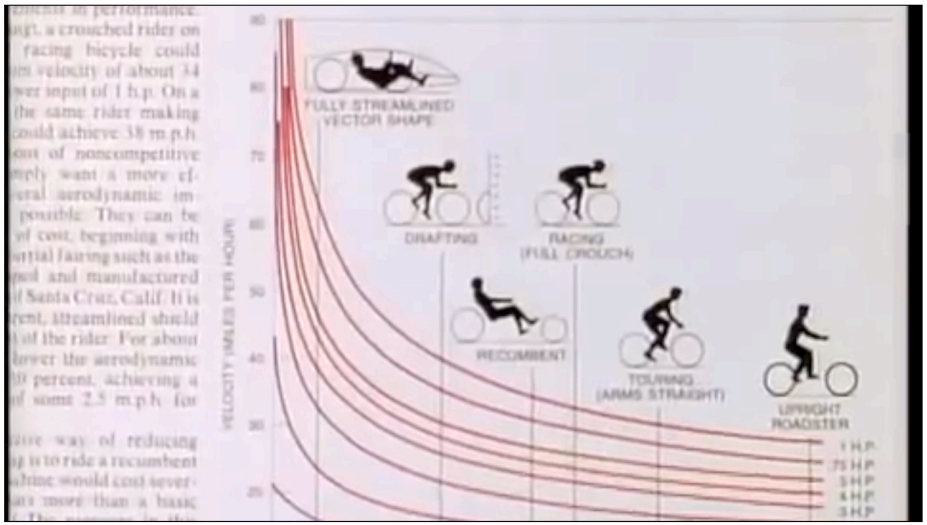
Instruction Set

- Can you upgrade the instruction set of your iPhone?
- Can you explain the difference between a software update and an instruction set change?

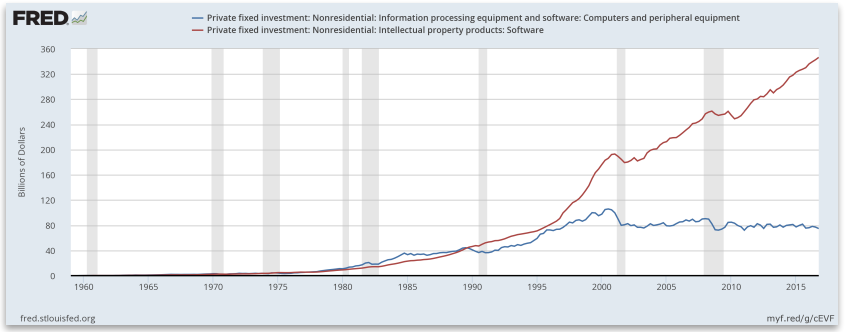
The Stored Program Concept

- Digital computers are multi-purpose devices
- Humans and computers are complementary
  - Humans are creative problems solvers
  - Computers execute tasks with speed and precision
- A computer is "like a bicycle for the mind"



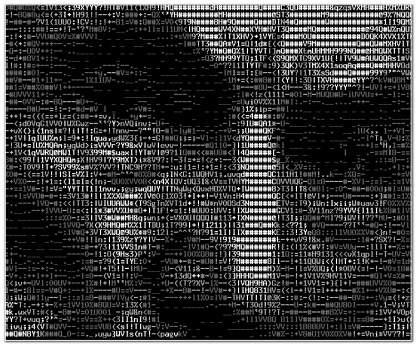


The Software Industry



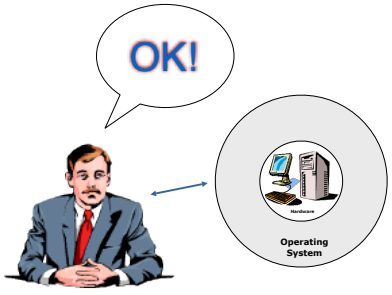
Classes of Software Programs

- Operating System
- System Software
- Application Software



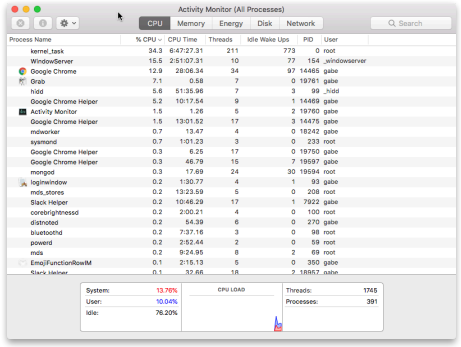
Operating System

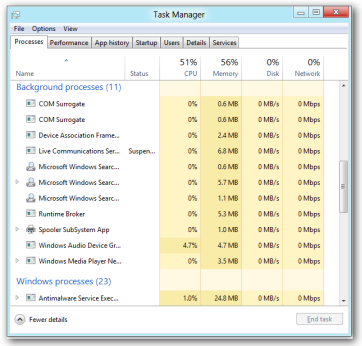
- It is the software program that *manages* a computer system and its operations
  - It allocate and assign system resources (e.g., manage user credentials, memory addressing)
  - It schedules access to resources (e.g., CPU time, printer queue)
  - It monitors system's activities (e.g,. status of jobs)



iOS ARM mbed

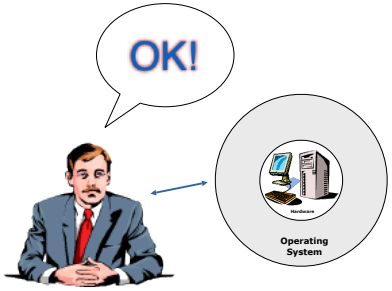
Monitor Resources





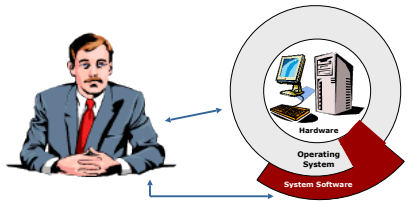
Operating System

- It is the software program that *manages* a computer system and its operations
  - It allocate and assign system resources (e.g., manage user credentials, memory addressing)
  - It schedules access to resources (e.g., CPU time, printer queue)
  - It monitors system's activities (e.g,. status of jobs)
- It is a platform for software applications
- It exposes Application Programming Interfaces (API)



System Software

- Software you wouldn't need if you didn't have a computer.
- A category of software programs designed to:
  - extend the functionality of the operating system (e.g., device drivers)
  - provide specific system management functionality (e.g, antivirus)
  - provide the facilities for software development (e.g., an Integrated Development Environment).

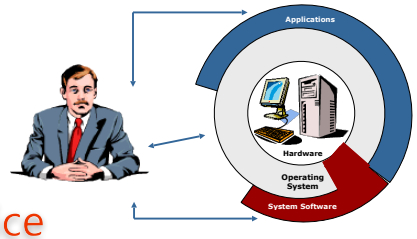


From System Software to Operating System

- Apple: Time Machine (automatic backups)
- Apple: Compression utility
- Windows: Firewall
- Windows: Disk Cleanup utility
- Android: Google Allo
- iOS: Siri

Application Software

- A software program that enables a user to perform an activity the would perform without a computer:
  - Excel - data analysis
  - Word - communication
  - GroupMe - verbal communication
  - Fortnite- entertainment
- Software Suites

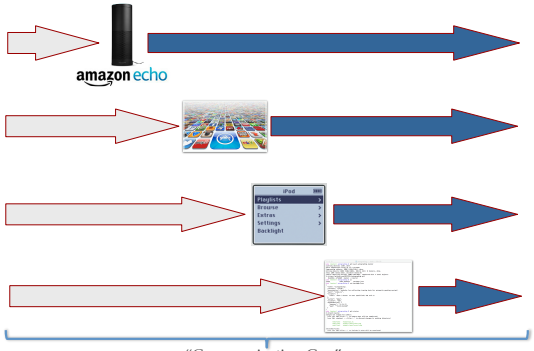


Office

User Interface

- Interface: The point of contact between two systems or two components of a system
- Types of UI:
  - Command line
  - Menu driven
  - Graphical (GUI)
  - Conversational Interfaces

User Interface Evolution



"Communication Gap"

Platforms

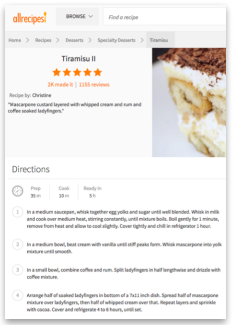
- Technical: The "environment" in which a software program is executed:
  - Hardware architecture (e.g., Intel, ARM)
  - Operating System (e.g., Wintel, iOS)
- Application Programming Interfaces: protocols that regulate how a software application can use external resources.

> "[Creating software] is like building something where you do not have to order cement. You can create a world of your own, your own environment, and never have to leave the room."

***Ken Thompson**, Pioneer of the Unix Operating System*

Programming a Digital Computer

- Software program: The set of instructions which govern the operations of a digital computer.
- Programming: The art and science of writing software programs
  - Algorithm design
  - Coding
- Algorithm: A set of rules for solving a problem, or achieving a result, in a finite number of steps.
- Algorithm design: The activity of envisioning and specifying the set of steps the computer must perform in order to produce the expected output (i.e., the algorithm).
- Coding: The activity of writing the algorithm as a set of precise instructions for the computer to execute - using a specific *programming language*.



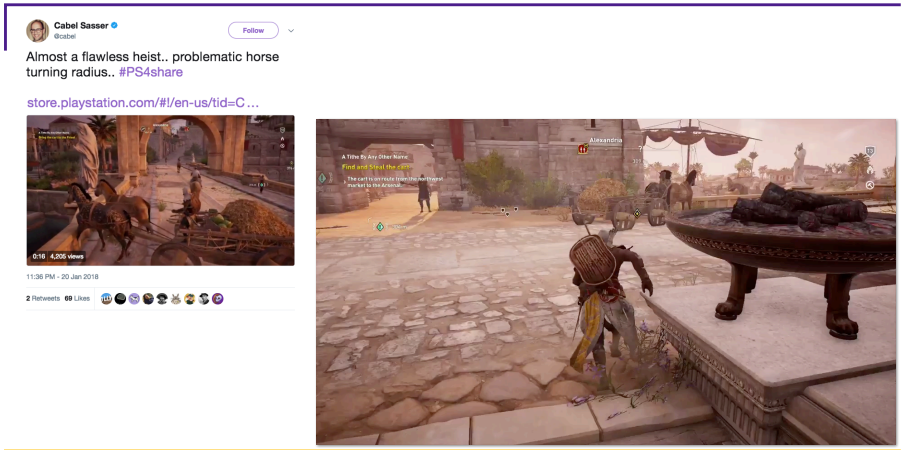
Mohammed ibn-Musa al-Khwarizmi

The Stored Program Concept

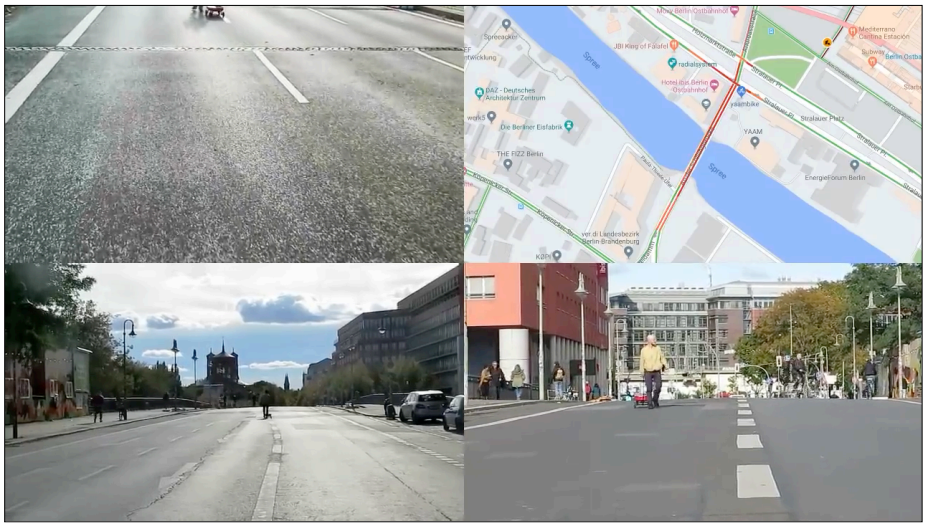
> The instructions which govern this operation must be given to the device in absolutely exhaustive detail. [...] Once these instructions are given to the device, it must be able to carry them out completely without any need for further intelligent human intervention.

John von Neumann, computer engineering pioneer

Algorithm Design Challenge



https://twitter.com/cabel/status/954980953964736512



Programming Languages Evolution

Fourth Generation Language

High Level Language

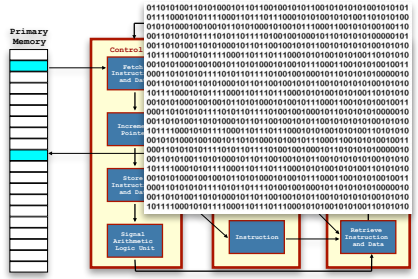
Assembly Language

Machine Language

"Communication Gap"

Machine Language

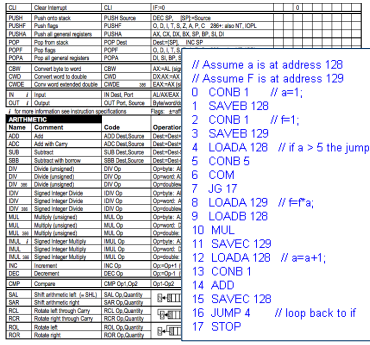
- The lowest level language
- Instructions are sequences of impulses for the CPU to process, represented as:
  - Binary representation
- Explicit memory addressing.
- Hardware specific: It maps directly to the **microprocessor's instruction set**.
- Very difficult to use. The programmer has to bridge the full 'communication gap'.
- Difficult to maintain programs over time.



Machine Language

Assembly Language

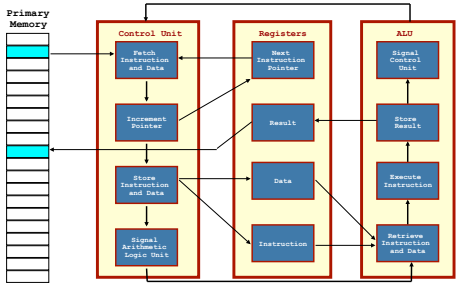
- Hardware specific low level language.
- One level of abstraction above machine language. It uses:
  - Mnemonic opcodes: Abbreviations to represent instructions (e.g., ADD, SUB)
  - Symbolic addresses: "meaningful" identifiers for memory locations (e.g., AX, BX, DI, SI).
- Requires knowledge of the hardware but easier than machine language (e.g., ADD AX BX).
- Assembler: The software program that converts assembly language code into machine language.
  - Source code: Human readable software program
  - Object code: Machine executable software program



```
// Assume a is at address 128
// Assume F is at address 129
0  CONB 1      // a=1;
1  SAVEB 128
2  CONB 1      // f=1;
3  SAVEB 129
4  LOADA 128   // if a > 5 the jump to 17
5  CONB 5
6  COM
7  JG 17
8  LOADA 129   // f=f*a;
9  LOADB 128
10 MUL
11 SAVEC 129
12 LOADA 128   // a=a+1;
13 CONB 1
14 ADD
15 SAVEC 128
16 JUMP 4      // loop back to if
17 STOP
```

Assembly Language

"Assembly Language" Programming



High Level Languages

- Also known as Third Generation Languages (3GL)
- Higher abstraction than Assembly Language.
- One to many mapping to machine language
- Platform independent
- It still must be converted into machine language in order to be executed
  - Compiler: Source to object all at once
  - Interpreter: Source to object one line at a time

JavaScript FORTRAN C++ python Java COBOL THE C PROGRAMMING LANGUAGE

High Level Language

```python
import sys
import dlib
import cv2
from skimage import io

# Take the image file name from the command line
file_name = sys.argv[1]

# Create a HOG face detector using the built-in dlib class
face_detector = dlib.get_frontal_face_detector()

#win = dlib.image_window()

# Load the image into an array
image = io.imread(file_name)

# Run the HOG face detector on the image data.
# The result will be the bounding boxes of the faces in our image.
detected_faces = face_detector(image, 1)

print("I found {} faces in the file {}".format(len(detected_faces), file_name))

# Open a window on the desktop showing the image
#win.set_image(image)


# Loop through each face we found in the image
for i, face_rect in enumerate(detected_faces):
    # Detected faces are returned as an object with the coordinates
    # of the top, left, right and bottom edges
    print("- Face #{} found at Left: {} Top: {} Right: {} Bottom: {}".format(i, face_rect.left(), face_rect.top(),
                                                                             face_rect.right(), face_rect.bottom()))

    # Draw a box around each face we found
    #  win.add_overlay(face_rect)


r = 1000.0 / image.shape[1]
dim = (1000, int(image.shape[0] * r))
resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)

for i, face_rect in enumerate(detected_faces):
    cv2.rectangle(resized, (int(face_rect.left()*r), int(face_rect.top()*r)), (int(face_rect.right()*r), int(face_rect.bottom()*r)), (0, 255, 0), 1)


cv2.imshow("Faces found", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Fourth Generation Languages

- Higher abstraction than 3GL.
- Non-procedural English-like languages
- Outcome-driven: Programmer specifies what (the outcome) not how (the procedure)

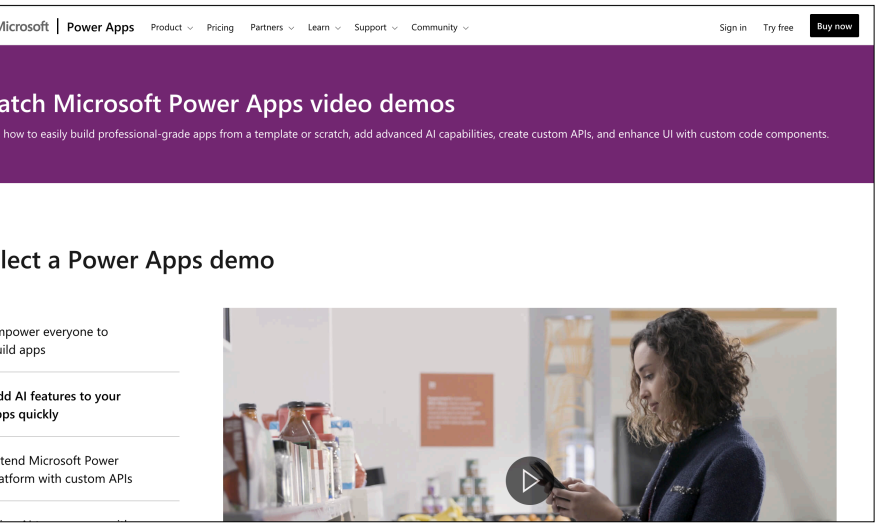
SQL (4GL):

```
SORT TABLE Accounts
ON Account_Num
```

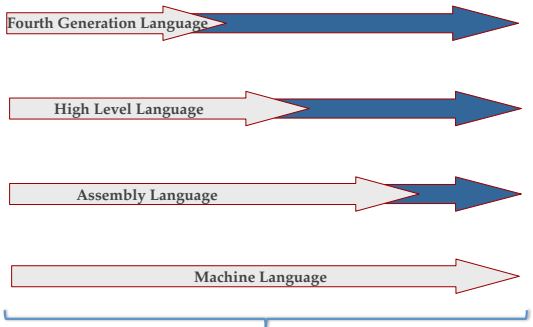
C (3GL):

```
Quicksort(left,right:integer;var arr:array[left..right] of integer);
  var m,i,j:integer;
begin
  i:=left;
  j:=right;
  m:=arr[trunc((left+right)/2)]; (* arr[left],arr[right],... *)
  repeat
    while (arr[i]<m) do i:=i+1;
    while (arr[j]>m) do j:=j-1;
    if (i<=j) then begin
      swap(arr[i],arr[j]);
      i:=i+1;
      j:=j-1;
    end;
  until (i>j);
  if (left<j)  then Quicksort(left,j,arr);
  if (i<right) then Quicksort(i,right,arr);
end;
```

Fourth Generation Language



Programming Languages Evolution



Summary

- In general, higher level of abstraction imply:
  - Greater ease of use
  - Greater portability across hardware platforms
  - Lower control by the programmer over hardware resources
  - Lower efficiency of the resulting software program

Language	Portable	Concise	Mnemonic	Procedural
Machine Language	no	no	no	yes
Assembly Language	no	no	yes	yes
High Level Languages	yes	yes	yes	yes
Fourth Generation Languages	yes	yes	yes	no
Natural Language	yes	yes	yes	no

What We Learned

- Be able to define what software is, and be familiar with the different classes of software in use today.
- Be able to define what an interface is, and be familiar with the different types of interfaces in use today.
- Be familiar with the process of software creation.
- Be able to define what an algorithm is and be familiar with the different types of programming language in use today.
- Understand the underlying logical structure of modern software applications.