# CPSC-440 Computer System Architecture

MATLAB Review

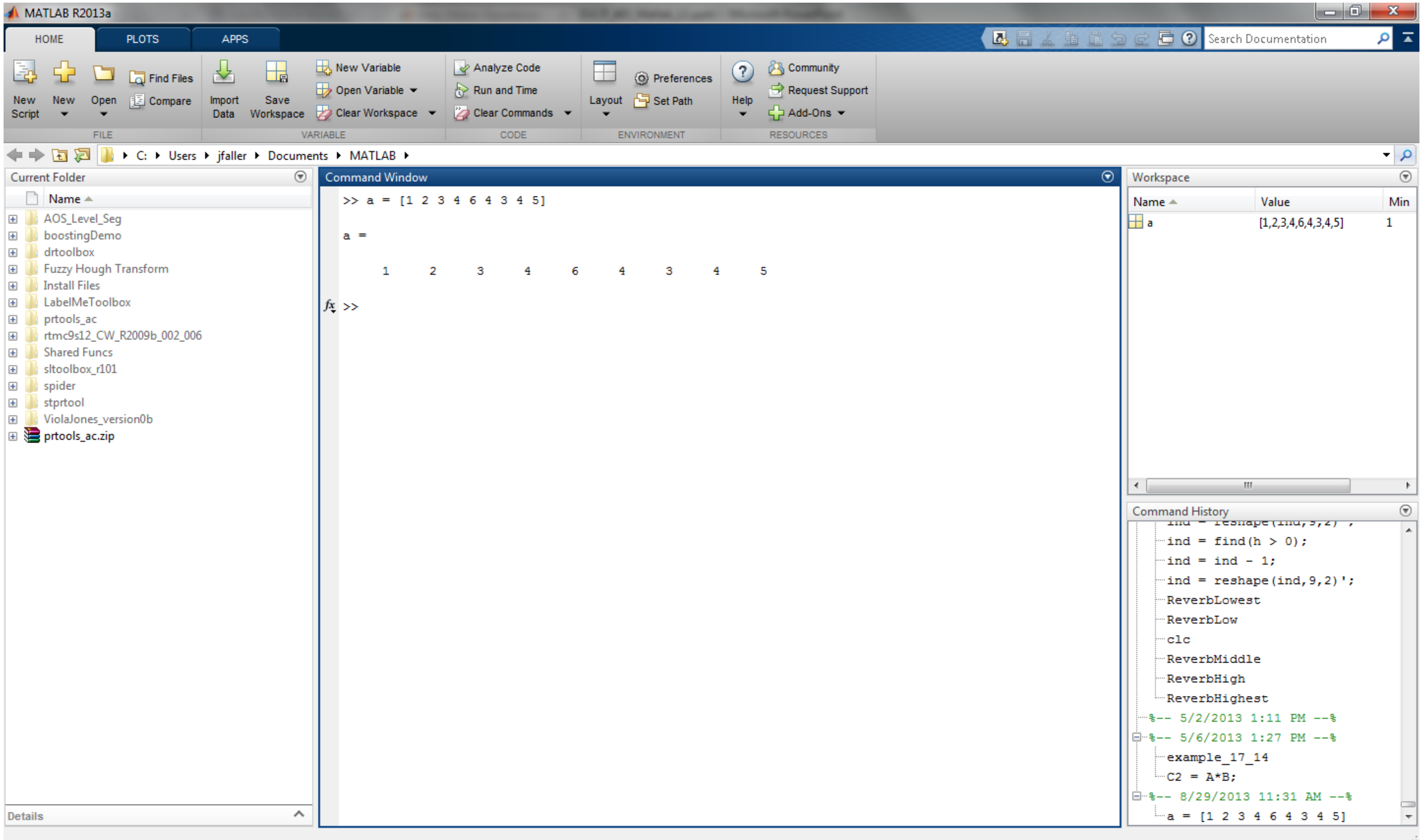
# Matlab

- Is a numerical computing environment and 4$^{th}$ generation programming language
- Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran

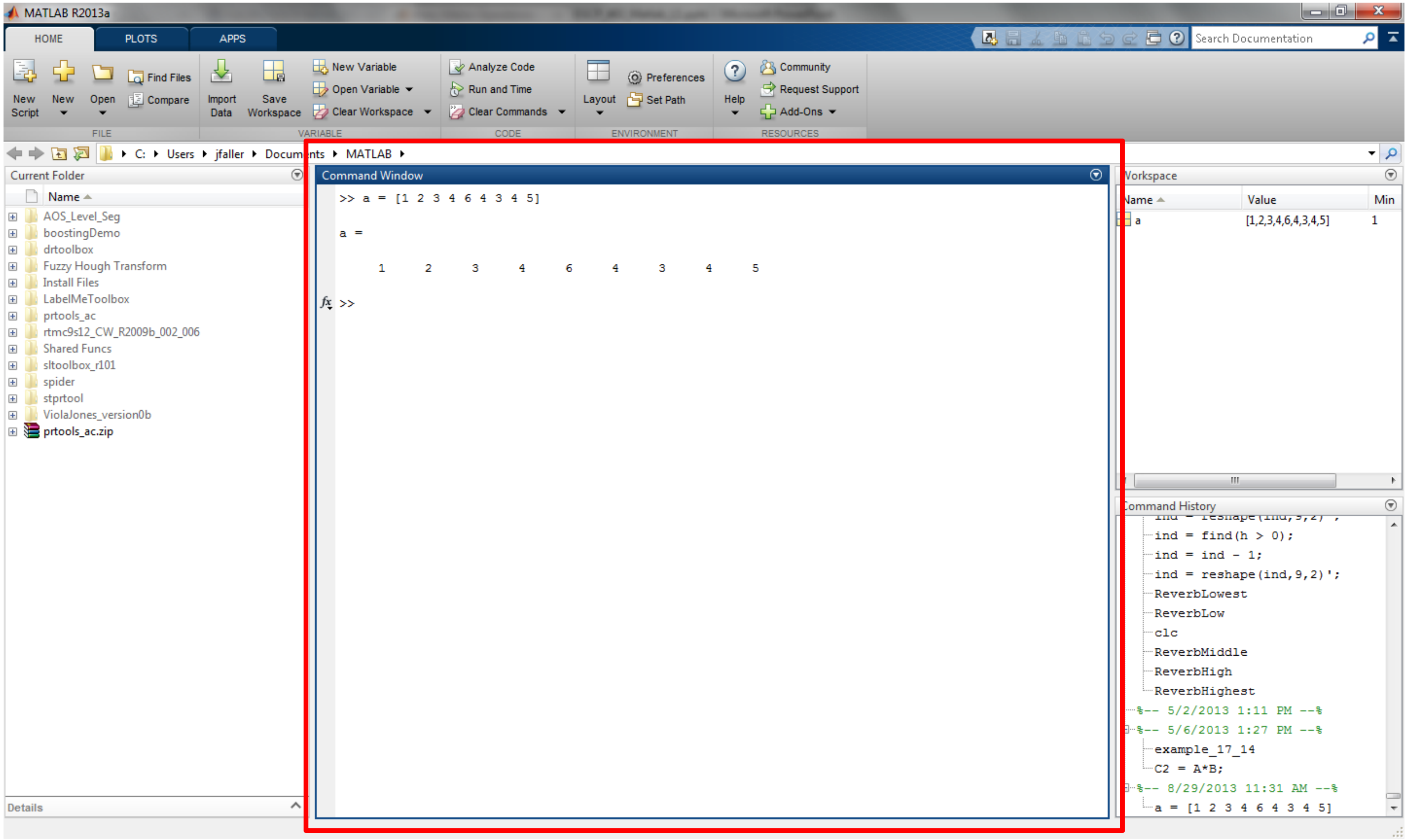
# Free Matlab for Students

- Available at CSUF IT website:
  - http://www.fullerton.edu/it/students/software/matlab/

# Matlab Default View

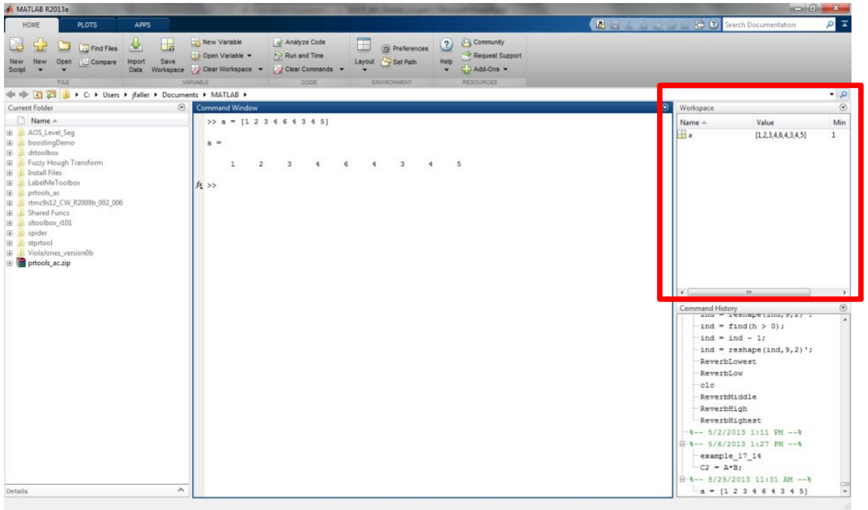


# Command Window



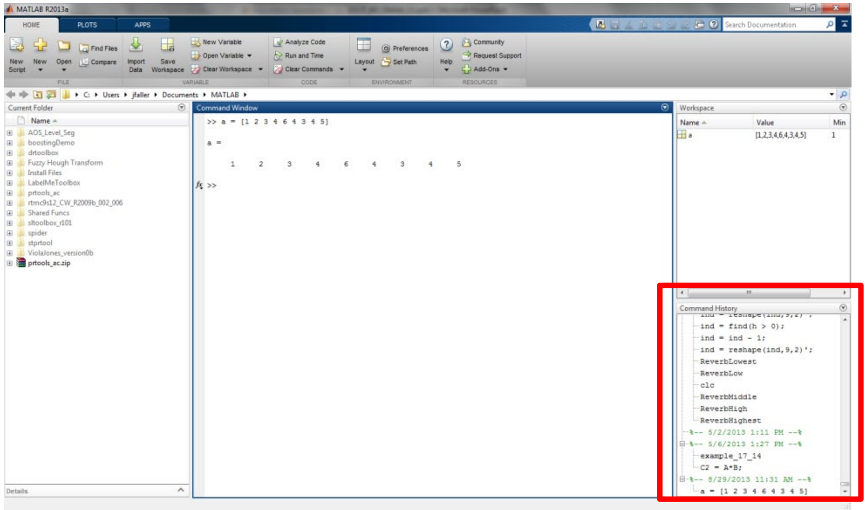
# Workspace Window

- Shows the variables currently available to you



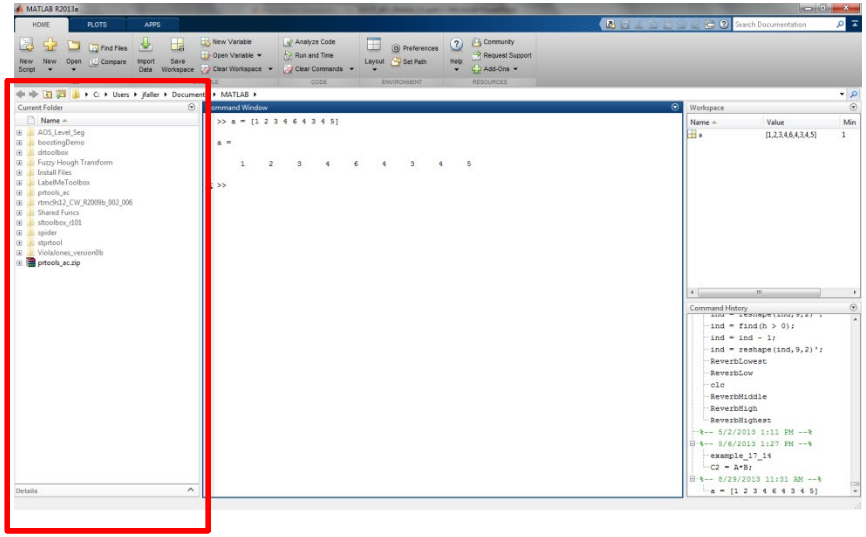
# Command History Window

- Shows the commands you have entered
- Sorted by date



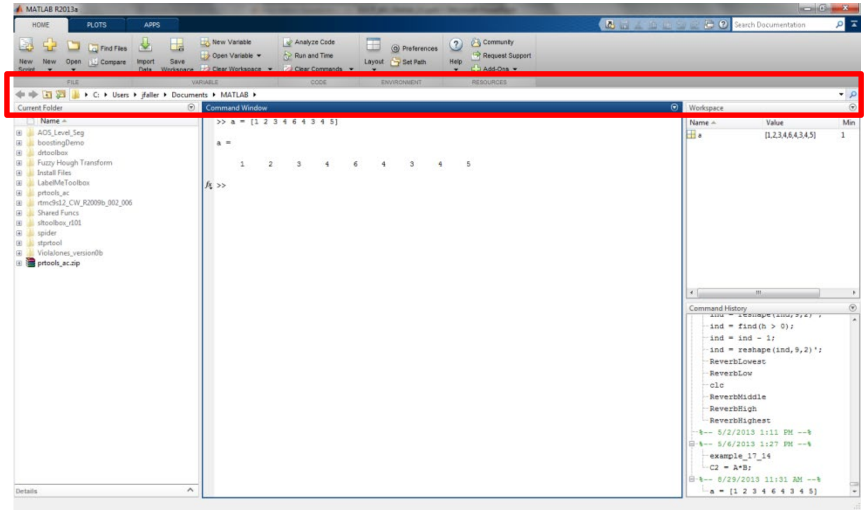
# Current Folder Window

- Shows the folders for the present working directory



# Present Working Directory

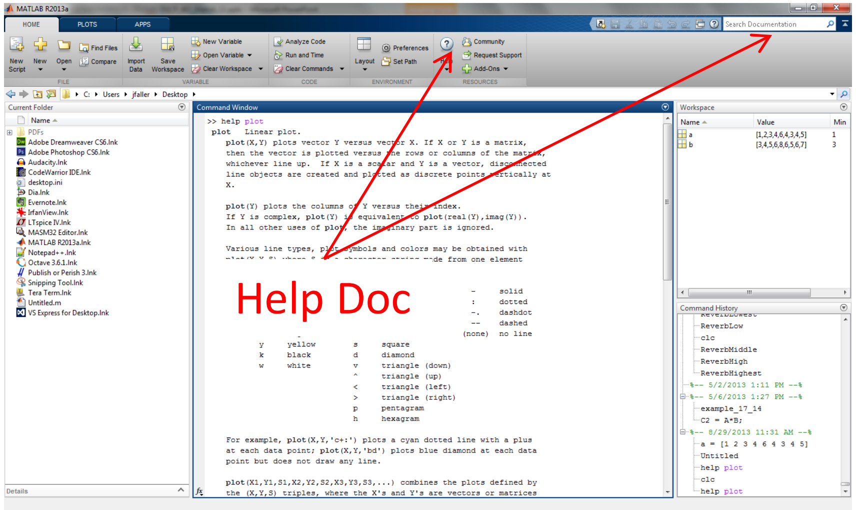
- Shows the current folder you are working in
- You can also use the command “pwd”



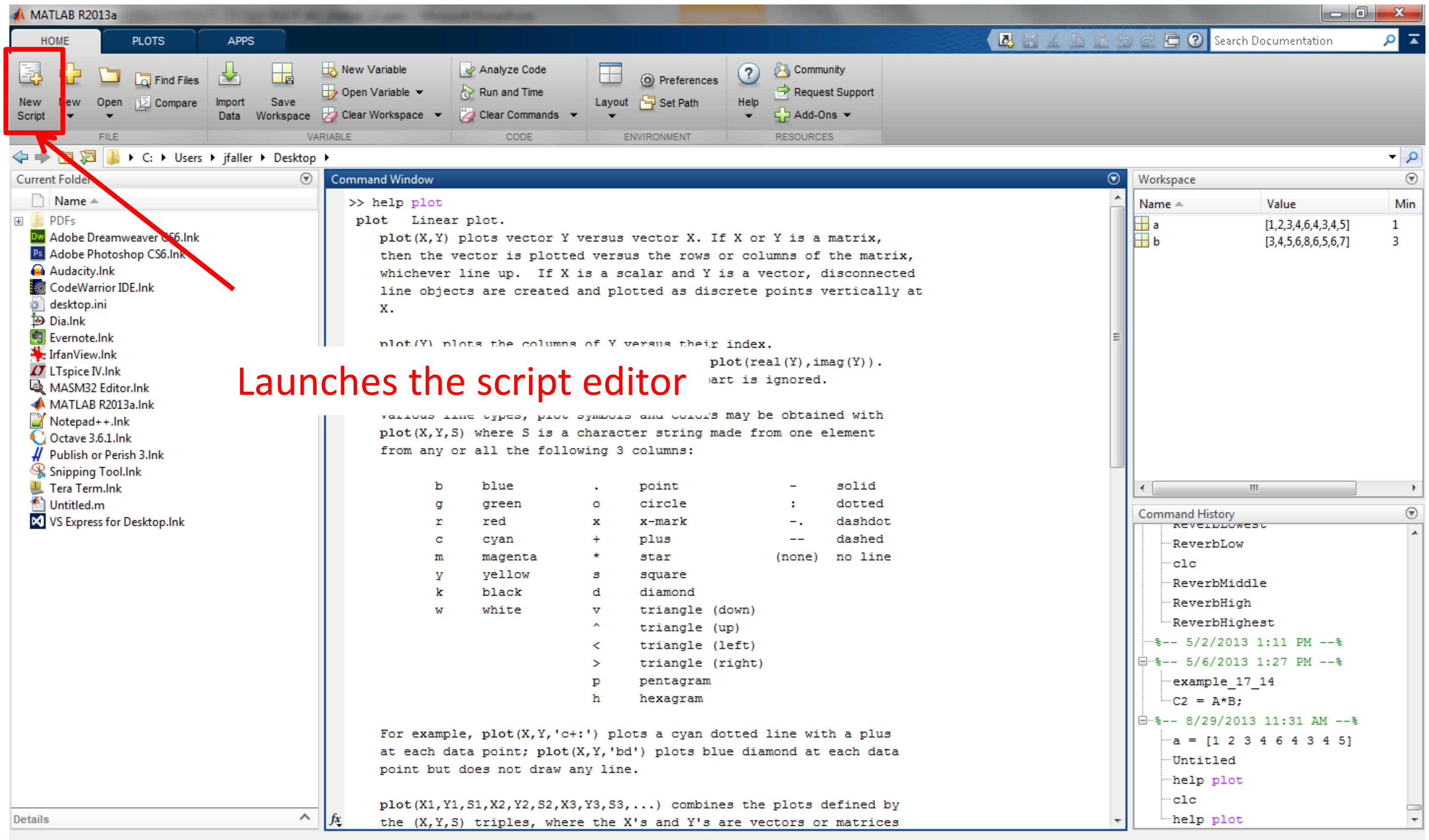
# Help Docs

- Searchable help doc
- You can also use the "help" command
- Example:

```
help plot
```

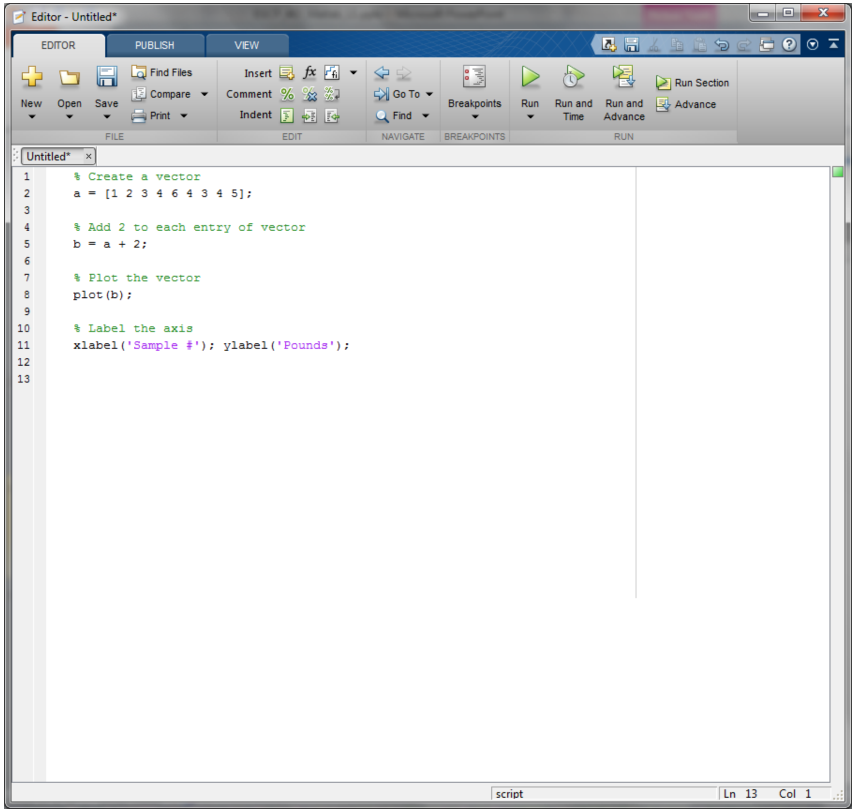


# Creating Scripts

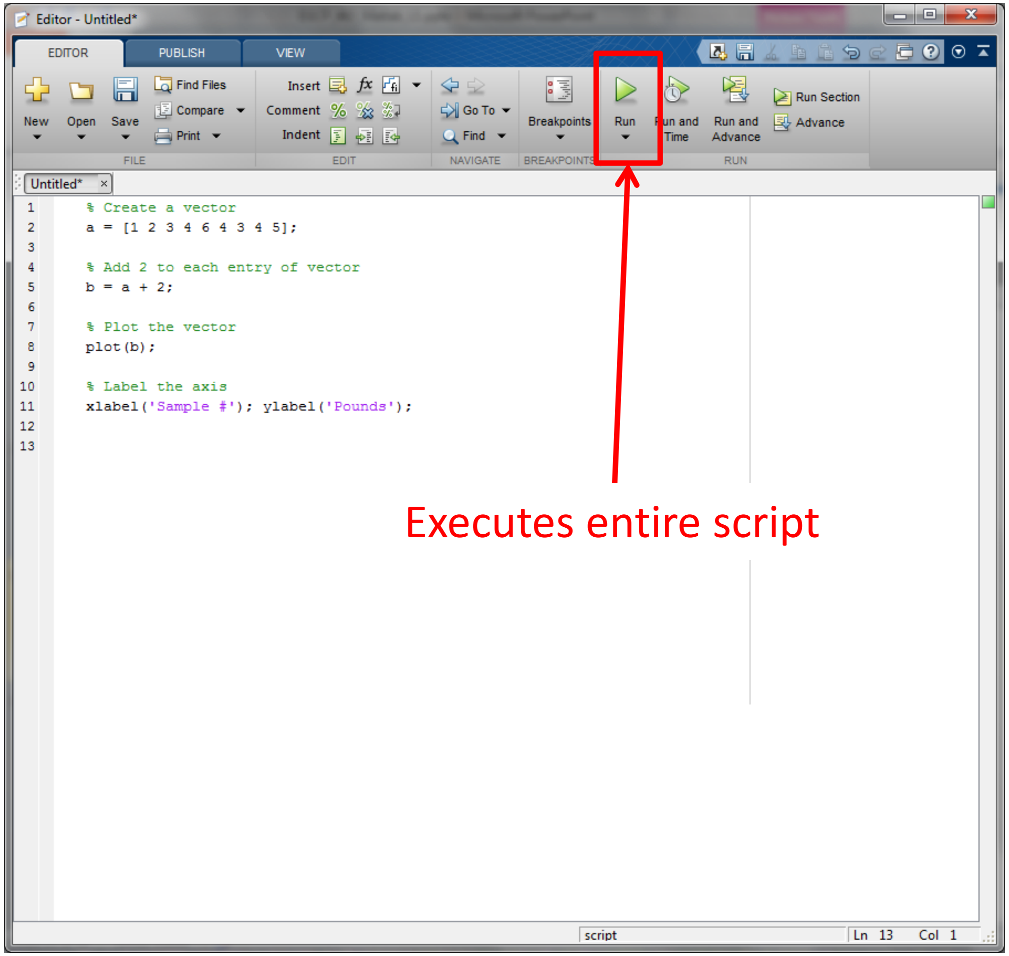


# Script Editor

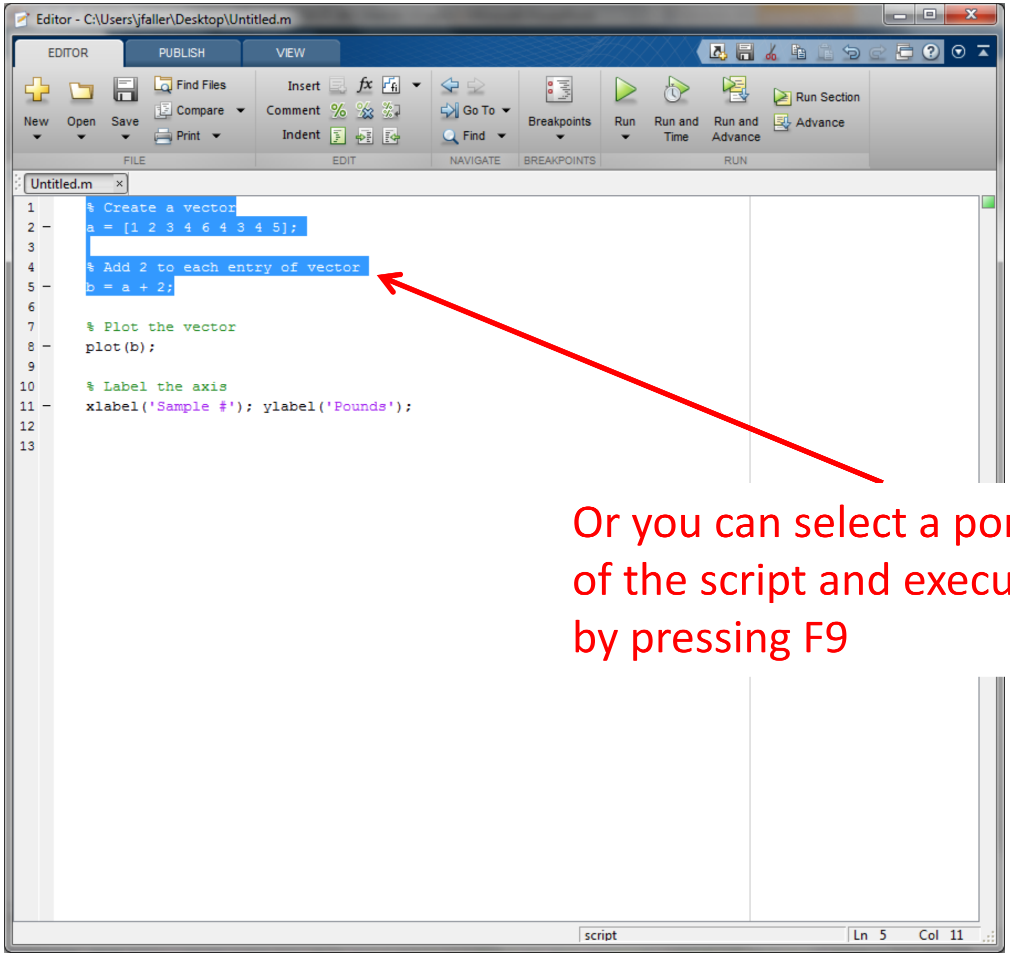
- Instead of entering in the command window directly, you can also enter commands in the script editor and save as a m-file



# Script Editor



# Script Editor




Or you can select a portion of the script and execute it only by pressing F9

# Getting Started

>> **a = [ 1 2; 2 1 ]**

a =

    1     2
    2     1

>> **a*a**

ans =

    5     4
    4     5

- Example
  - Define a matrix “a” and computed its square
  - “a times a”
- Text in bold is what you type in the command window
- Ordinary text is what Matlab outputs

# Matrices

- To enter the matrix:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- and store it in a variable “a”, do this:

  >> **a = [1 2; 3 4];**

- To redisplay the matrix, just type its name:

  >> **a**

- Once you know how to enter and display matrices, it is easy to compute with them. First we will square the matrix “a”:

  >> **a * a**

# Matrices

- Now we'll try something a little harder. First we define a matrix b:

  >> **b = [1 2; 0 1];**

- Then we compute the product ab:

  >> **a*b**

- Finally, we compute the product in the other order:

  >> **b*a**

# Matrices

- Notice that the two products are different
  - Matrix multiplication is non-commmutative
- Of course, we can also add matrices:

  >> **a + b**

- Now let's store the result of this addition so that we can use it later:

  >> **s = a + b**

# Matrices

- Matrices can sometimes be inverted:

  >> **inv(s)**

- To check that this is correct, we compute the product of s and its inverse:

  >> **s * inv(s)**

- The result is the unit, or identity matrix. We can also write the computation as

  >> **s/s**

- We can also write

  >> **s\s**

- which is the same as

  >> **inv(s) * s**

# Matrices

- To see that these operations, left and right division, are really different, we do the following:

  >> **a/b**

  >> **a\b**

- Not all matrices can be inverted, or used as the denominator in matrix division:

  >> **c = [ 1 1; 1 1 ];**

  >> **inv(c);**

- A matrix can be inverted if and only if its determinant is nonzero:

  >> **det(a)**

  >> **det(c)**

# Systems of Equations

- Now consider a linear equation

  ax + by = p

  cx + dy = q
- We can write this more compactly as

  AX = B
- where the coefficient matrix A is

  a b

  c d
- the vector of unknowns is

  x

  y
- and the vector on the right-hand side is

  p

  q
- If A is invertible, X = (1/A)B, or, using Matlab notation, X = A\B. Let's try this out by solving ax = b with a as before and b = [ 1; 0 ]. Note that b is a column vector.

  >> **b = [ 1; 0 ]**

  >> **a\b**

# Loops

- Loop Example
  - We regard x as representing (for example) the population state of an island
  - The first entry (1) gives the fraction of the population in the west half of the island, the second entry (0) give the fraction in the east half
  - The state of the population T units of time later is given by the rule y = ax
  - This expresses the fact that an individual in the west half stays put with probability 0.8 and moves east with probability 0.2 (note 0.8 + 0.2 = 1), and the fact that in individual in the east stays put with probability 0.9 and moves west with probability 0.1
  - Thus, successive population states can be predicted/computed by repeated matrix multiplication

```
>> a = [ 0.8 0.1; 0.2 0.9 ]
>> x = [ 1; 0 ]
>> for i = 1:20, x = a*x, end
```

# Graphing

## Functions of One Variable

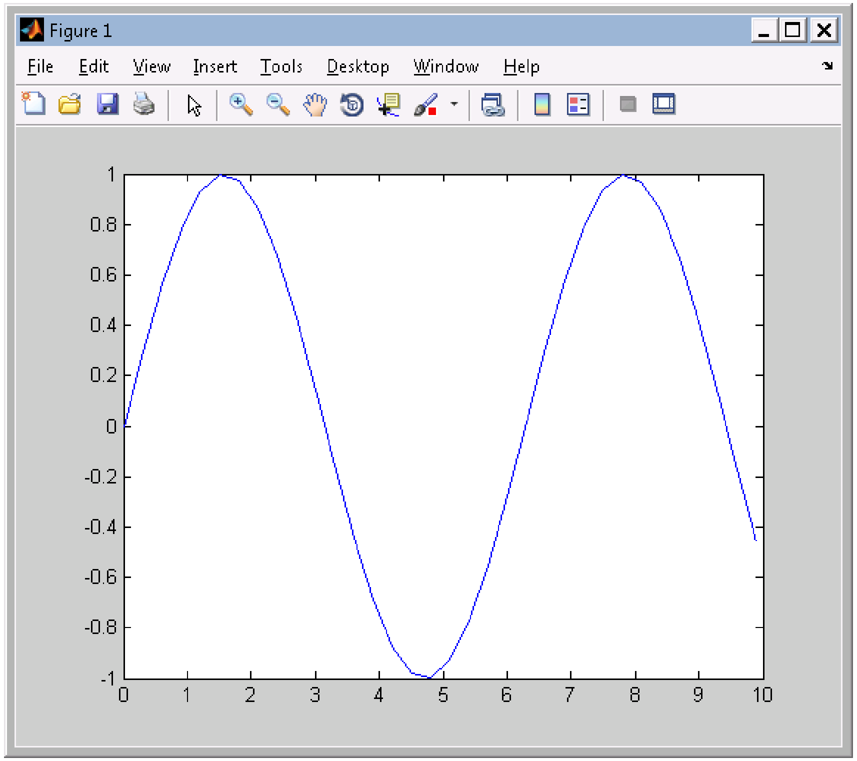
- To make a graph of y = sin(t) on the interval t = 0 to t = 10 we do the following:

  >> **t = 0:.3:10;**

  >> **y = sin(t);**

  >> **plot(t,y)**

- The command t = 0:.3:10; defines a vector with components ranging from 0 to 10 in steps of 0.3
- The y = sin(t); defines a vector whose components are sin(0), sin(0.3), sin(0.6), etc.
- Finally, plot(t,y) use the vector of t and y values to construct the graph



# Graphing

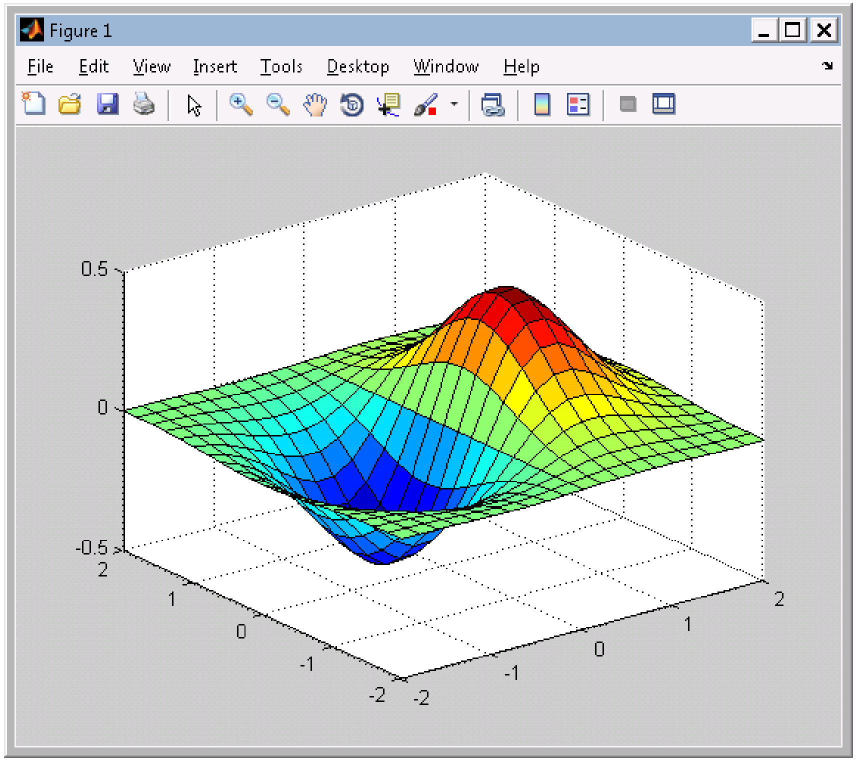
## Functions of Two Variable

- Here is how we graph the function

$$z(x, y) = xe^{(-x^2 - y^2)}$$

```
>> [x,y] = meshgrid(-2:.2:2, -2:.2:2);
>> z = x .* exp(-x.^2 - y.^2);
>> surf(x,y,z)
```

- The first command creates a matrix whose entries are the points of a grid in the square -2 <= x <= 2, -2 <= y <= 2
- The small squares which make up the grid are 0.2 units wide and 0.2 unit tall
- The second command creates a matrix whose entries are the values of the function z(x,y) at the grid points
- The third command uses this information to construct the graph



# Common Commands and Operators

- http://www.hkn.umn.edu/resources/files/matlab/MatlabCommands.pdf

# Useful Tutorials

- Download MATLAB and do the following tutorials:
  - Basic Matric Operations
  - Getting Started with MATLAB
  - Matlab Overview Video
  - Analyzing and Visualizing Data with MATLAB
  - Programming and Developing Algorithms with MATLAB
  - Signal Related Videos

# CPSC-440 Computer System Architecture

Lecture 1

Introduction

# Introduction

# Computer Architecture
# Computer Organization

**Computer Architecture**

- Attributes of a system visible to the programmer
- Have a direct impact on the logical execution of a program

**Architectural attributes include:**

- Instruction set, number of bits used to represent various data types, I/O mechanisms, techniques for addressing memory

**Computer Organization**

- The operational units and their interconnections that realize the architectural specifications

**Organizational attributes include:**

- Hardware details transparent to the programmer, control signals, interfaces between the computer and peripherals, memory technology used

# Structure and Function

- Hierarchical system
  - Set of interrelated subsystems
- Hierarchical nature of complex systems is essential to both their design and their description
- Designer needs to only deal with a particular level of the system at a time
  - Concerned with structure and function at each level
- Structure
  - The way in which components relate to each other
- Function
  - The operation of individual components as part of the structure

# Function

- A computer can perform four basic functions:
  - Data processing
  - Data storage
  - Data movement
  - Control

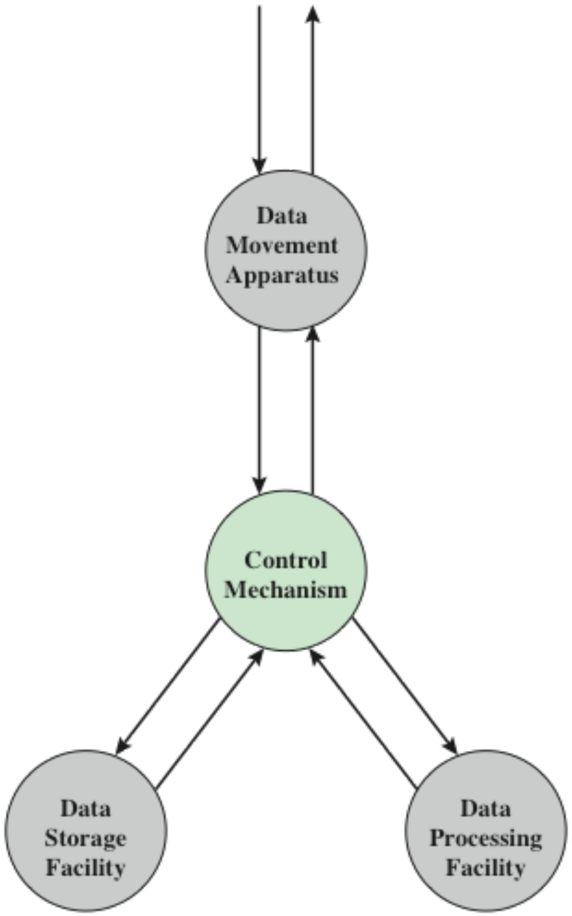


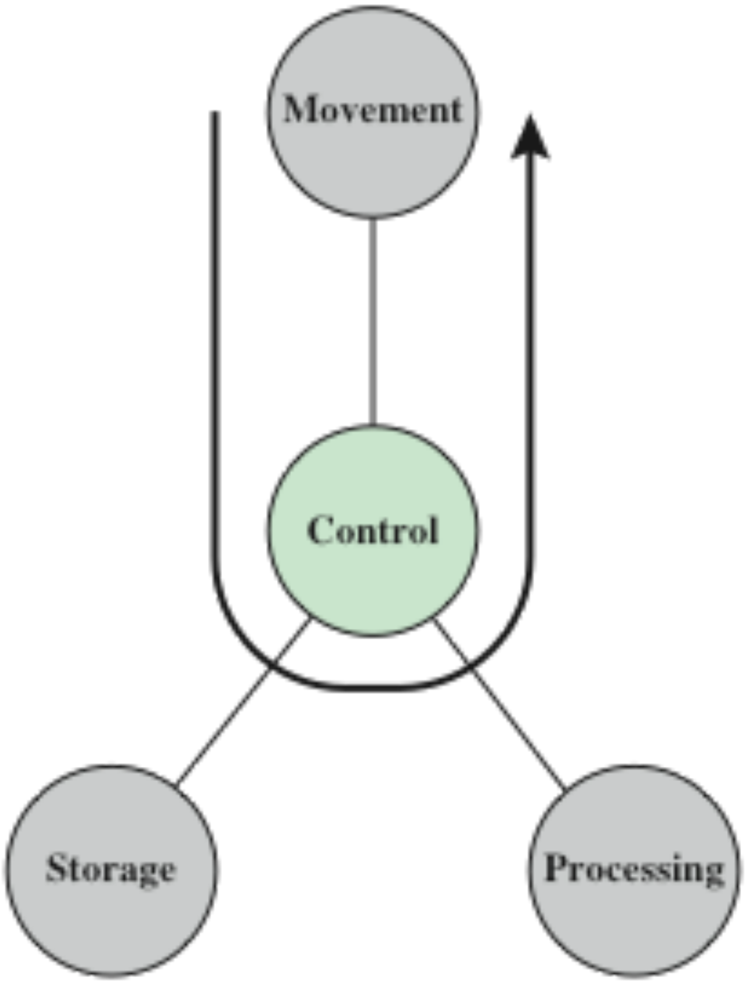

**Figure 1.1 A Functional View of the Computer**

+

## Operations

**(a)**

**Data Movement**




(a)

Figure 1.2 Possible Computer Operations

## Operations

**(b)**

**Data Storage**

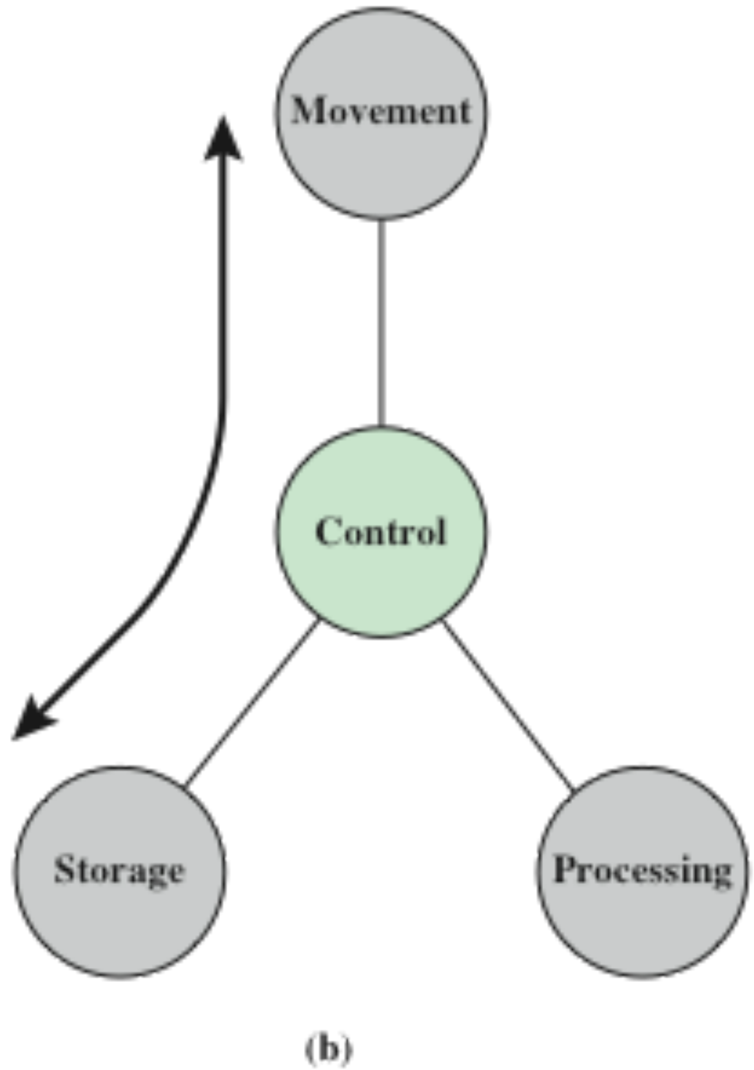



Figure 1.2 Possible Computer Operations

# Operations

## (C)
## Data Processing

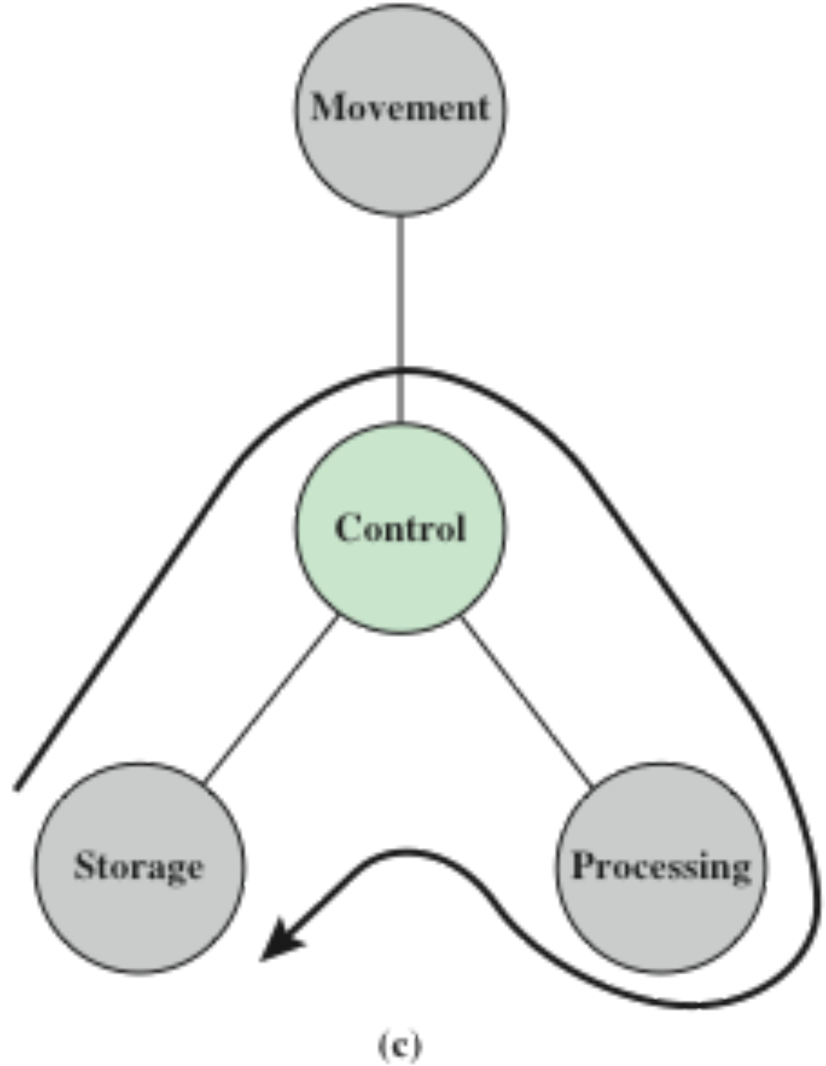



Figure 1.2 Possible Computer Operations

# Operations

**(D)**

**Data Processing**

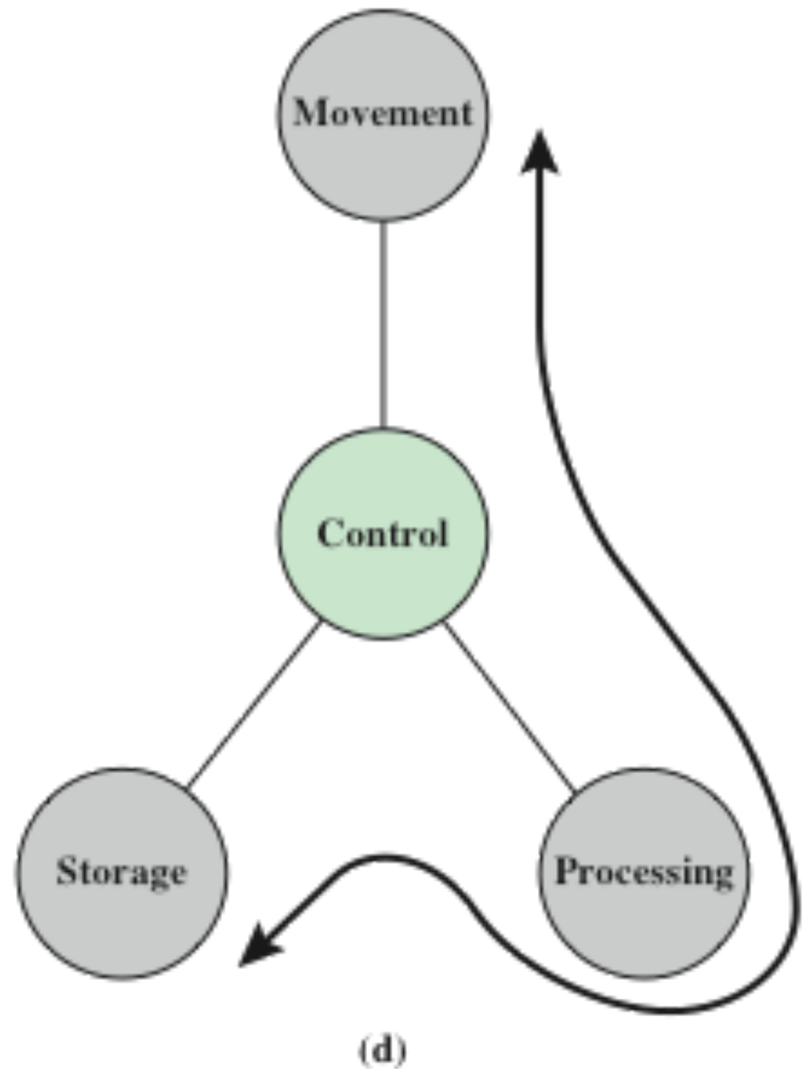



Figure 1.2 Possible Computer Operations

# The Computer

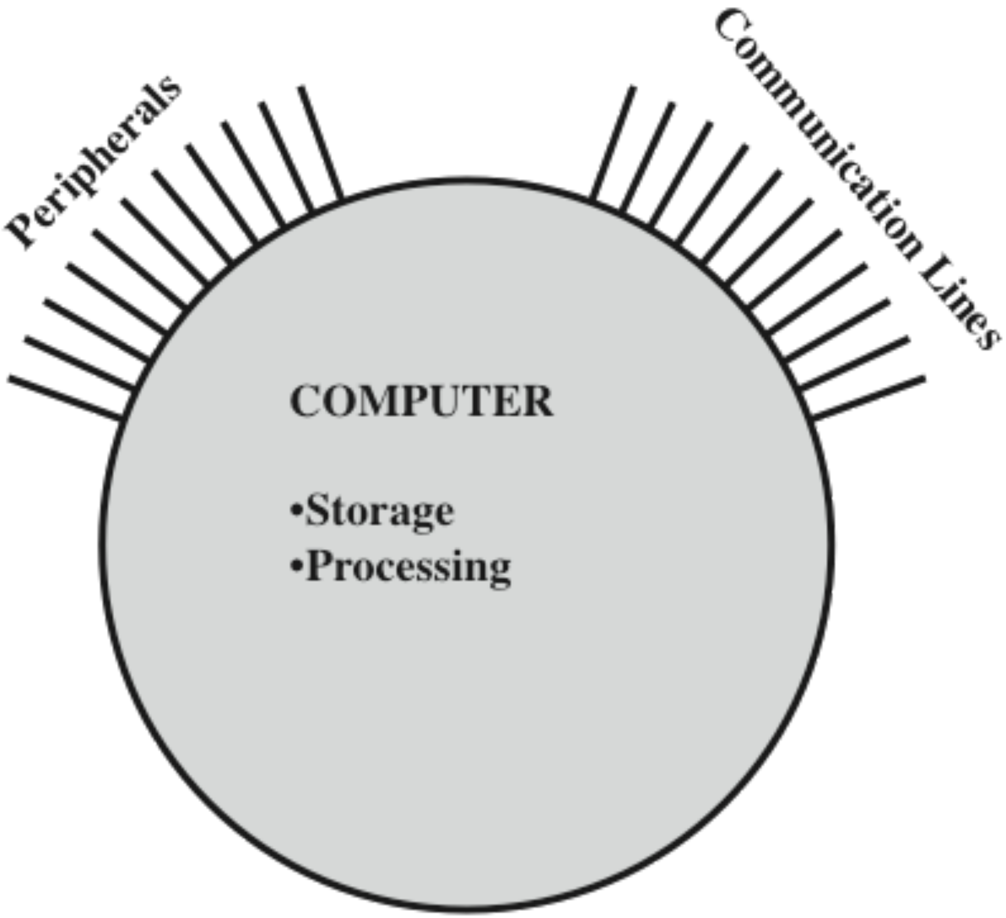



Figure 1.3 The Computer

# Structure

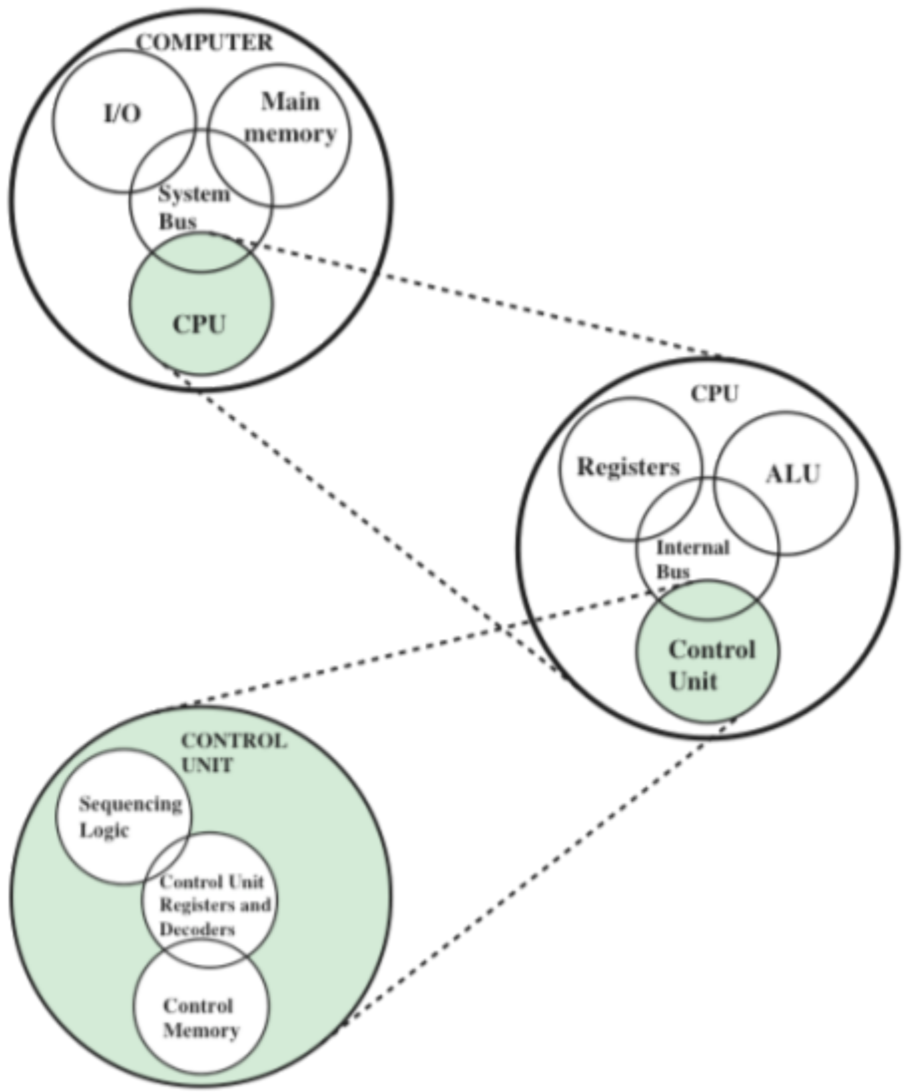
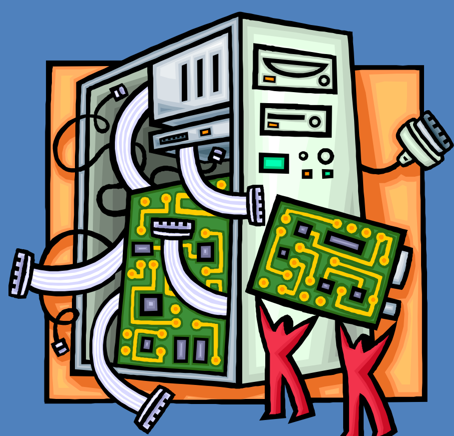



**Figure 1.4 A Top-Down View of a Computer**

+

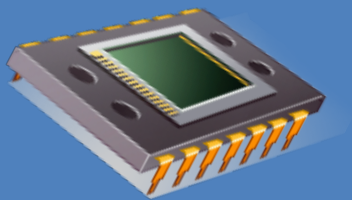
There are four main structural components of the computer:

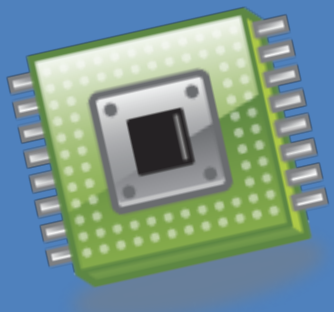


- CPU
  - Controls the operation of the computer and performs its data processing functions
- Main Memory
  - Stores data
- I/O
  - Moves data between the computer and its external environment
- System Interconnection
  - Some mechanism that provides for communication among CPU, main memory, and I/O

## CPU

Major structural components:





- Control Unit
  - Controls the operation of the CPU and hence the computer
- Arithmetic and Logic Unit (ALU)
  - Performs the computer's data processing function
- Registers
  - Provide storage internal to the CPU
- CPU Interconnection
  - Some mechanism that provides for communication among the control unit, ALU, and registers

# Questions

1. What, in general terms, is the distinction between computer organization and computer architecture?

2. What, in general terms, is the distinction between computer structure and computer function?

3. What are the four main functions of a computer?

4. List and briefly define the main structural components of a computer.

5. List and briefly define the main structural components of a processor.

# HW 1

- Problems 1 to 5
- HW template with problems will be available on Canvas

CALIFORNIA STATE UNIVERSITY FULLERTON

# CPSC-440 Computer System Architecture

Lecture 2

Performance Assessment

# Performance

- What we care most about…
  - How fast the computer can run a program
  - Response time or throughput
    - Response time: time to finish one single program
    - Throughput: total amount of work done in unit time

# CPU Performance Equation

- CPU Time

$$CPU\ Time = \frac{Clock\ cycles\ for\ a\ program\ (cycles)}{Clock\ Freq\ (cycles/sec)}$$

- If we know…
  - Total Instruction Counts ($I_c$)
  - Cycles Per Instruction (CPI)
  - Clock Frequency ($f$)
  - Cycle Time ($\tau$) - the inverse of the clock frequency ($1/f$)
  - CPU Time ($T$):

$$T = \frac{I_c \times CPI}{f} = I_c \times CPI \times \tau$$

# What if different instructions have different CPIs?

- CPU Time

$$T = \left( \sum_{i=1}^{n} (I_i \times CPI_i) \right) \times \tau$$

  - Where $i$ is the instruction type
- CPI

$$CPI = \frac{\sum_{i=1}^{n} (I_i \times CPI_i)}{I_c}$$

- IPC (Instructions Per Cycle)
  - Inverse of CPI

# MIPS and MFLOPS Rates

- MIPS (Millions of Instructions Per Second) Rate

$$MIPS\ Rate = \frac{I_C}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

- MFLOPS (Millions of Floating Point Operations Per Second) Rate

$$MFLOPS\ Rate = \frac{\#\ of\ executed\ floating\ point\ operations\ in\ a\ program}{Execution\ time \times 10^6}$$

# Example

- 2 million instructions on a 400 MHz processor
- 4 major types of instructions
- What's the MIPS rate?

$$CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24$$
$$MIPS\ Rate = (400 \times 10^6)/(2.24 \times 10^6) \approx 178$$

| **Instruction Type** | $CPI_i$ | $I_i$ **(%)** |
|---|---|---|
| Arithmetic and Logic | 1 | 60 |
| Load/Store with Cache Hit | 2 | 18 |
| Branch | 4 | 12 |
| Memory Reference with Cache Miss | 8 | 10 |

# Improve CPU time

- Instruction count
  - ISA and compiler technology
- CPI
  - Organization and ISA
- Clock cycle time
  - Hardware technology and organization

# Benchmarks

- MIPS and MFLOPS rates are inadequate to evaluate performance of processors
  - Because of differences in instruction sets, these rates are not valid means of comparing the performance of different architectures

# Example

$$A = B + C$$

Assume all quantities in main memory

| Complex Instruction Set Computer (CISC) | Reduced Instruction Set Computer (RISC) |
|---|---|
| • Can be compiled into one instruction<br>• Rated at 1 MIPS<br>add mem(B), mem(C), mem(A) | • Rated at 4 MIPS<br>load mem(B), reg(1)<br>load mem(C), reg(2)<br>add reg(1), reg(2), reg(3)<br>store reg(3), mem(A) |

# Standard Performance Evaluation Corporation (SPEC) Benchmark

- Benchmark Suite
  - Collection of programs
  - Provides a representative test of a computer in a particular application or area

# Performance Comparison

| Which One is Faster? |
|---|
| A is 10x faster than B for Prog P1 |
| B is 10x faster than A for Prog P2 |
| A is 20x faster than C for Prog P1 |
| C is 50x faster than A for Prog P2 |
| B is 2x faster than C for Prog P1 |
| C is 5x faster than B for Prog P2 |

# Total Execution Rates

- Both program A and B have equal number of instructions
- Below shows the execution rates

| | Computer 1 | Computer 2 | Computer 3 |
|---|---|---|---|
| Program A | 1 | 10 | 20 |
| Program B | 1000 | 100 | 20 |
| Total | 1001 | 110 | 40 |

# Average Execution Rate

- What if Program A and B have a different number of instructions?
- If there are $m$ different benchmark programs

$$R_A = \frac{1}{m}\sum_{i=1}^{m} R_i$$

  - Where $R_i$ is the high-level language instruction execution rate for the $i^{th}$ benchmark program
- The throughput of a machine carrying out a number of tasks
  - The higher the rate ($R_A$) the better

# Harmonic Mean

- Alternative to average execution rate

$$R_H = \frac{m}{\sum_{i=1}^{m} \frac{1}{R_i}}$$

- The reciprocal of the arithmetic mean of the reciprocals
- Gives the inverse of the average execution rate
- Again, the higher the rate ($R_H$) the better

# Total Execution Time Example

The top table shows the execution rates. Assume each program has equal weight.

| | Computer A | Computer B | Computer C |
|---|---|---|---|
| Program 1 | 100 | 10 | 5 |
| Program 2 | 0.1 | 1 | 5 |
| Program 3 | 0.2 | 0.1 | 2 |
| Program 4 | 1 | 0.125 | 1 |

| | $R_A$ | Rank | $R_H$ | Rank |
|---|---|---|---|---|
| Computer A | 25.325 | 1 | 0.25 | 2 |
| Computer B | 2.8 | 3 | 0.21 | 3 |
| Computer C | 3.25 | 2 | 2.1 | 1 |

# SPEC Benchmark
## Speed Metrics

- Measures the ability of a computer to complete a single task

$$r_i = \frac{Tref_i}{Tsut_i}$$

  - $Tref_i$ - execution time of benchmark program $i$ on the reference system
  - $Tsut_i$ - execution time of benchmark program $i$ on the system under test
- The larger the ratio, the higher the speed

# SPEC Benchmark
## Speed Metrics

- Example
  - A system executes a program in 934 sec.
  - The reference implementation requires 22,135 sec.

$$\frac{22{,}135\ sec}{934\ sec} = 23.7$$

# SPEC Benchmark
## Rate Metric

- Throughput/rate of a machine carrying out a number of tasks
- Multiple copies of benchmarks run simultaneously

$$r_i = \frac{N \times Tref_i}{Tsut_i}$$

  - N – number of copies of the program that are run simultaneously

# SPEC Benchmark
## Geometric Mean

- Averages ratios for all 12 integer benchmarks
- Used to determine the overall performance measure

$$r_G = \left(\prod_{i=1}^{n} r_i\right)^{1/n}$$

| Benchmark | Ratio |
|---|---|
| 400.perlbench | 17.5 |
| 401.bzip2 | 14.0 |
| 403.gcc | 13.7 |
| 429.mcf | 17.6 |
| 445.gobmk | 14.7 |
| 456.hmmer | 18.6 |

| Benchmark | Ratio |
|---|---|
| 458.sjeng | 17.0 |
| 462.libquantum | 31.3 |
| 464.h264ref | 23.7 |
| 471.omnetpp | 9.23 |
| 473.astar | 10.9 |
| 483.xalancbmk | 14.7 |

$$(17.5 \times 14 \times 13.7 \times 17.6 \times 14.7 \times 18.6 \times 17 \times 31.3 \times 23.7 \times 9.23 \times 10.9 \times 14.7)^{1/12} = 18.5$$

# Amdahl's Law

- *Speedup in one aspect of technology/design does not result in a corresponding improvement in performance*

$$Speedup = \frac{Execution\ time\ before\ enhancement}{Execution\ time\ after\ enhancement}$$

# Amdahl's Law Example

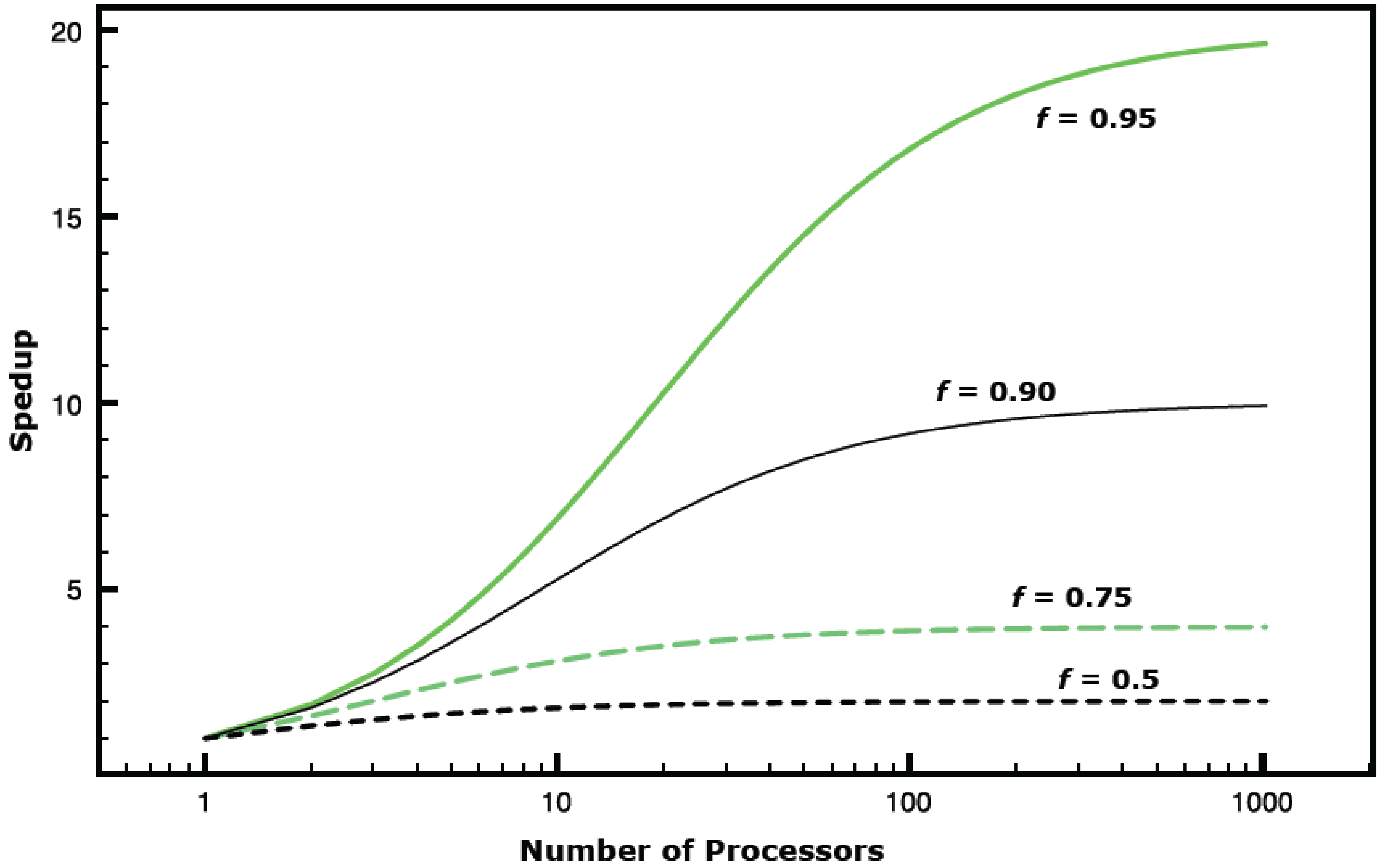
- Single vs. Multiple processors

$$Speedup = \frac{X}{Y} = \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + \frac{f}{N}}$$

  - $X$: Time to execute a program on a single processor
  - $Y$: Time to execute a program on N parallel processors
  - $T$: Total execution time
  - $f$: Fraction of code executed on parallel processors (no scheduling overhead)
  - $(1-f)$: Fraction of code executed on a single processor

1. When $f$ is small, the use of parallel processors has little effect
2. As $N \rightarrow \infty$, speedup bound by $1/(1-f)$
   - Diminishing returns for using more processors

# Amdahl's Law Example



CALIFORNIA STATE UNIVERSITY FULLERTON

# CPSC-440 Computer System Architecture

Lecture 3

Von Neumann Machines (IAS)

# History of Computers

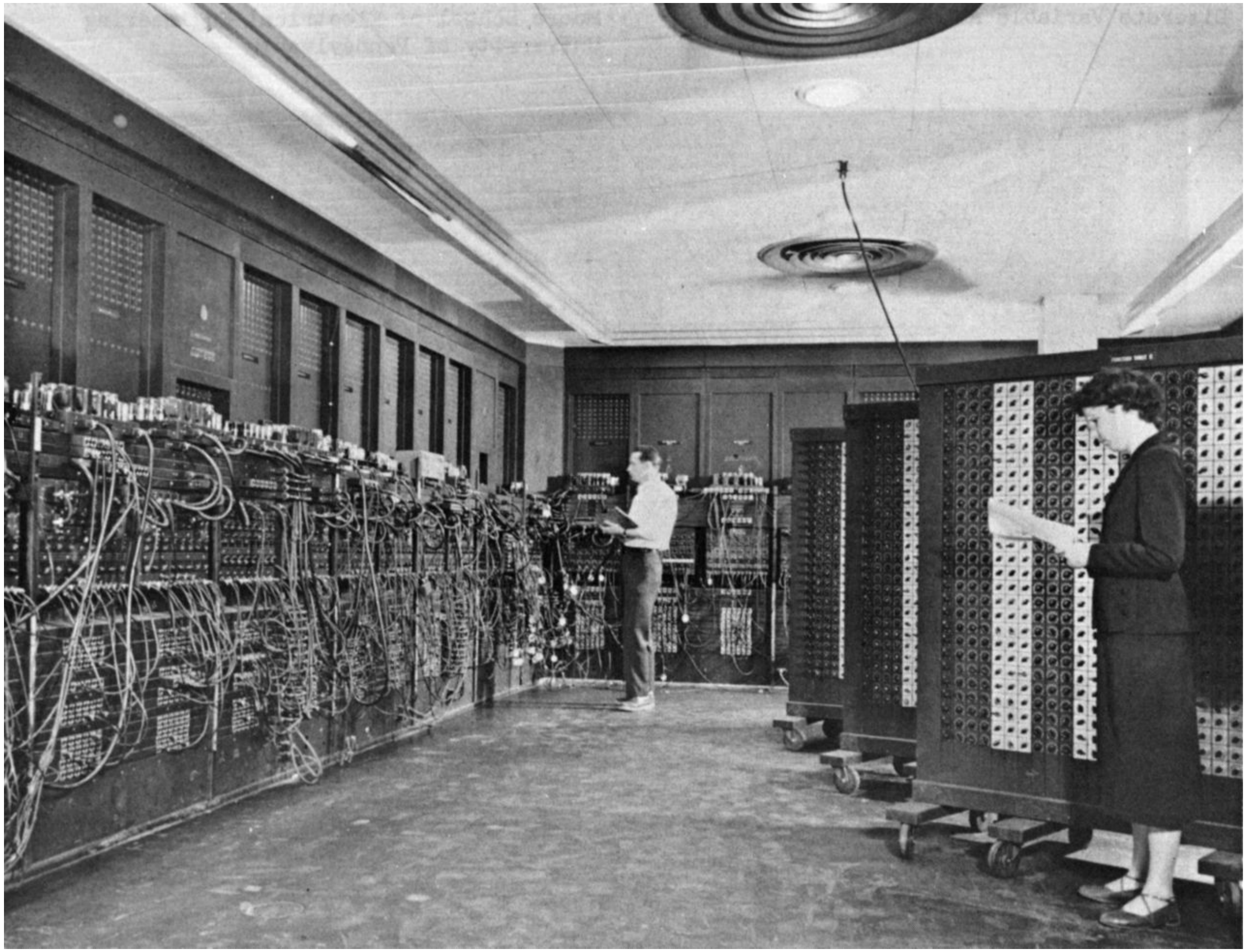
## First Generation: Vacuum Tubes

- ENIAC
  - Electronic Numerical Integrator And Computer
- Designed and constructed at the University of Pennsylvania
  - Started in 1943 – completed in 1946
  - By John Mauchly and John Eckert
- World's first general purpose electronic digital computer
  - Army's Ballistics Research Laboratory (BRL) needed a way to supply trajectory tables for new weapons accurately and within a reasonable time frame
  - Was not finished in time to be used in the war effort
- Its first task was to perform a series of calculations that were used to help determine the feasibility of the hydrogen bomb
- Continued to operate under BRL management until 1955 when it was disassembled

# ENIAC

- Weighed 30 tons
- Occupied 1500 square feet of floor space
- Contained more than 18,000 vacuum tubes
- 140 kW Power consumption
- Capable of 5000 additions per second
- Decimal rather than binary machine
- Memory consisted of 20 accumulators, each capable of holding a 10 digit number
- Major drawback was the need for manual programming by setting switches and plugging/ unplugging cables

# ENIAC



# John von Neumann

## EDVAC (Electronic Discrete Variable Computer)

- First publication of the idea was in 1945
- Stored program concept
  - Attributed to ENIAC designers, most notably the mathematician John von Neumann
  - Program represented in a form suitable for storing in memory alongside the data
- IAS computer
  - Princeton Institute for Advanced Studies
  - Prototype of all subsequent general-purpose computers
  - Completed in 1952

# Structure of von Neumann Machine

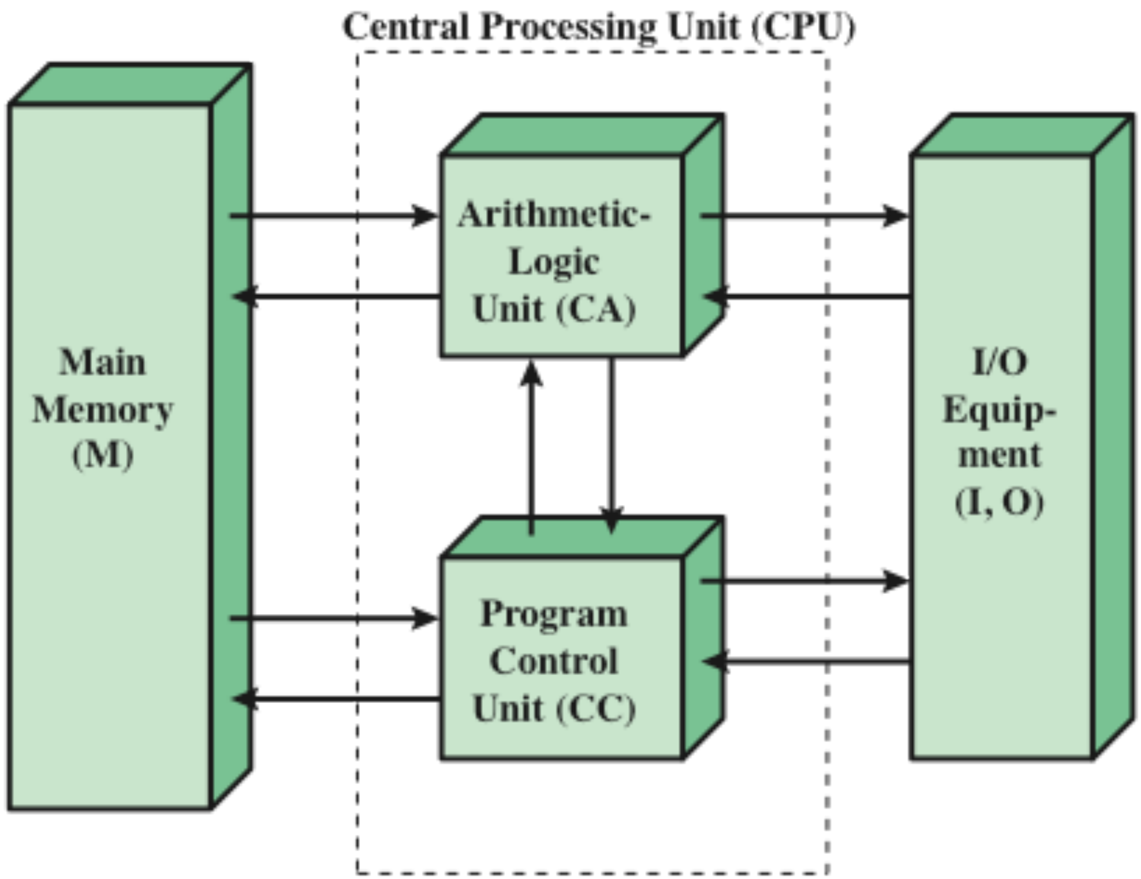



**Figure 2.1 Structure of the IAS Computer**

# IAS Memory Formats

- The memory of the IAS consists of 1000 storage locations (called words) of 40 bits each

- Both data and instructions are stored there
- Numbers are represented in binary form and each instruction is a binary code

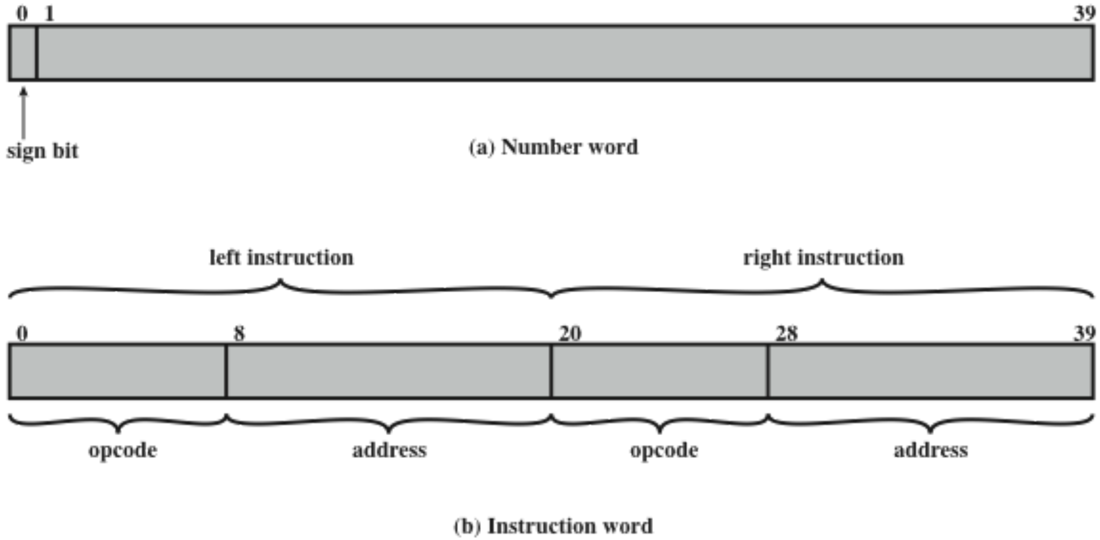
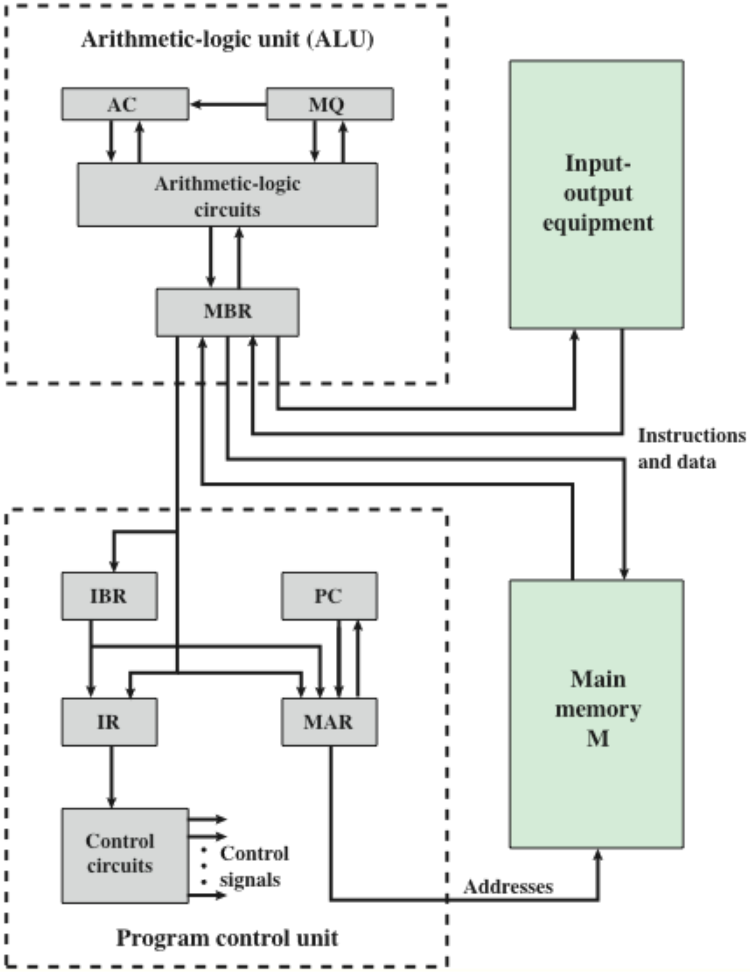



Figure 2.2 IAS Memory Formats

# Structure of IAS Computer

## Registers

- Memory Buffer Register (MBR)
  - Word to be stored/received in/from memory or I/O unit
- Memory Address Register (MAR)
  - Memory Address of the word to be (written from)/(read into) the MBR
- Instruction Register (IR)
  - Contains 8-bit opcode
- Instruction Buffer Register (IBR)
  - Temporarily holds the right-hand instruction
- Program Counter (PC)
  - Contains address of the next instruction pair to be fetched from memory
- Accumulator (AC) and Multiplier Quotient (MQ)
  - Employed to temporarily hold operands and results of ALU operations



| Instruction Type | Opcode | Symbolic Representation | Description |
|---|---|---|---|
| Data transfer | 00001010 | LOAD MQ | Transfer contents of register MQ to the accumulator AC |
| | 00001001 | LOAD MQ,M(X) | Transfer contents of memory location X to MQ |
| | 00100001 | STOR M(X) | Transfer contents of accumulator to memory location X |
| | 00000001 | LOAD M(X) | Transfer M(X) to the accumulator |
| | 00000010 | LOAD –M(X) | Transfer –M(X) to the accumulator |
| | 00000011 | LOAD \|M(X)\| | Transfer absolute value of M(X) to the accumulator |
| | 00000100 | LOAD –\|M(X)\| | Transfer –\|M(X)\| to the accumulator |
| Unconditional branch | 00001101 | JUMP M(X,0:19) | Take next instruction from left half of M(X) |
| | 00001110 | JUMP M(X,20:39) | Take next instruction from right half of M(X) |
| Conditional branch | 00001111 | JUMP+ M(X,0:19) | If number in the accumulator is nonnegative, take next instruction from left half of M(X) |
| | 00010000 | JUMP+ M(X,20:39) | If number in the accumulator is nonnegative, take next instruction from right half of M(X) |
| Arithmetic | 00000101 | ADD M(X) | Add M(X) to AC; put the result in AC |
| | 00000111 | ADD \|M(X)\| | Add \|M(X)\| to AC; put the result in AC |
| | 00000110 | SUB M(X) | Subtract M(X) from AC; put the result in AC |
| | 00001000 | SUB \|M(X)\| | Subtract \|M(X)\| from AC; put the remainder in AC |
| | 00001011 | MUL M(X) | Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ |
| | 00001100 | DIV M(X) | Divide AC by M(X); put the quotient in MQ and the remainder in AC |
| | 00010100 | LSH | Multiply accumulator by 2; i.e., shift left one bit position |
| | 00010101 | RSH | Divide accumulator by 2; i.e., shift right one position |
| Address modify | 00010010 | STOR M(X,8:19) | Replace left address field at M(X) by 12 rightmost bits of AC |
| | 00010011 | STOR M(X,28:39) | Replace right address field at M(X) by 12 rightmost bits of AC |

Table 2.1: The IAS Instruction Set

# Table 2.1

# The IAS Instruction Set

# Example 1

- What would the machine code instruction look like to add the contents of memory address 5CD (HEX) with the accumulator and stores the result back into the accumulator?

| Left Instruction | | Right Instruction | |
|---|---|---|---|
| 0 | 8 | 20 | 28 … 39 |
| 0000 0101 | 0101 1100 1101 | XXXX XXXX | XXXX XXXX XXXX |
| Opcode | Address | Opcode | Address |

| Symbolic | Description |
|---|---|
| ADD M(X) | Add M(X) to AC; put the result in AC |

# Example 1

Left instruction first

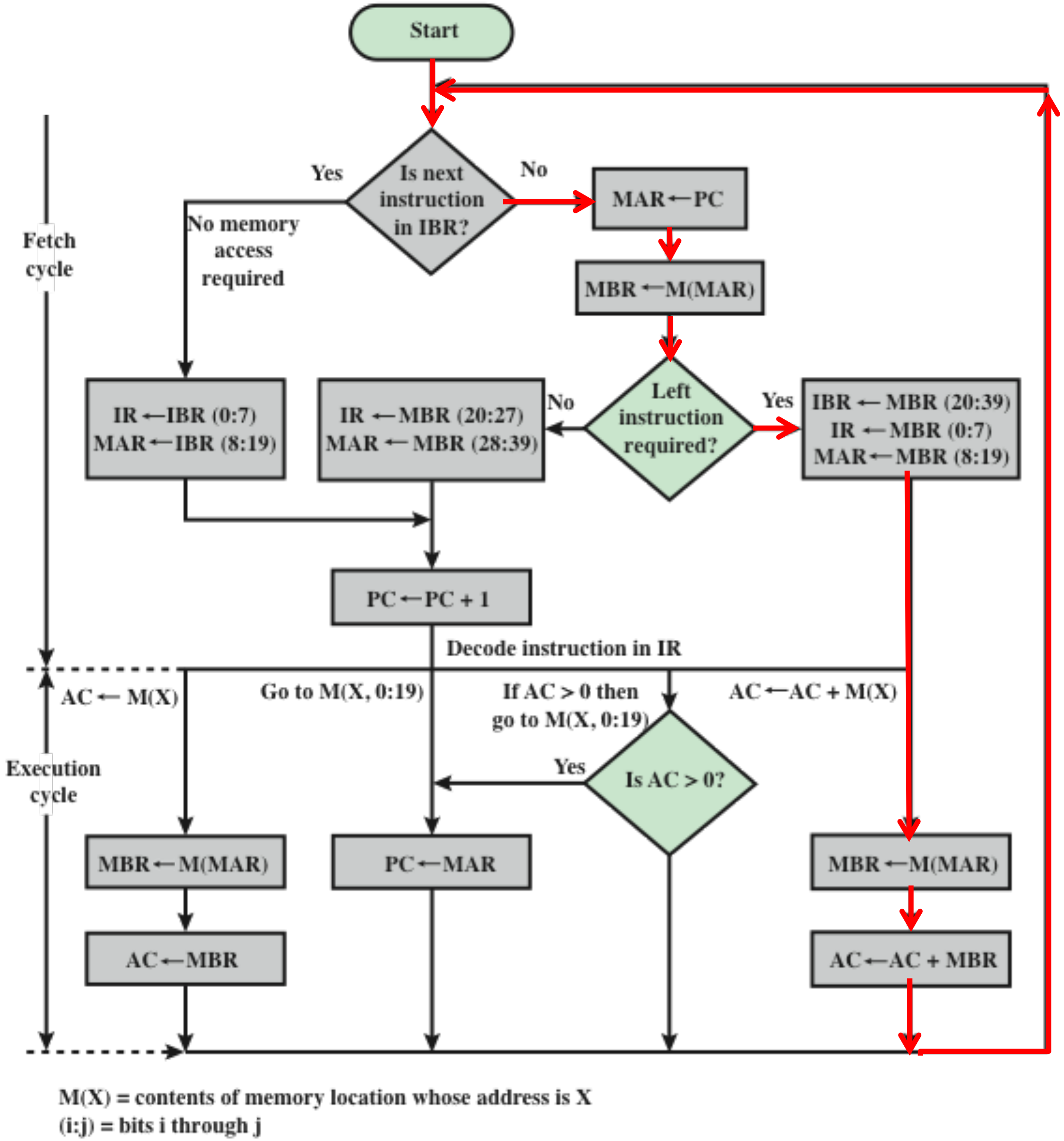
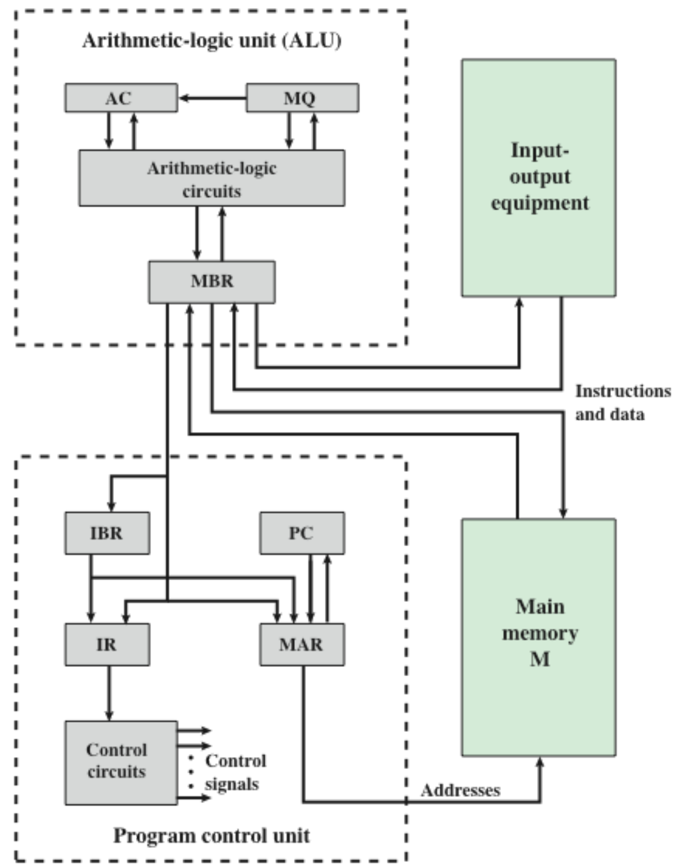
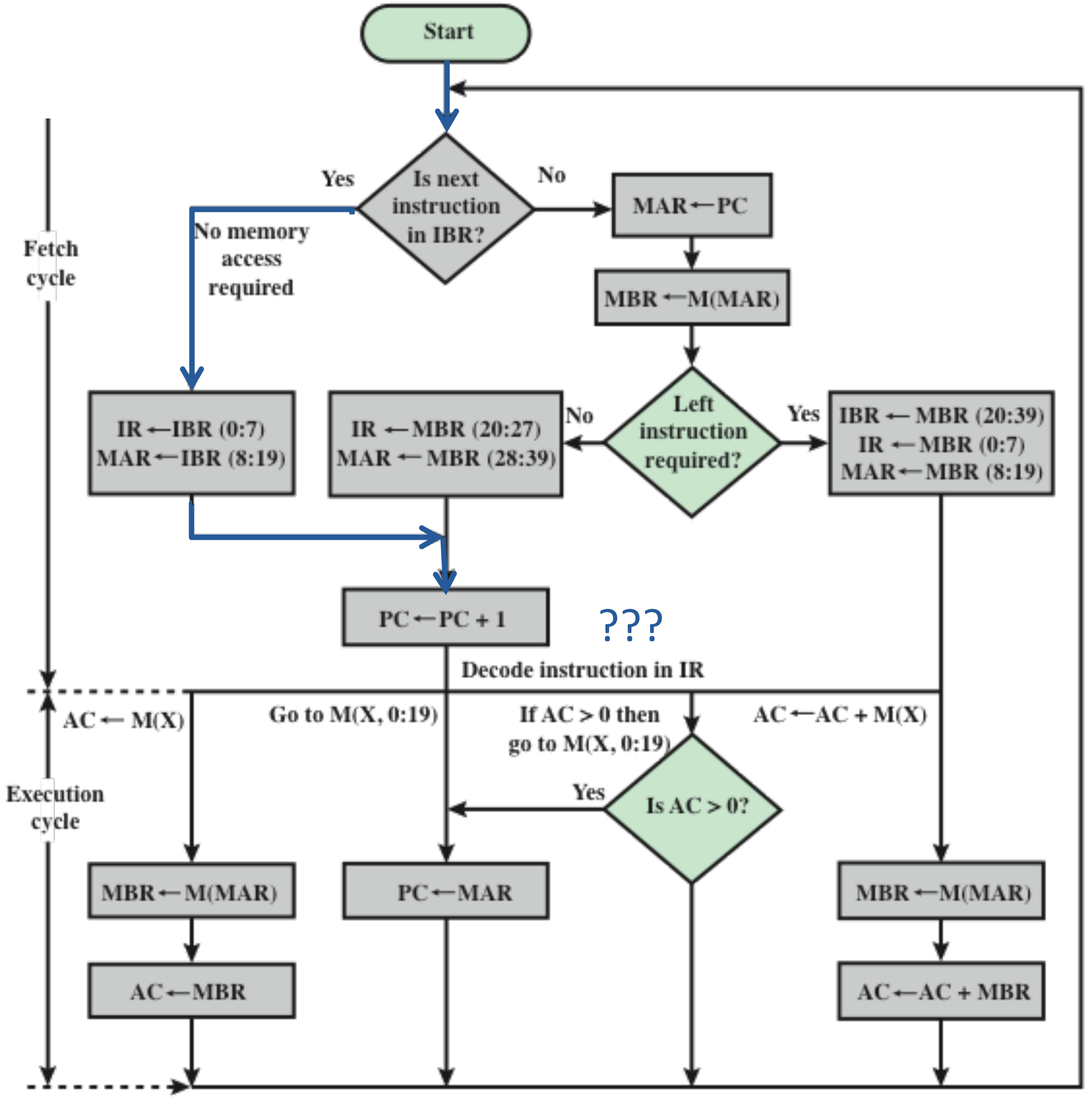



Figure 2.4 Partial Flowchart of IAS Operation



# Example 1

The execution path would depend on the right instruction




M(X) = contents of memory location whose address is X
(i:j) = bits i through j

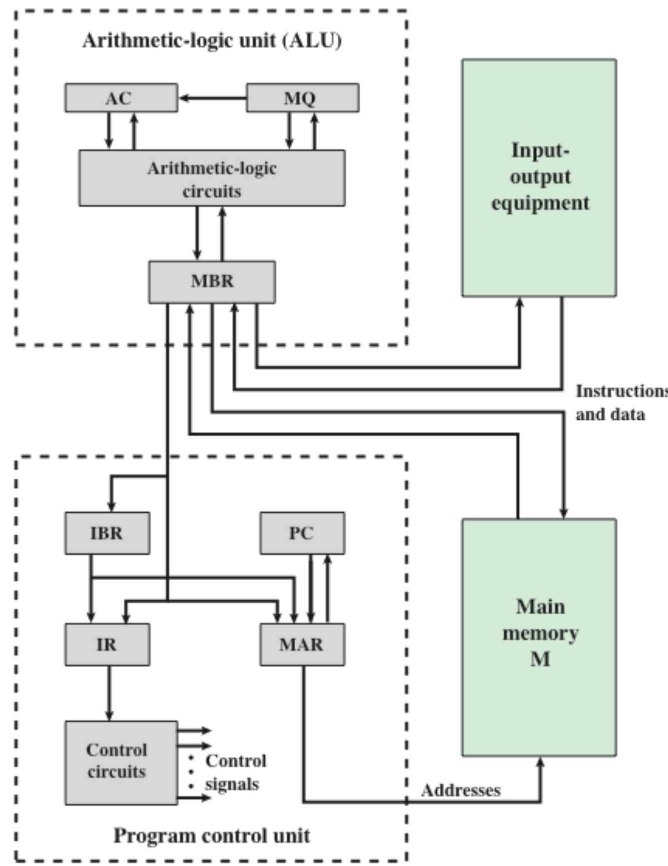



**Figure 2.4 Partial Flowchart of IAS Operation**

# Example 2

- What is the assembly language code for the program:

| Left Instruction | | Right Instruction | |
|---|---|---|---|
| 0 | 8 | 20 | 28 – 39 |
| 0010 0001 | 1100 0110 1111 | 0001 0100 | XXXX XXXX XXXX |
| Opcode | Address | Opcode | Address |

| Address | Machine Code |
|---|---|
| 06B | 21C6F14XXX |

NOTE: IAS doesn't actually have an assembly language

# Example 2

- What is the assembly language code for the program:

| Left Instruction | | Right Instruction | |
|---|---|---|---|
| 0 | 8 | 20 | 28 … 39 |
| 0010 0001 | 1100 0110 1111 | 0001 0100 | XXXX XXXX XXXX |
| Opcode | Address | Opcode | Address |

| Address | Machine Code |
|---|---|
| 06B | 21C6F14XXX |

| Address | Symbolic |
|---|---|
| 06B | STOR M(C6F) |
| | LSH |

# Example 3

- Write an IAS program to compute the results of the following equation:

$$Y = \frac{N(N+1)}{2}$$

- Assume that the result of the computation doesn't overflow and N is a positive integer

# Example 3

$$Y = \frac{N(N+1)}{2}$$

| Location | Instruction/Value | Comments |
|---|---|---|
| 0 | <> | Constant (N) [initialized to some value] |
| 1 | 1 | Constant; Integer value = 1 |
| 2 | 2 | Constant; Integer value = 2 |
| 3 | 0 | Variable Y (initialized to integer zero) |
| 4 | 0 | Variable X (initialized to integer zero) |
| 5L | LOAD M(0) | N → AC |
| 5R | ADD M(1) | AC + 1 → AC; (N+1) |
| 6L | STOR M(4) | AC → X; X=N+1 |
| 6R | LOAD MQ,M(4) | X→ MQ; MQ=N+1 |
| 7L | MUL M(0) | MQ*M(0) → N(N+1) → AC |
| 7R | DIV M(2) | AC/2 → AC; AC=N(N+1)/2 |
| 8L | STOR M(3) | AC → Y; saving the Sum in variable Y |
| 8R | JUMP M(8,20:39) | Jump to 8R; Done |

# Example 4

- Write an IAS program to compute the results of the following equation:

$$Y = \sum_{X=1}^{N} X$$

- Assume that the result of the computation doesn’t overflow, and that X, Y, and N are positive integers

# Example 4

$$Y = \sum_{X=1}^{N} X$$

| Location | Instruction/Value | Comments |
|---|---|---|
| 0 | <> | Constant (N) [initialized to some value] |
| 1 | 1 | Constant (loop counter increment) |
| 2 | 1 | Variable i (loop index value; current) |
| 3 | 1 | Variable Y = Sum of X values (Initialized to One) |
| 4 | LOAD M(0) | N → AC (the max limit) |
| 5L | SUB M(2) | Compute N–i → AC |
| 5R | JUMP + M(6,20:39) | If AC > 0 [i < N] then jump to 6R |
| 6L | JUMP M(6,0:19) | Loop here (HALT) |
| 6R | LOAD M(2) | i<N so continue; Get loop counter i |
| 7L | ADD M(1) | i+1 in AC |
| 7R | STOR M(2) | AC → i |
| 8L | ADD M(3) | i + Y in AC |
| 8R | STOR M(3) | AC → Y |
| 9L | JUMP M(5,0:19) | Jump to 5L |

# Homework Problems

- Problems are available on Canvas

# Study Guide Exam #1 — CS 440 Computer Architecture

Jared Dyreson
California State University, Fullerton

February 24, 2021

## Contents

# 1 Lecture 00

## 1.1 Matrices

### Example Usage

```
a = [12; 21]
% 1 2
% 2 1


a * a

% 5 4
% 4 5
```

1. Matrix multiplication is not commutative
2. Inverse function is the same as division
3. Cannot invert all matrices (only with determinant not equal to 0)
4. System of equations can be solved
5. Ranges follow this pattern "begin:step:end"
6. Steps can be any decimal value

## 2 Lecture 01

1. Difference between architecture and organization
    - **Architecture:** Specifications of the system being built, which are a set of rules/methods. These describe the functionality, organization and implementation of computer systems.
    - **Organization:** Deals with the hardware components of a computer system, which include I/O devices, the CPU, storage and primary memory devices (RAM).
2. Four structural components for computer:
    - CPU
    - Volatile Memory (RAM)
    - I/O
    - System Interconnections
3. Four structural components for computer:
    - Control Unit (CU)
    - Arithmetic Logic Unit (ALU)
    - Registers
    - CPU Interconnections

# 3 Lecture 02

1. Performance Assessment
   - **Qualitative:** relating to the possession of qualities without reference to the quantities involved
   - **Quantitative:** relating to a measurable and numeric representation of a given entity (this is how we gauge the performance of a chipset)
2. **CPU Time:** $\frac{\text{Clock cycles for a program (cycles)}}{\text{Clock Frequency (cycles/sec)}}$. The amount of time it takes for a CPU to complete a given set of instructions.
3. **CPI:** Cycles Per Instruction
4. **IPC:** Instructions Per Cycle (inverse of CPI)
5. **MIPS:** Million(s) of Instructions per Second
6. **MFLOPS:** Million(s) of Floating Point Operations Per Second
7. For benchmarks however, this will not suffice as it is hard to see which machine is faster

## 3.1 Benchmark Types

1. **Total Execution Rate:** If Program A and B have equal amount of instructions, you can sum them up individually
2. **Average Execution Rate:** When Program A and B have an unequal amount of instructions
3. **Harmonic Mean:** The reciprocal of the arithmetic mean of the reciprocals. Alternative to average execution rate.
4. **SPEC Benchmark:** Measures the ability of a computer to complete a single task.

$$r_i = \frac{\text{Tref}_i}{\text{Tsut}_i}$$

   - $\text{Tref}_i$ - execution time of benchmark program $i$ on the reference system
   - $\text{Tsut}_i$ - execution time of benchmark program $i$ on the system under test
   - The larger the ratio, the higher the speed

### 3.2 Amdahl's Law

Adding more processors does not make the program execution time improve.

$$\text{Speedup} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}}$$

Good Video Explanation

The more of the code that is running on multiple processors, it will speed up, but it will become saturated.

# 4 Lecture 03

1. ENIAC does not use binary number
2. Binary only has 1's and 0's
3. Can utilize left/right bit shifts (dividing/multiplying by 2), boolean executions
4. You can't use this with decimal numbers (base 10)
5. It never had any memory, so programs could not be stored
6. Von Neumann machines has same four components of computer
7. IAS Memory formats, which hold 40 bits and both instructions/data are stored
    - **Left Hand Side**
    - Opcode: 0 - 7
    - Address: 8 - 19
    - **Right Hand Side**
    - Opcode: 20 - 27
    - Address: 28 - 39

## Registers

- Memory Buffer Register (MBR)
  - Word to be stored/received in/from memory or I/O unit
- Memory Address Register (MAR)
  - Memory Address of the word to be (written from)/(read into) the MBR
- Instruction Register (IR)
  - Contains 8-bit opcode
- Instruction Buffer Register (IBR)
  - Temporarily holds the right-hand instruction
- Program Counter (PC)
  - Contains address of the next instruction pair to be fetched from memory
- Accumulator (AC) and Multiplier Quotient (MQ)
  - Employed to temporarily hold operands and results of ALU operations