

Introduction to Computing

Michael C. Hackett
Assistant Professor, Computer Science

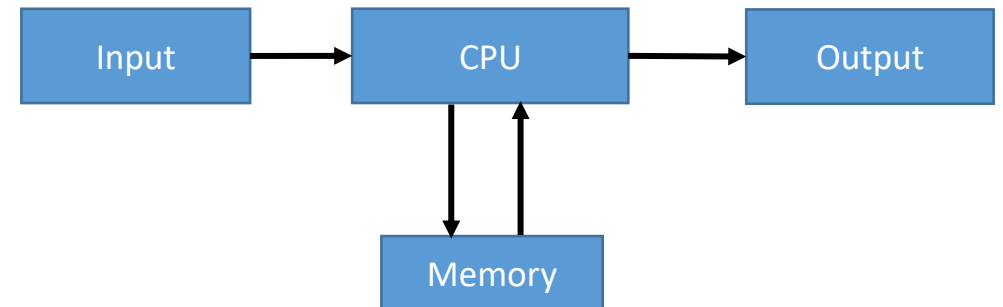


What is Computer Science?

- ***Computer Science*** is the scientific study of computation, theoretically and in practice, as the basis for determining what problems can be computed.
- Computer Science is **not** just the study or use of computers.
 - It's a branch of mathematics.
 - Computers are the tools we use.
- Some things change every day (new tech) and some things never change (math and logic).
 - This course is about the things that never change.

What is a computer?

- A **computer** is a device or machine that is capable of performing arithmetic and/or logical operations.
 - A modern definition would include the capability of storing and processing information.
- A modern computer system is comprised of:
 - Central Processing Unit
 - Memory
 - Input
 - Output

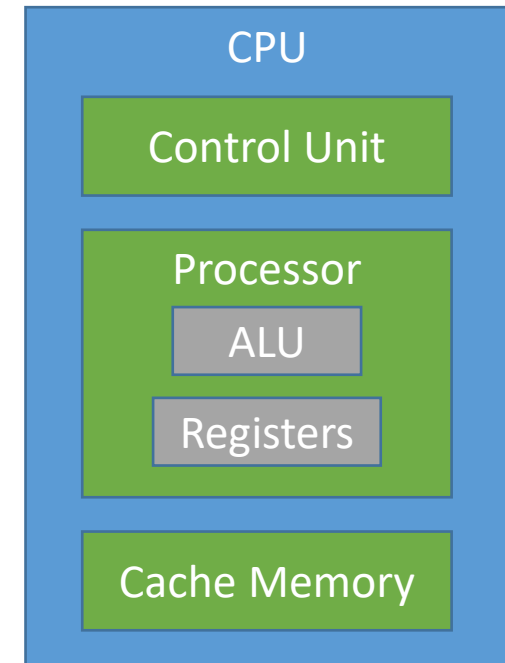


Hardware and Software

- **Hardware** is any component of a computer that you can physically touch.
 - Processors, disk drives, RAM, monitors, keyboards, and mice.
- **Software** is any intangible component of a computer.
 - Operating systems, applications, pictures, videos, and files.

Central Processing Unit (CPU)

- The Central Processing Unit is a piece of computer hardware that performs the instructions of computer programs.
 - Performs logical and arithmetic operations.
- Control Unit (CU)
 - Governs how instructions are carried out by the processor.
- Arithmetic Logic Unit (ALU)
 - Performs arithmetic and logical operations.
- Registers
 - Small amounts of memory available to the processor.
 - Single numbers and instructions.
- Cache Memory
 - Larger amount of memory available to the CPU.
 - Holds values recently sent to or retrieved from main memory for faster recall.



Memory Hierarchy

1. Registers

- *Stores data being used right now*

2. Cache

- *Stores data used recently*

3. Main Memory

- *Stores data that might be needed later*

4. Secondary Storage

- *Stores data even after the computer is powered off*
- E.g. Hard Drives

Computer Generations

- Zeroth Generation – Mechanical Computers (1640s – 1940s)
- First Generation – Vacuum Tubes (1940s – 1950s)
- Second Generation – Transistors (1950s – 1960s)
- Third Generation – Integrated Circuits (1960s – 1980s)
- Fourth Generation – Very Large Scale Integration (1980s – Present)

Mechanical Computers

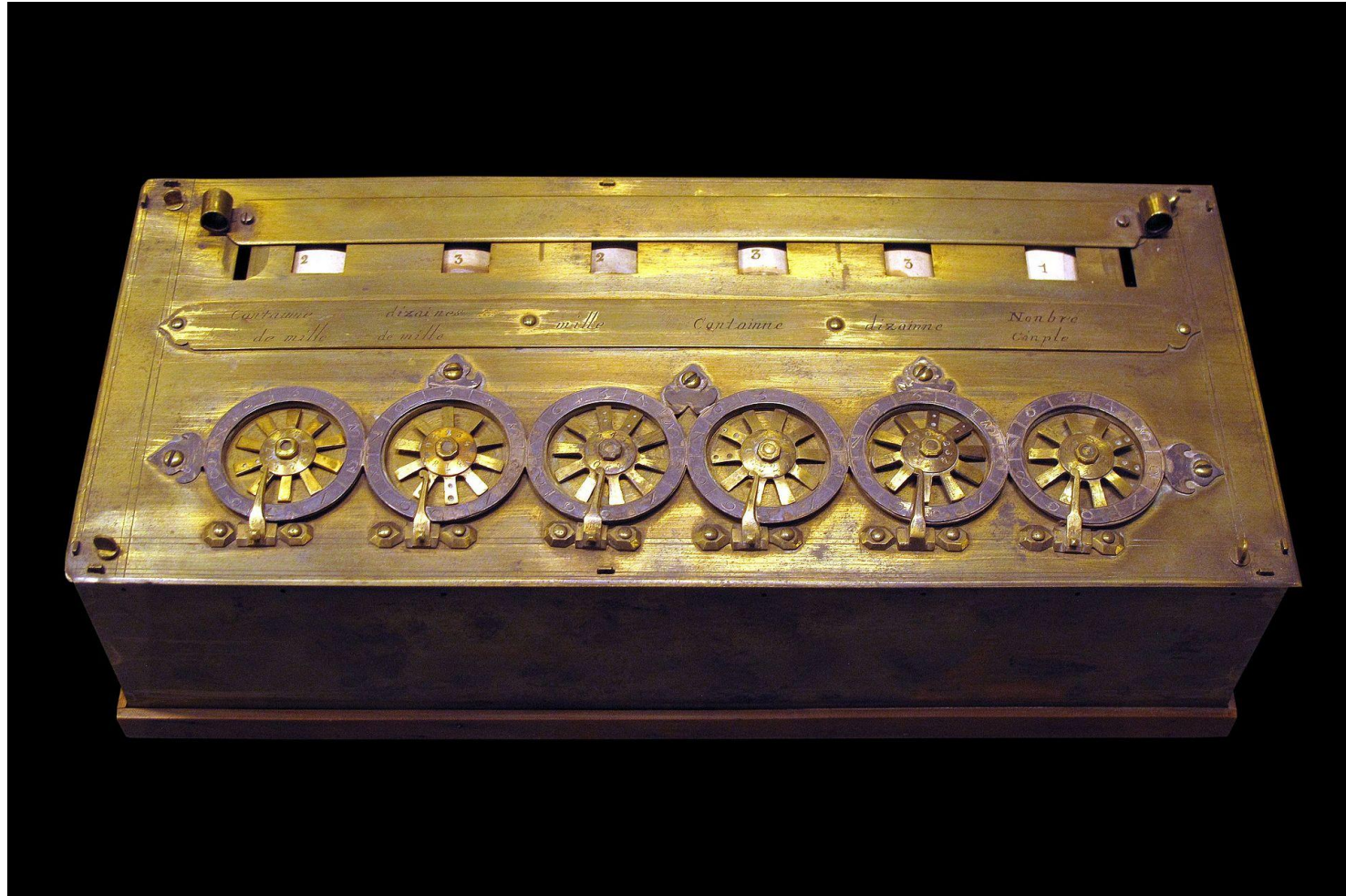
- No electricity.*
 - Operated by hand.
 - *Some used electromechanical parts near the end of the generation.
- Typically made of wood and metal.
- More sophisticated than other tools like an abacus or slide rule.

The Pascaline

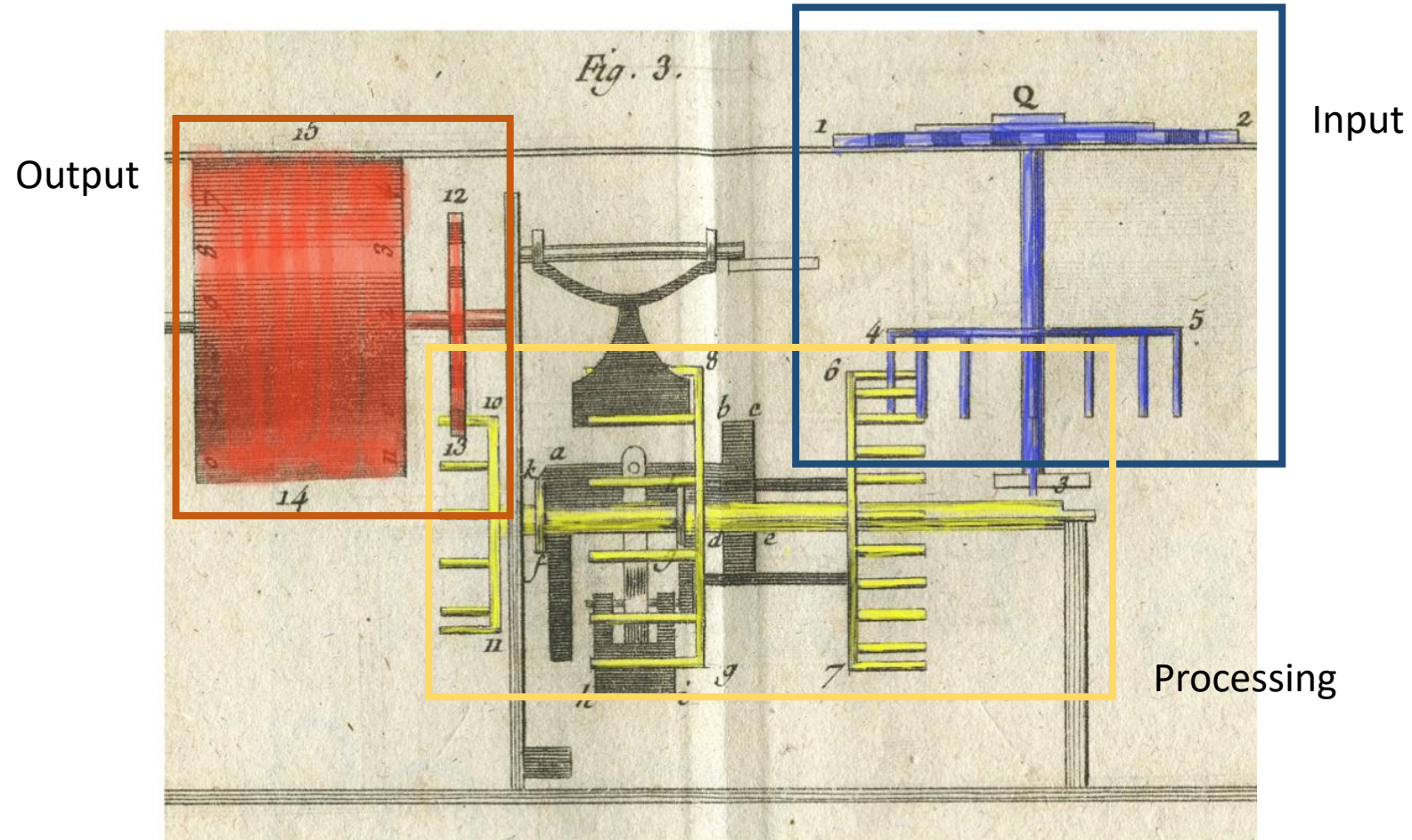
- Also known as Pascal's Calculator
- Invented by Blaise Pascal
 - French mathematician
- Started work in 1642 (20 years old)
 - Finished in 1645
- Only made about 2 dozen.



The Pascaline



The Pascaline



The Pascaline

- Could add and subtract.
- Could multiply and divide (using repeated addition or subtraction).
- Had a few variations:
 - Scientific
 - Accounting
 - Land Surveying

[Pascaline Simulation Video](#)

Leibniz Wheel/Stepped Drum

- Invented by Gottfried Leibniz
 - German mathematician
- Created in 1673
- Heavily influenced by the Pascaline.
 - Wanted to design a device that could add, subtract, multiply, and divide.



Stepped Reckoner

- Invented by Gottfried Leibniz
- Completed in 1694
- Could add, subtract, multiply, and divide.
- Required very precise components
 - Reliability problems

Stepped Reckoner



Leibniz Wheel/Stepped Drum



[Stepped Drum Simulation Video](#)



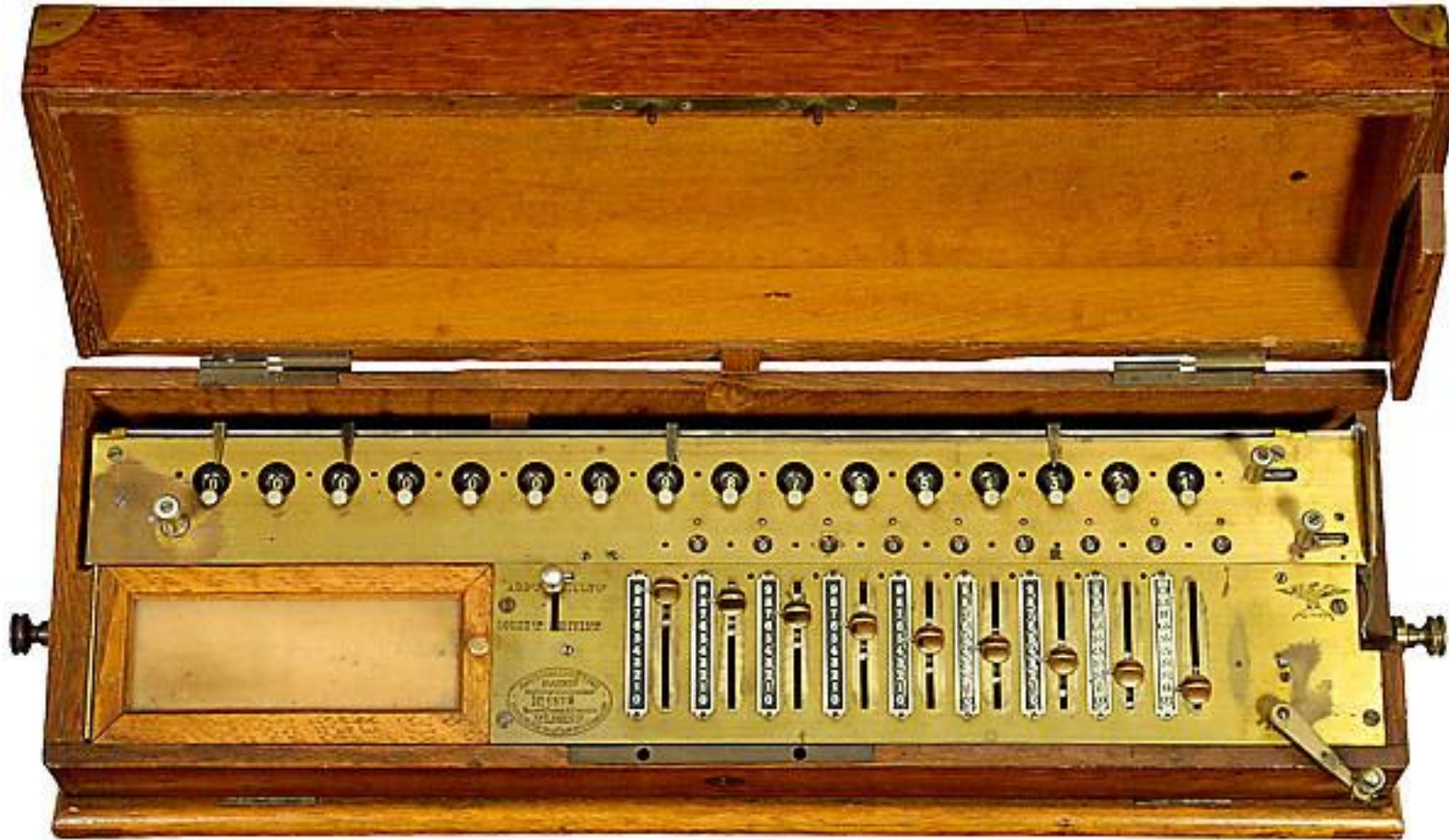
Stepped Reckoner

- Never brought to market.
 - Only two prototypes
 - Showed the potential/possibility of this concept
- The Leibniz Wheel was used in many other mechanical calculators through the 1970s.

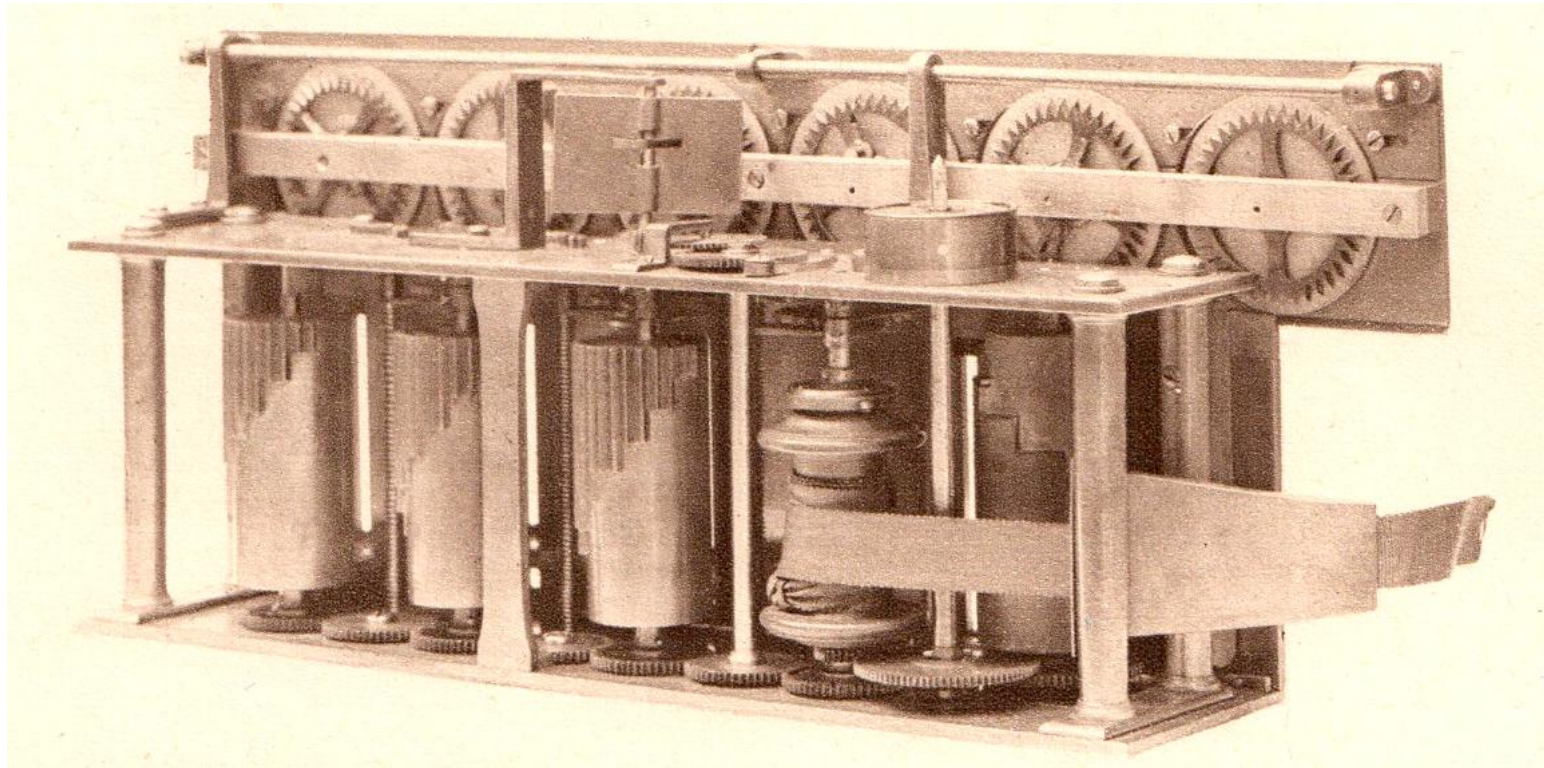
Arithmometer

- Created by Charles Thomas
 - French Inventor
- First commercially successful mechanical calculator
 - 1850s to 1890s – only commercially available mechanical calculator.
- Utilized concepts from the Pascaline and Stepped Reckoner.

Arithmometer



Arithmometer



[Arithmometer Simulation Video](#)

Curta

- Created by Curt Herzstark
 - Austrian Engineer
- First successful portable calculator (1950s/1960s)
 - Up until electronic handheld calculators came out in the 1970s.
- Utilized concepts from the Stepped Reckoner and Arithmometer.

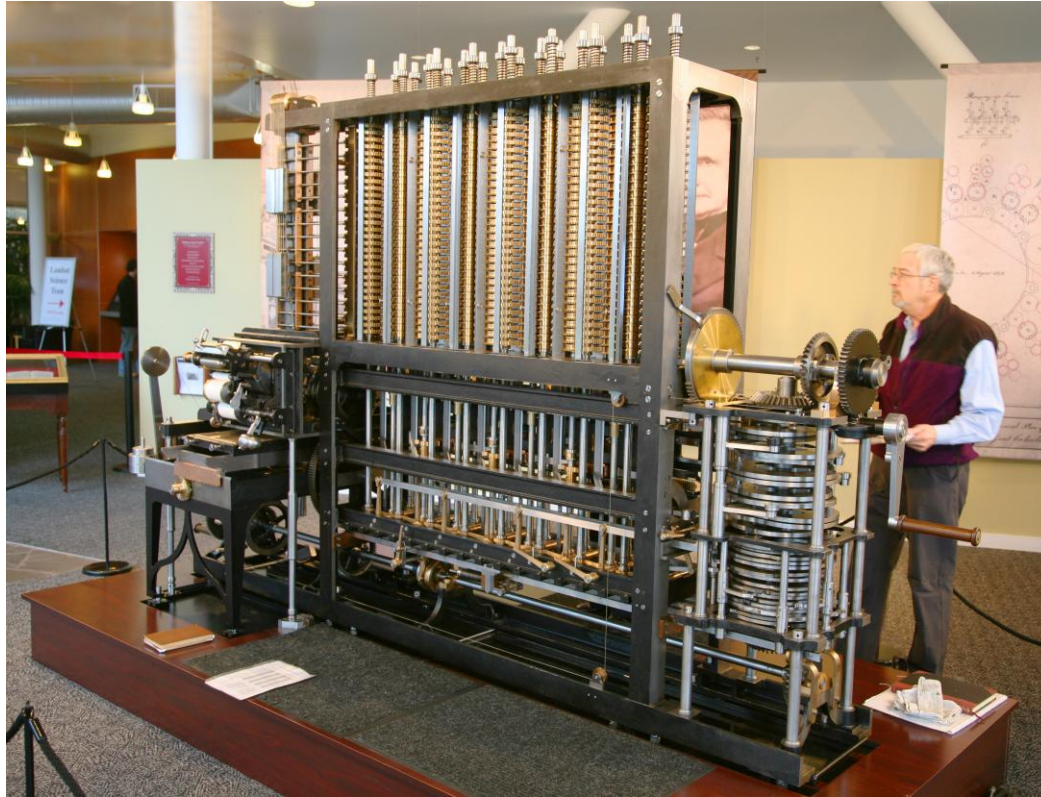


[Curta Simulation Video](#)

Difference Engine

- First theorized in the late 1700s.
- Could tabulate polynomial functions.
- Prototype (Difference Engine No. 0) was completed by Charles Babbage in 1822
- Difference Engine No. 1 was planned and funded but not completed.
- Difference Engine No. 2 was designed but never built.

Difference Engine



[Difference Engine Simulation Video \(Part 1\)](#)

[Difference Engine Simulation Video \(Part 2\)](#)

[Difference Engine Simulation Video \(Part 3\)](#)

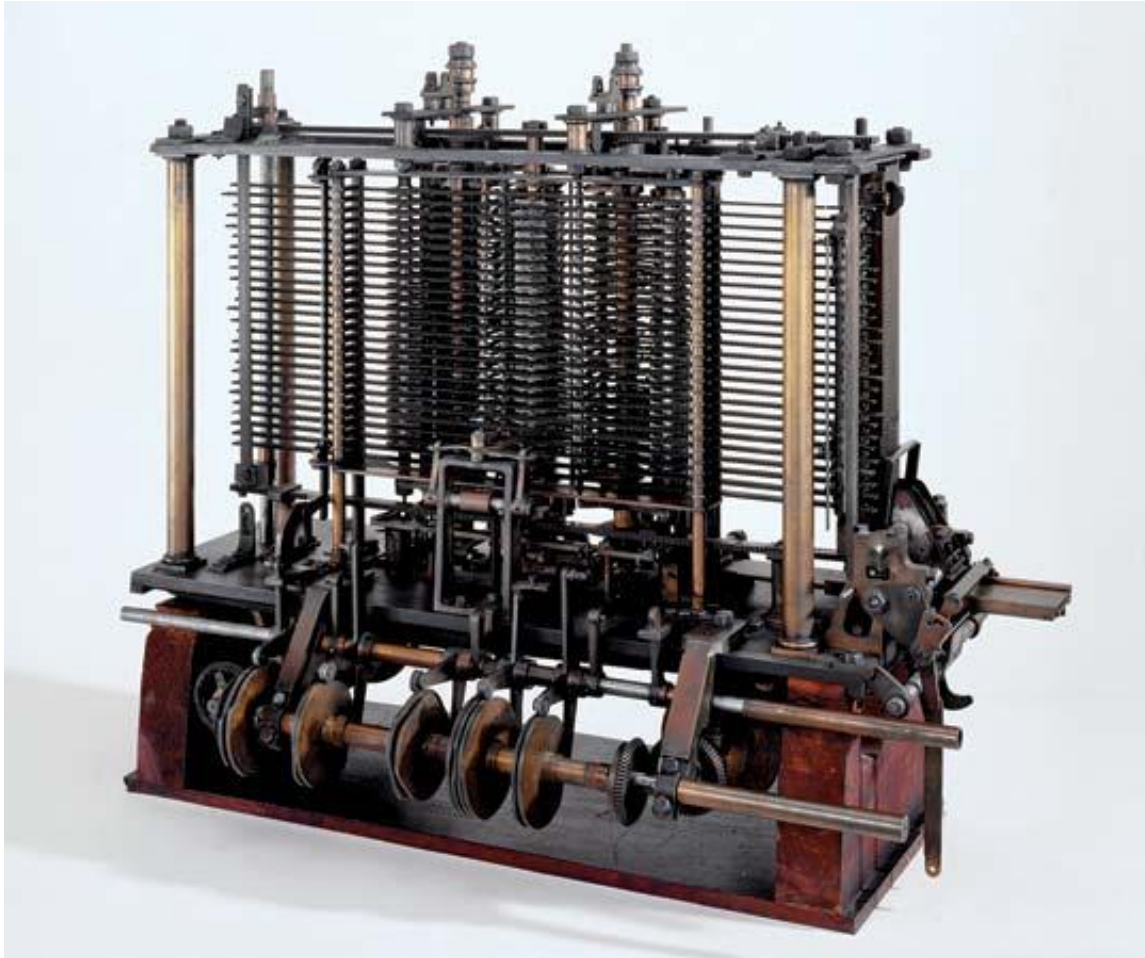
[Difference Engine Simulation Video \(Part 4\)](#)

[Difference Engine Simulation Video \(Part 5\)](#)

Analytical Engine

- Babbage's next project.
 - Started in the 1830s
- Designed to be general-purpose and programmable.
 - Programmed using cards with holes in them (***punched cards***)
- Construction was never completed.
 - It was ahead of its time.
 - The first working general-purpose computers wouldn't come around for another 100 years.

Analytical Engine



[Analytical Engine Simulation Video](#)

Harvard Mark I

- Designed by Howard Aiken/IBM
 - 1930s – 1940s
- Automatic, general-purpose and programmable.
- Among the first to use electromechanical parts
- Computed and printed mathematical tables.
 - Influenced by the Analytical Engine.

Harvard Mark I



[Harvard Mark 1 Video](#)

[Harvard Mark 1 Video 2](#)

Vacuum Tubes

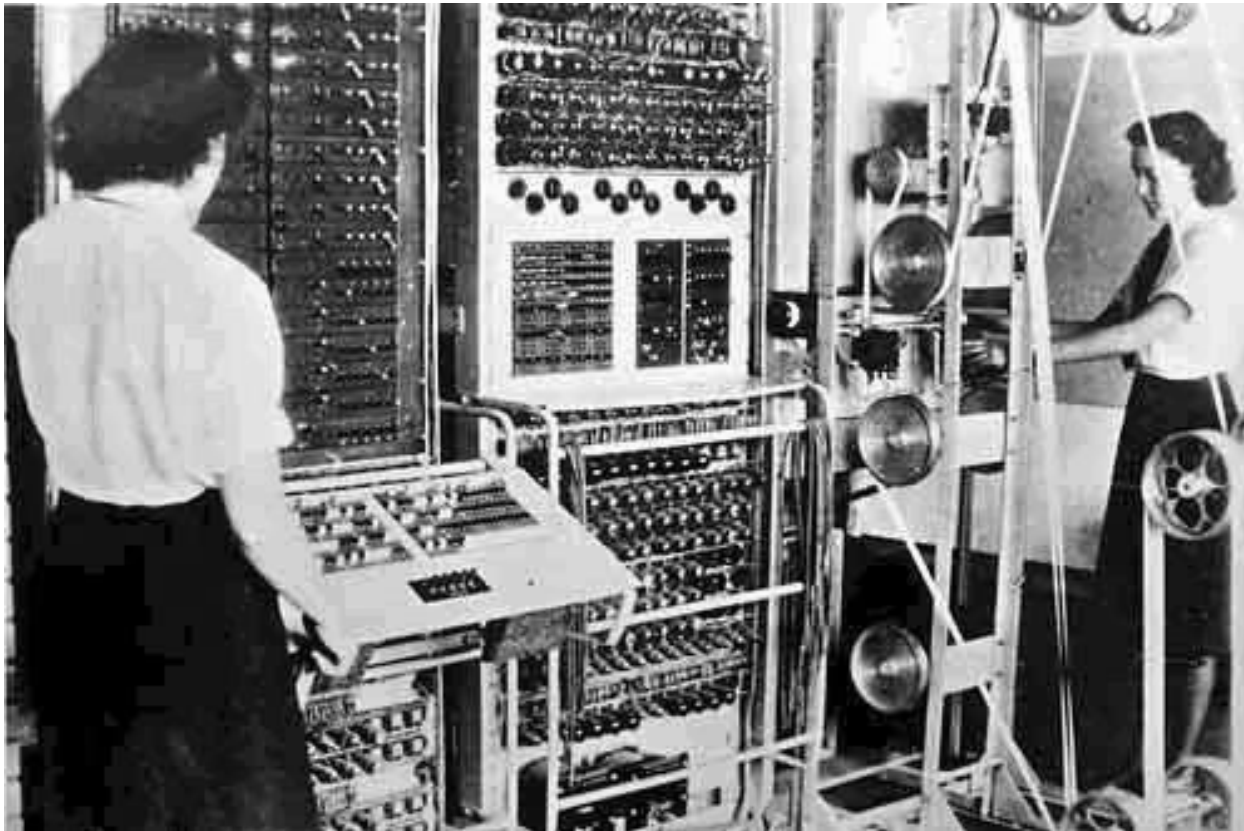
- Replaced the slower electromagnetic relays in early electronic computers.
 - Also used in radios and televisions.
- Controlled electrical flow, like a switch.
- There is a good description of how vacuum tubes work in the transistors video shown later in the lecture.



Colossus

- Designed by Tommy Flowers at Bletchley Park in the 1940s
- Used for codebreaking, specifically the Lorenz cipher, during World War II.
- First electronic, digital, programmable computer.
 - Did not store programs in memory.
 - Was not general-purpose.
- Was not public knowledge until the 1970s.

Colossus



[Colossus Video](#)

ENIAC

- Electronic Numerical Integrator And Computer
- Created by John Mauchly and John Presper Eckert (1940s)
 - Built at the University of Pennsylvania
 - February 15th was declared “ENIAC Day” in Philadelphia in 2011.
- Electronic, digital, general-purpose, programmable with stored programs.
 - Wasn’t a secret or undiscovered like Colossus or other electronic computers at that time.

ENIAC



[ENIAC Video](#)

UNIVAC

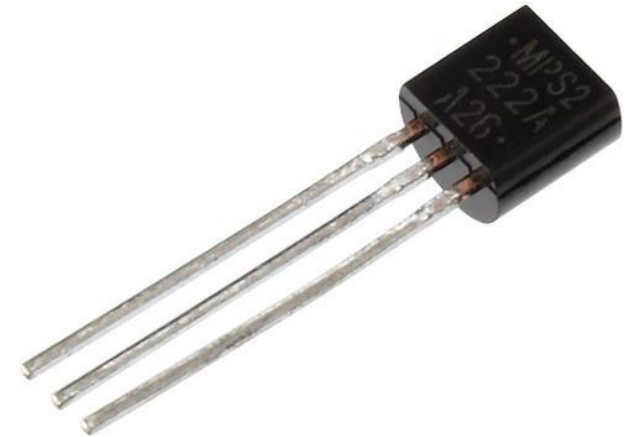
- A family of Universal Automatic Computers
- Created by John Mauchly and John Presper Eckert (also in the 1940s)
 - Formed their own company
 - Later bought by Remington-Rand.
- First commercially successful electronic, digital, general-purpose, and programmable computers.
 - Originally intended for the Census Bureau.
 - Produced through the 1970s
- Used magnetic tape for memory.
 - Could read punched cards to tape.

UNIVAC



Transistors

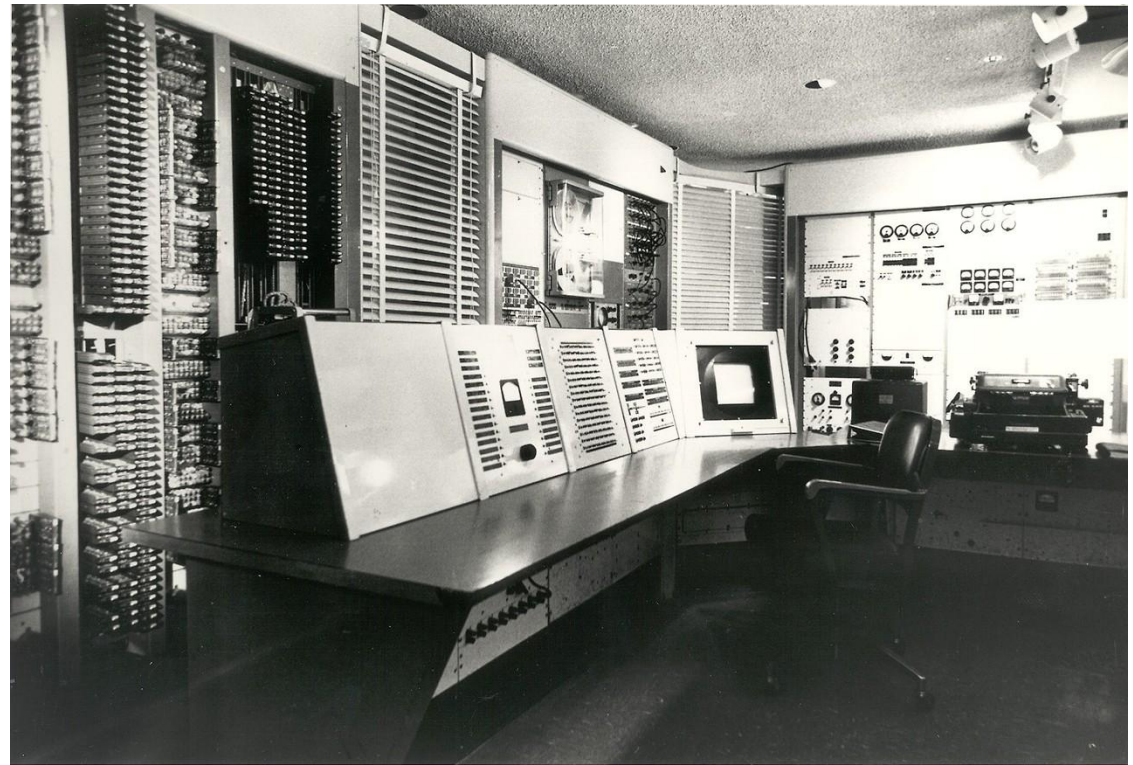
- Replaced vacuum tubes.
- Can control the amount of current or voltage, amplification/modulation or switching of an electronic signal.
- The transistor became the primary building block of computing and electronic equipment as we know it today.
- Magnetic System Memory



[Transistors Video](#)

TX-0

- Fully transistorized computer.
 - Created at MIT (1956)
- Experimental



DEC and the PDP

- Digital Equipment Corporation (1950s – 1990s)
- Created a transistorized line of computers, starting in 1961.
- Programmed Data Processor (PDP)
 - Commercially available

PDP-1

- First of DEC's PDP series.
- A lot of software firsts
 - First computer games
 - First text editor
 - First word processor
 - First debugger
 - First computerized music
- Focused on the user

[PDP-1 Video](#)



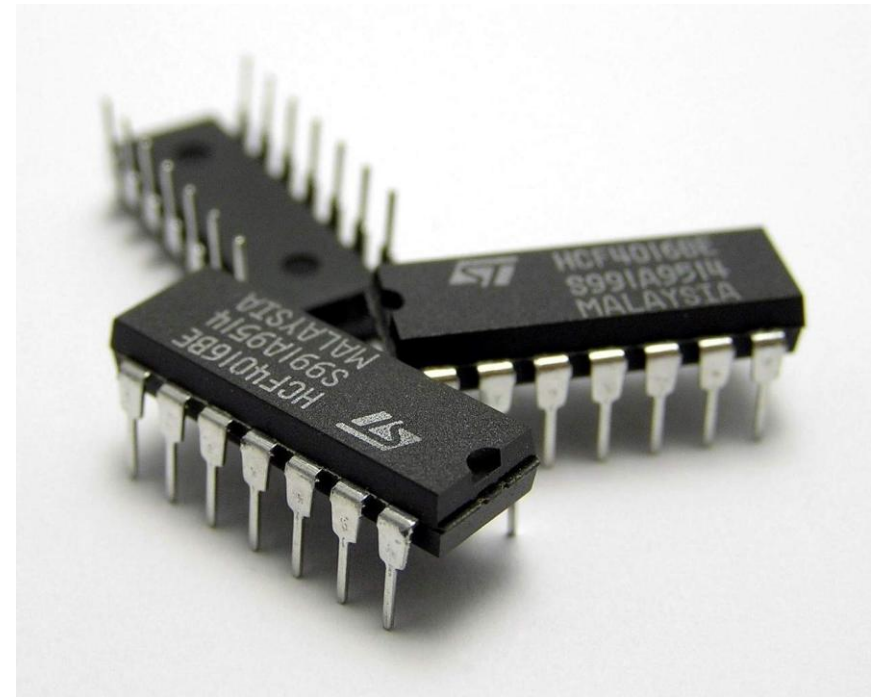
PDP-8

- First commercially successful minicomputer.
 - 1965
- Affordable and accessible to new businesses.
- Focused on the value.



Integrated Circuits

- Holds a large number of transistors on a single “chip”.



[Integrated Circuits Video](#)

IBM 360

- Designed for commercial and scientific applications.
 - 1960s – 1970s
- Scalable
 - Add or remove components
- Software written for one could be run on others.
- Several programs in memory at once.



[IBM 360 Video](#)

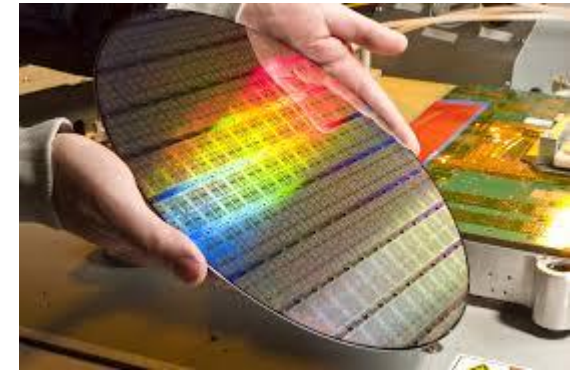
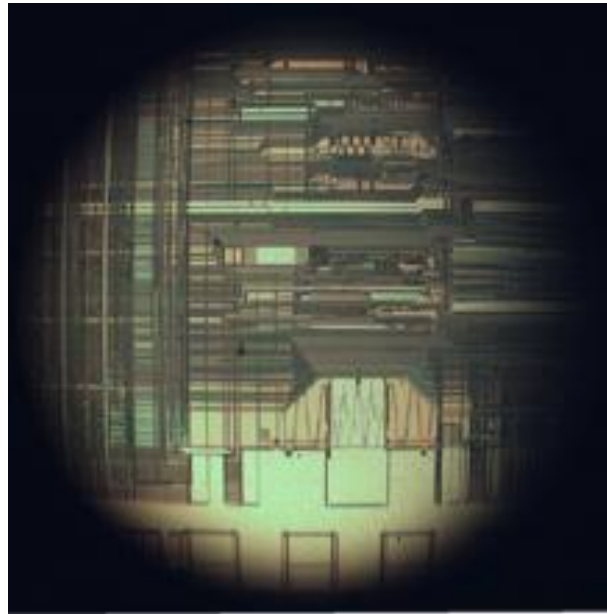
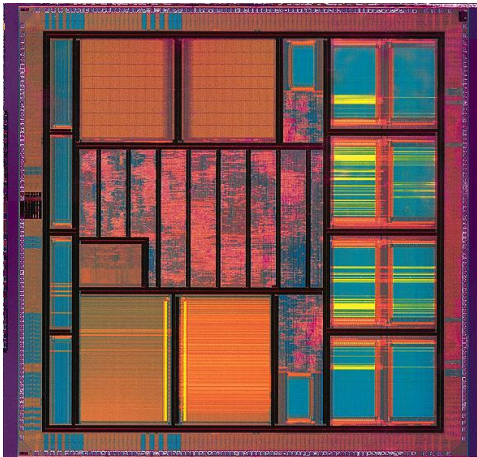
PDP-11

- DEC's most successful minicomputer.
 - 1970s to 1990s
- Large number of peripheral devices.
- Could be upgraded/scaled as needed.



Microprocessors and VLSI

- Billions of nanoscale transistors and multiple integrated circuits in one device.
 - This process is mentioned at the end of the Integrated Circuits Video.



Personal Computers



Major Types of Software

- Application Software
 - Programs that make your computer useful.
 - Word processors, Internet browsers, video games and mobile apps
- System Software
 - Programs that control the computer.
 - Operating Systems, device drivers, utility programs and software development tools.

What is a computer programming language?

- A ***programming language*** is a formal language that consists of a set of instructions that cause a computer to execute a series of operations or tasks.
 - A group of instructions that completes some task is a ***computer program***.
- There are, in general, two major types of programming languages: low-level and high-level.

What is a low-level programming language?

- A **low-level programming language** is one where the instructions are (or closely related to) the instructions for the processor/CPU.
 - The language may only work for a specific processor or other hardware.
- Usually refers to assembly language or machine code.
- Difficult to program with.
 - With machine code, it's typically done in binary.
 - Assembly language maps the binary instructions to somewhat less vague instructions. An *assembler* translates those instructions back to binary.

```
; Global declarations
STATUS equ 3 ; Status register is File 3
C equ 0 ; Carry/Not Borrow flag is bit0

cblock 20h ; Number: high byte, low byte
NUM:2
endc

MAIN goto SQR_ROOT

; *****
; * FUNCTION: Calculates the square root of a 16-bit integer *
; * EXAMPLE : Number = FFFFh (65,535d), Root = FFh (255d) *
; * ENTRY : Number in File NUM:NUM+1 *
; * EXIT : Root in W, NUM:NUM+1; I:I+1 and COUNT altered *
; *****

; Local declarations
cblock
I:2, COUNT ; Magic number hi:lo byte & loop count
endc

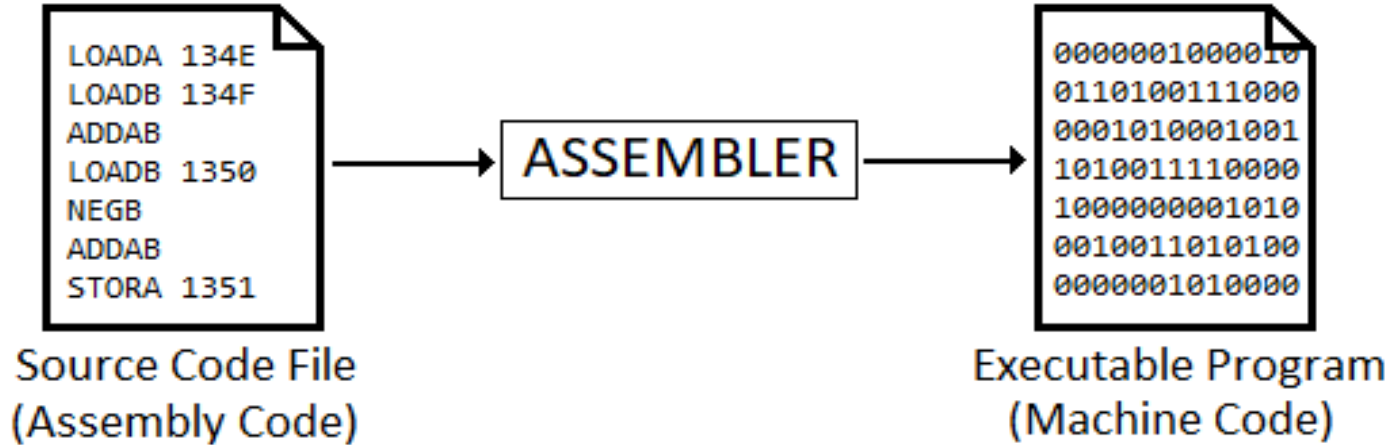
SQR_ROOT org 200h ; Code to begin @ 200h in Program store
clrf COUNT ; Task 1: Zero loop count
clrf I ; Task 2: Set magic number I to one
clrf I+1
incf I+1,f

; Task 3: DO
SQR_LOOP movf I+1,w ; Task 3(a): Number - I
subwf NUM+1,f ; Subtract lo byte I from lo byte Num
movf I,w ; Get high byte magic number
btfss STATUS,C ; Skip if No Borrow out
addlw 1 ; Return borrow
subwf NUM,f ; Subtract high bytes

; Task 3(b): IF underflow THEN exit
btfss STATUS,C ; IF No Borrow THEN continue
goto SQR_END ; ELSE the process is complete
incf COUNT,f ; Task 3(c): ELSE inc loop count
movf I+1,w ; Task 3(d): Add 2 to the magic number
addlw 2
btfsc STATUS,C ; IF no carry THEN done
incf I,f ; ELSE add carry to upper byte I
movwf I+1
goto SQR_LOOP

SQR_END movf COUNT,w ; Task 4: Return loop count as the root
return
end
```

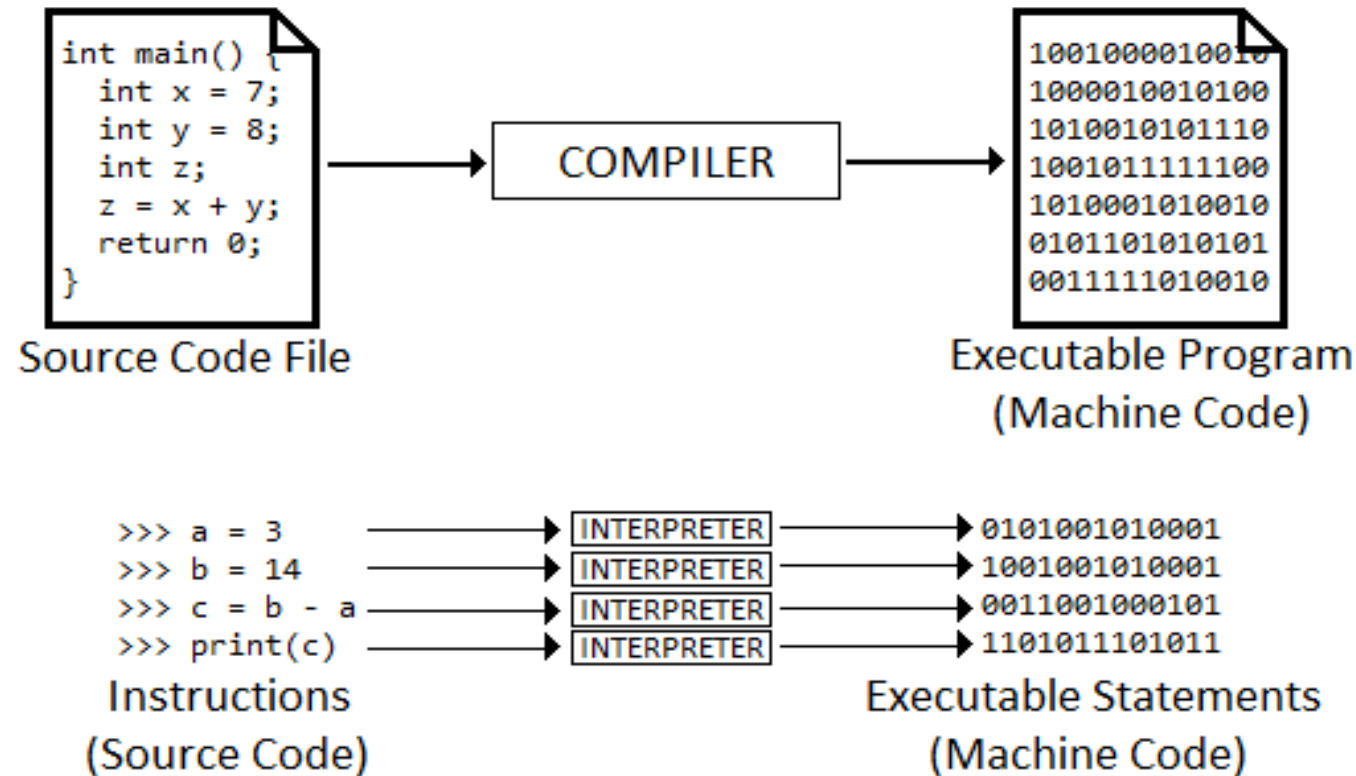

Assembly Language



What is a high-level programming language?

- A ***high-level programming language*** is one where the instructions read more closely to a human language.
 - Normally, the language will work for a variety of different platforms/processors.
- Programs in high-level languages are easier to read, write and update when compared to programs written in a low-level language.

Compilers and Interpreters



How are programming languages classified?

- Most programming languages follow a *paradigm* or style of how instructions are written.
- The two most common types are:
 - ***Procedural Programming*** which seeks to break computer programs into separate routines or *procedures* that are sent to the processor to be executed.
 - ***Object-Oriented Programming*** which seeks to break computer programs into self-contained objects that use fields and methods to manipulate the program's data.
- Some languages, like Java, are multi-paradigm.

What is Java?



- A high-level, object-oriented programming languages.
- Created by a team led by James Gosling at Sun Microsystems
 - Started development in 1991.
 - Released in 1995.
- Originally called Oak, then Green, then finally Java.
- Sun Microsystems (and Java) was purchased by Oracle in 2010.

How does Java work?

- Java source code is written by a programmer.
- Java source code is compiled to bytecode.
- The compiled bytecode is executed in a virtual machine.
 - The ***Java Virtual Machine (JVM)*** interprets the bytecode to machine code.

What is a Virtual Machine?

- A ***virtual machine*** (or ***VM***) is a software application implemented in such a way that it functions like a physical computer.
- Two types: System VM or Process VM
 - System VMs: Acts like a complete physical computer; the system's hardware is emulated by software.
 - Some popular System VMs are VMWare and VirtualBox.
 - Process VMs: A VM that only runs a single program.
 - Java utilizes a Process VM

Why Use Virtual Machines?

- The use of a virtual machine allows Java applications to function across a variety of hardware platforms and operating systems.
- There is no need to rewrite or recompile programs for, say, Windows and OSX.
- The virtual machine (for each platform) interprets the Java bytecode for that platform.

What do I need to run Java programs?

- Java Runtime Environment (JRE)
 - Includes the JVM for running Java programs.
- Downloadable for free.

What do I need to develop Java programs?

- Java Development Kit (JDK)
 - Includes all features of the JRE and includes the Java compiler.
 - We can't create executable Java programs without the compiler.
- Java source code can be written in a text editor, like Notepad.
 - Better tools exist to simplify the development process.
- More powerful IDEs exist for developing Java applications.
 - Visual Studio Code – We will be using this.
 - Eclipse, IntelliJ are other popular IDEs

How much does it cost to develop Java programs? Do I need a developer's license?

- Nothing and No!
- The JDK is available for free.
 - Anyone can develop Java programs.
- Visual Studio Code is also free.

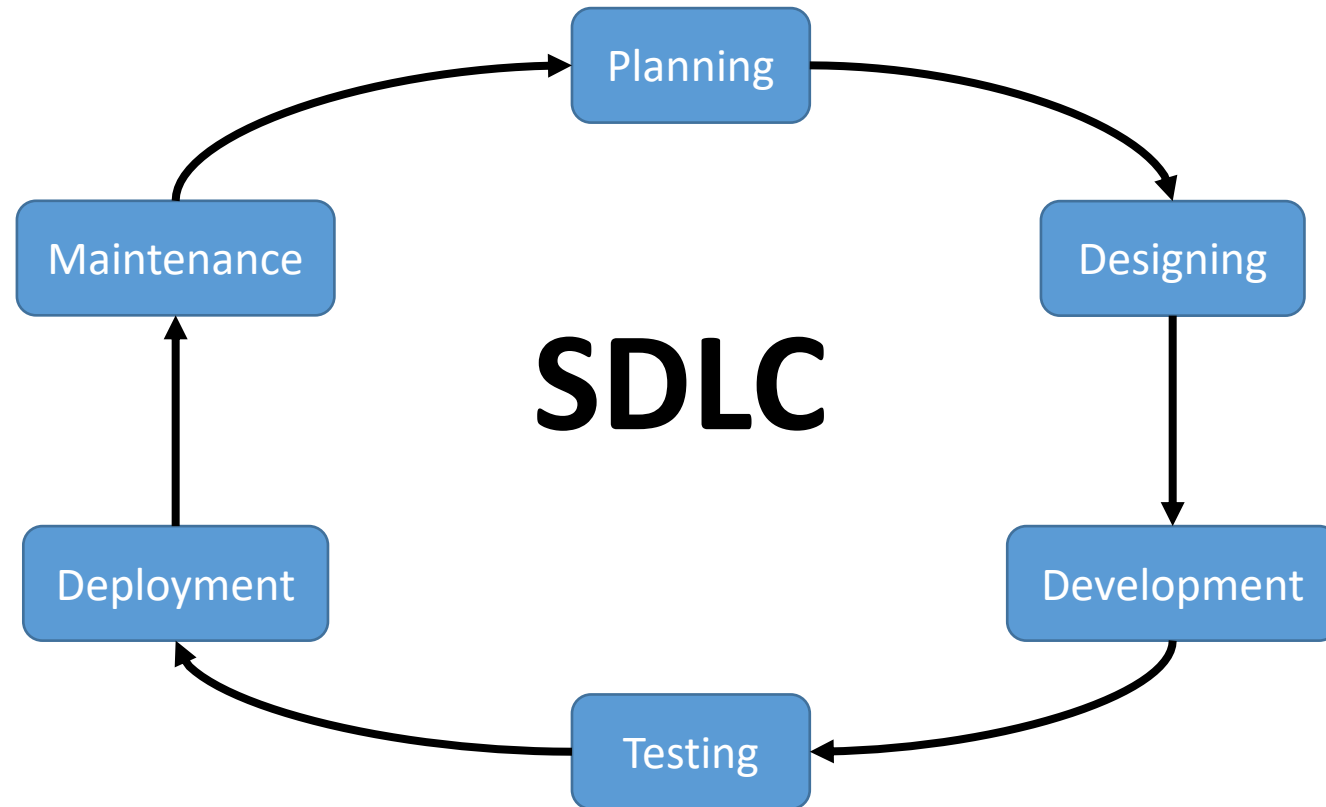
What are the different editions of Java?

- JavaSE – Standard Edition (What we will be using)
- JavaEE – Enterprise Edition (Enterprise Environments)
- JavaME – Micro Edition (Embedded Systems)

The Software Development Life Cycle

- The ***Software Development Life Cycle*** (SDLC) is a process to produce computer software.
 - Highest Quality
 - Lowest Cost
 - Shortest Time
- Consists of (normally) six stages.

The Software Development Life Cycle



SDLC Stage 1 – Planning

- The Planning Stage involves input from all project stakeholders to determine the project's objective.
 - The Customer
 - Senior Management
 - Sales/Marketing
 - Technical Experts
- This is also when an estimate of resources and costs is determined.
 - Equipment, labor, etc.



SDLC Stage 1 – Planning

- The Planning Stage is sometimes called a ***requirement analysis***.
 - What do we want?
 - What don't we want?
- Risks
 - Is the project's timeline feasible?
 - Does the technology exist?
 - Is the cost too high?
 - **Minimize Risk**



SDLC Stage 1 – Planning

- Near the end of the Planning Stage, the requirements of the product will need to be formalized.
- A **Software Requirement Specification (SRS)** document will outline what functionality the product should have.
 - Requirements should not be ambiguous.
 - “Good User Interface”
 - This document should be reviewed and approved by stakeholders.



SDLC Stage 2 – Designing

- The Designing Stage involves creating the overall architecture of the application.
- **Design Document Specification (DDS)** documents will contain different design approaches for the architecture.
 - Is based on the SRS
 - With input from stakeholders, the best design approach is selected.
- Each approach should:
 - Identify the separate components of the architecture.
 - Identify how the components will work together.
 - Ensure the application's requirements are met.



SDLC Stage 2 – Designing

- The DDS should also contain a list of milestones
 - What will be completed in certain timeframes?
- Functionality of the application should be detailed.
 - User Interfaces
 - Failure
 - Limitations
- Misunderstandings will cause problems later.



SDLC Stage 3 – Development

- With the requirement analysis and design document complete, software development can begin.
 - The better requirements were defined in the previous stages, the easier it will be for the programmers to create the actual product.



SDLC Stage 4 – Testing

- After development is complete, the product needs to be tested.
- While testing is performed by programmers as they develop, a formal test procedure or test plan must be created.
 - The test plan should incorporate testing the features and functions described in the DDS.



SDLC Stage 4 – Testing

- Some organizations have entire departments (***Quality Assurance*** or ***QA***) devoted to testing.
- QA testers follow the test plans to ensure the product works as intended.
 - Programming teams are notified if the testers discover issues.
- QA testers will also try to find and report any odd or abnormal behavior (*glitches*) in the product/application.



SDLC Stage 5 – Deployment

- After the product has passed all tests and is determined to function as designed, the product is ready to be delivered to the customer.
- Often, the deployment stage will involve teams who visit the customer on-site to install and configure hardware/software.
 - Will work closely with the customer's IT staff.
 - Ensures the product was delivered and is working correctly.



SDLC Stage 6 – Maintenance

- Problems may arise after deployment.
 - Issues not anticipated or discovered during testing.
- The customer will often be provided with an update or ***software patch*** that fixes the problem.
- Customer Support services may be offered.
 - Product support may have ***end-of-life*** terms.



What next?

- If this was a one-time software solution, the product and SDLC is complete.
- Normally, this isn't the end.
 - After getting customer feedback and patching problems, work for the next version of the software can be started.
 - The cycle begins again at the Planning Stage.

Developing Software

- During the Development stage (Stage 3) of the SDLC, the programming team will begin by reviewing and understanding the DDS.
 - Sometimes, this is the responsibility of a software development manager.
- Different parts of the application will be assigned to different team members.
 - Usually matched with their ability/expertise.

Developing Software

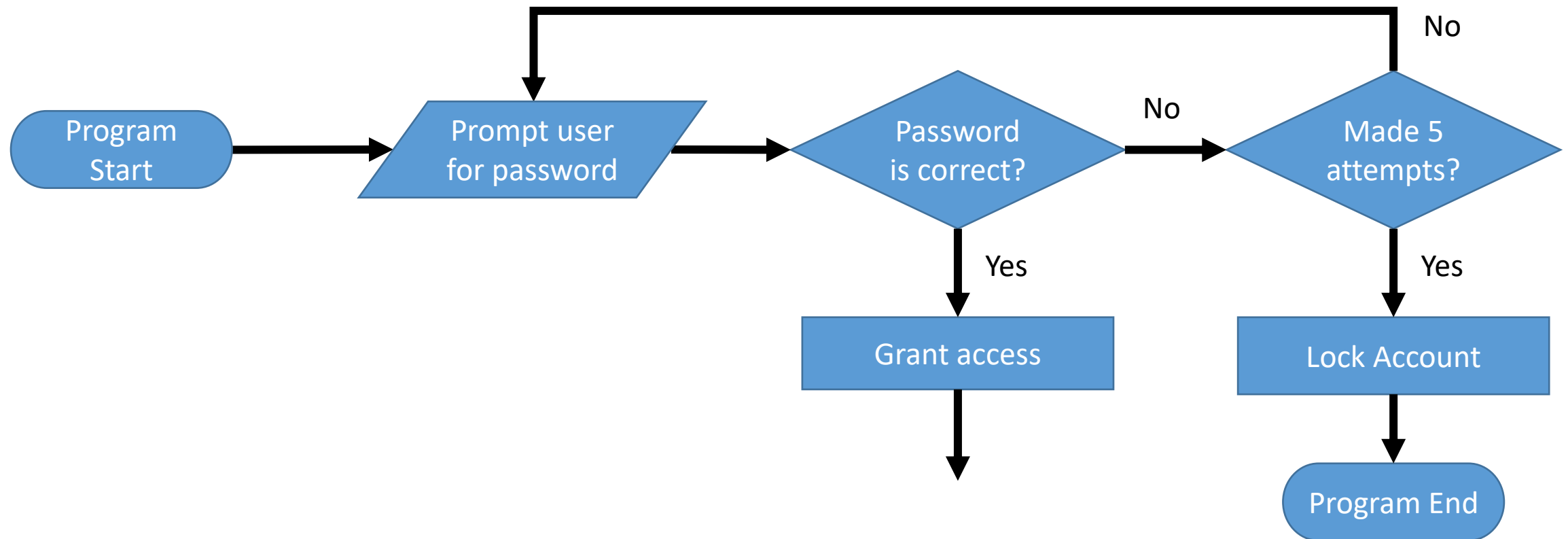
- Programmers use a variety of non-programming techniques when developing software.
- A programmer must have a plan **before** they write a single line of code.

“Plans are worthless, but planning is everything.”



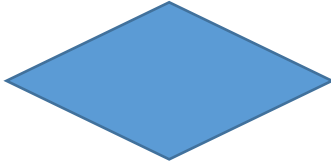


- Dwight Eisenhower

Developing Software – Flowcharts

- Drawing flowcharts is a great way to aid in your planning by visualizing processes.



Developing Software – Flowchart Symbols

- Oval – Program start or stop 
- Rectangle – Process 
- Diamond – Decision 
- Input/Output – Parallelogram 
- Arrows – Direction of Flow 

Developing Software – Pseudocode

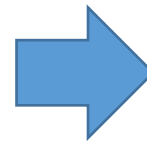
- Based on the programmer's notes and flowcharts, a “script” of how the program should work can be written.
 - The script will contain the step-by-step processes completed by the program.
 - The processes are often written in plain text, mixed with actual programming code.
- This is referred to as ***pseudocode***.
 - It's not really valid, working code; Serves as a guide for how the actual code will be written.

```
Ask user for password
while user_guess != password :
    Print error message
    attempts += 1
    Check if too many attempts
        Print error message / Stop program
    Ask for password again
...
```

Developing Software – Programming

- With the completed flowcharts and pseudocode scripts, the programmer can begin to write the actual program.
 - You've already drawn/written out exactly (or pretty close, at least) how the program should function.
 - The flowcharts and pseudocode act as a road map of all the steps the program needs to take to complete its task.

```
Ask user for password
while user_guess != password :
    Print error message
    attempts += 1
    Check if too many attempts
        Print error message / Stop program
    Ask for password again
...
```



```
user_guess = input("Enter password: ")
while user_guess != password :
    print("Invalid Password.")
    attempts += 1
    if attempts == 5 :
        print("Login attempts exceeded.")
        exit()
    user_guess = input("Try Again: ")
```

Developing Software – Documentation

- Programmers will document their code using ***comments***.
 - Comments are notes that explains the “why’s” and “what’s” of the code.
- Other programmers may not understand what certain statements are doing, why they are there, and/or why they are important.
 - YOU might even forget why you have certain statements in the program.
- Properly documented code makes debugging, maintenance, and working as a team easier.
 - It also shows me that you understand what your statements are doing and why you wrote those statements.

Developing Software – Testing

- As the programmer develops the program, he or she must test that the functionality works correctly.
 - Many programmers will develop iteratively- create or change code and test it, create or change more code and test it, and so on.
- A programmer may encounter a few types of errors during the development process.
 - A ***compile-time error*** is an error that occurs when the program is compiled into machine code.

Developing Software – Testing

- A ***run-time error*** is an error that occurs while the program is running, causing the program to crash.
 - When a program crashes, the program will stop executing its statements.
- The source of a run-time error can sometimes be difficult to pinpoint and can require considerable time to solve.
 - When a run-time error occurs, it will often provide some details to help track down the cause.

Developing Software – Testing

- You may, during testing, discover your program exhibit unintentional behaviors or glitches.
- A **bug** is a colloquial term for some erroneous code, logic, or unexpected behavior in a program.
 - **Debugging** is the term used to describe the process of searching for the cause of an error or unexpected behaviors.

Developing Software – Best Practices

- Always start a program with a pencil and paper.
 - Draw flowcharts
 - Write a pseudocode script.
- Test, Test, Test.
 - Validate your program works as designed and there are no bugs.
- Manage your time effectively.
 - Expect to spend time planning, programming, and testing/debugging.