# Behind the Curtain

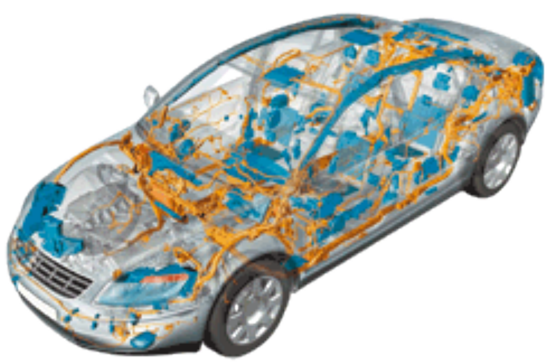
1. Computer organization
2. A look inside
3. Switches and wires
4. Memory concepts
5. Computers as systems

(In Chapter 1)

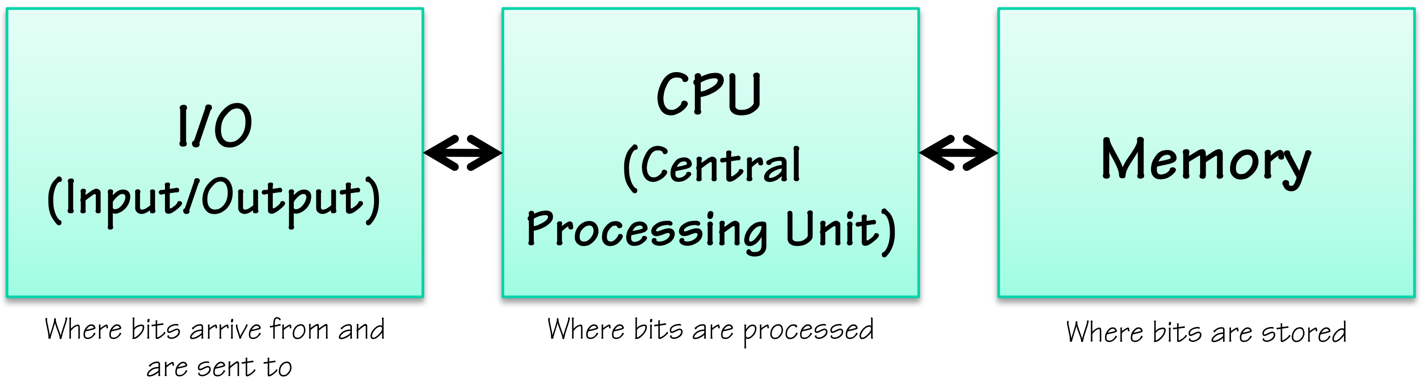
# Computers Everywhere

- The computers we are used to
  - Desktops
  - Laptops


  - Embedded processors
    - Cars
    - Mobile phones
    - Toasters, irons, wristwatches, happy-meal toys

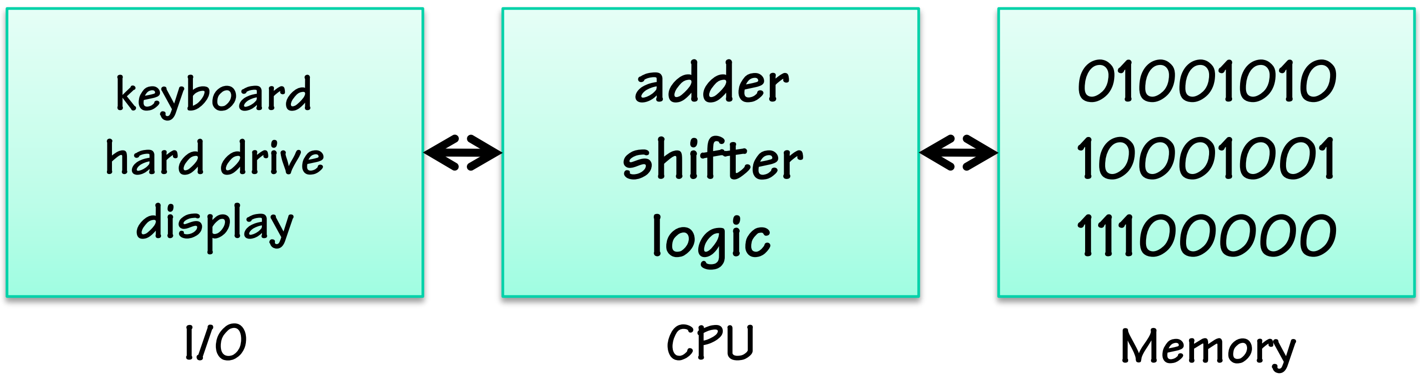



# Computer Organization




- Every computer has at least three basic units
  - Input/Output
    - where data is entered from the outside world
    - where data is displayed to the outside world
    - where data is archived for the long term (i.e. when the lights go out)
  - Memory
    - where data is stored (numbers, text, lists, arrays, data structures)
  - Central Processing Unit
    - where data is manipulated, analyzed, etc.

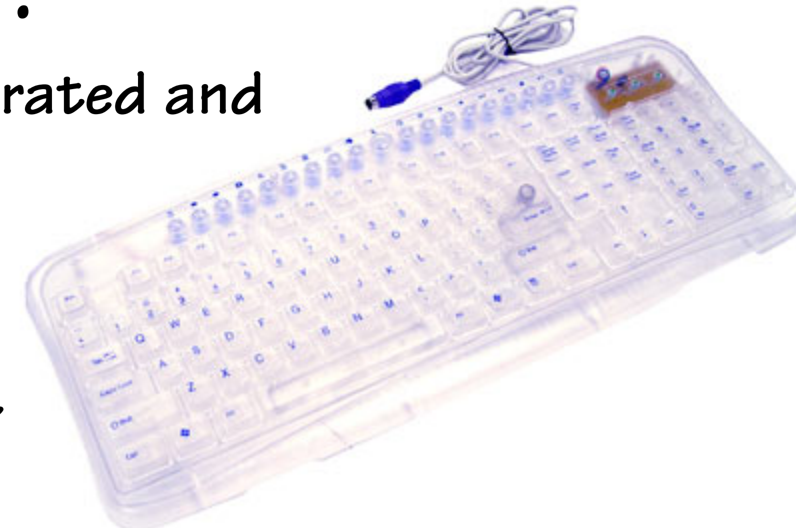
# Computer Organization (cont)

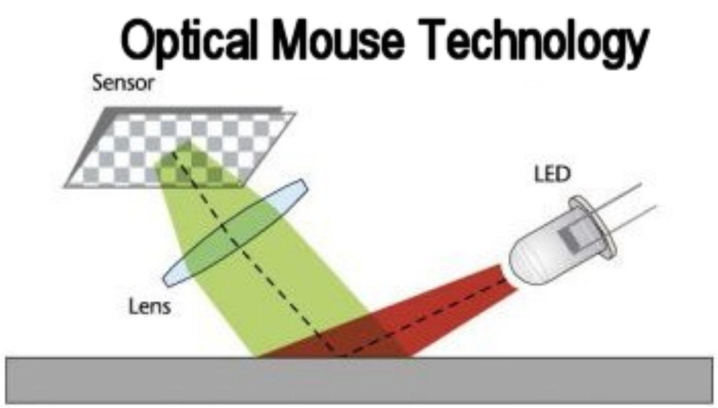



- Properties of units
  - Input/Output
    - must convert symbols to bits and vice versa
    - where the analog "real world" meets the digital "computer world"
    - must somehow synchronize to the CPU's clock
  - Memory
    - stores bit in "addressable" units, such as bytes or words
    - every memory unit has an "address" and "contents", like a mailbox
  - Central Processing Unit
    - besides processing, it also coordinates data's movements between units

# Ins and Outs

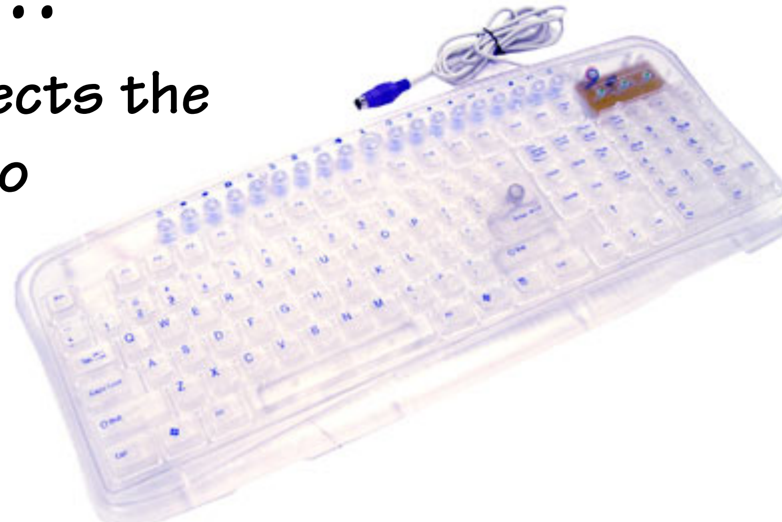
- When you press a key…
  - A binary "key code" is generated and sent to the CPU, which is interrupted, stores the key code in memory and then continues doing what it was doing

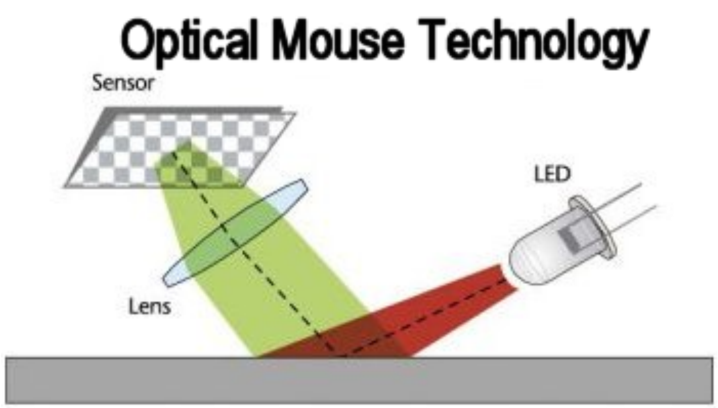





# Ins and Outs (cont)

- When you roll a mouse…
  - A dedicated computer detects the amount of movement in two orthogonal directions (X and Y), encodes them as a binary number, interrupts the CPU, who stores the values in memory

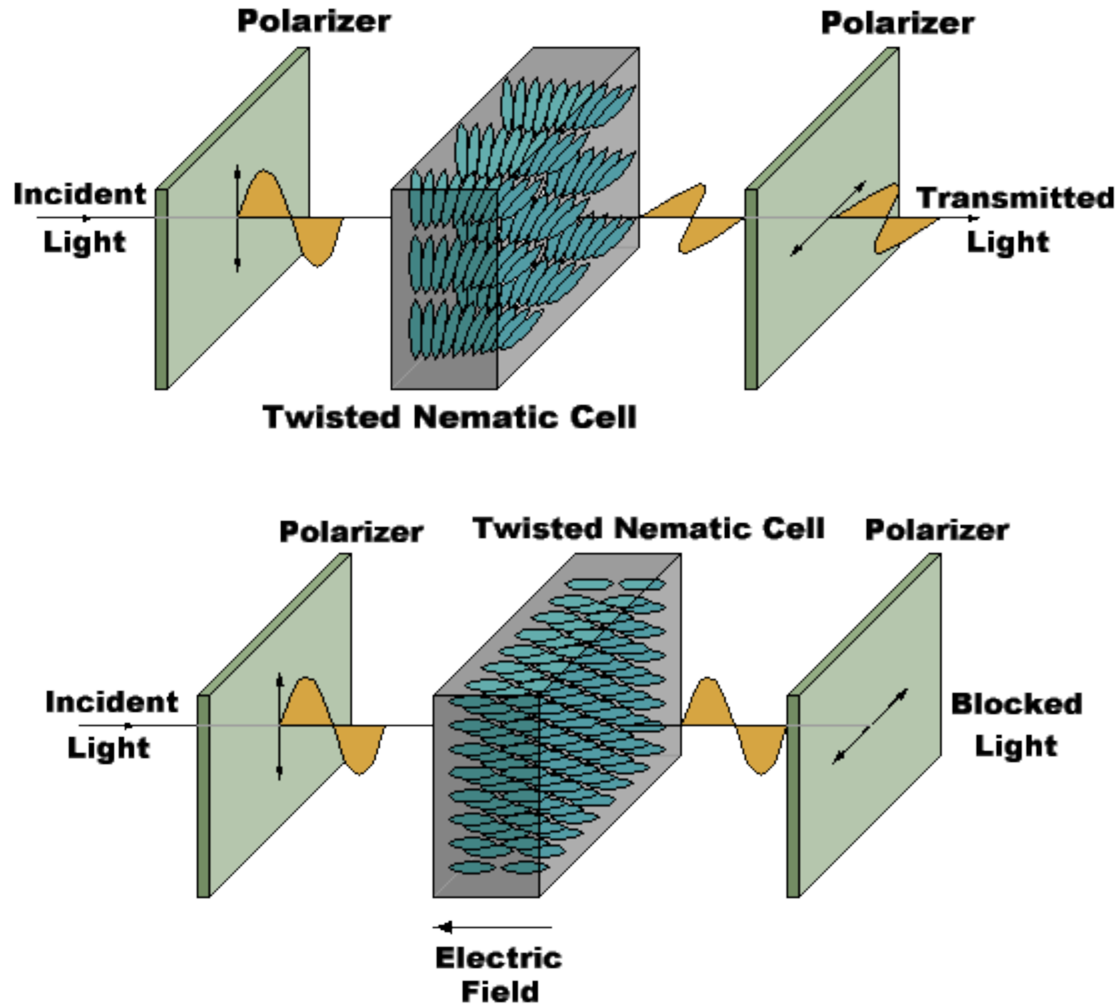





# Ins and Outs (cont)

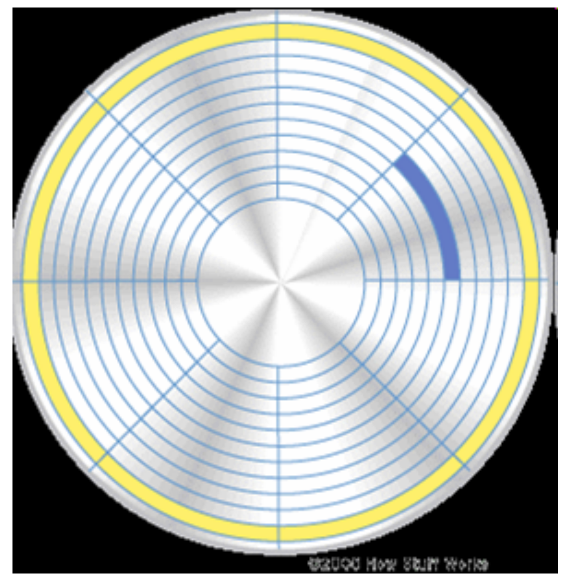
- LCD display
  - Binary numbers representing the relative amounts of red, green, and blue light at each pixel are used to modulate a polarizer that varies from transmissive to opaque.

## Liquid Crystal Displays (LCDs)



# Ins and Outs (cont)

- Hard Drive
  - "Blocks" of data are transferred to and from a spinning ferromagnetic disk. Little electromagnets on a moving "head" are used to write an read bits which are encoded by their magnetic polarity (N and S).



# Memory

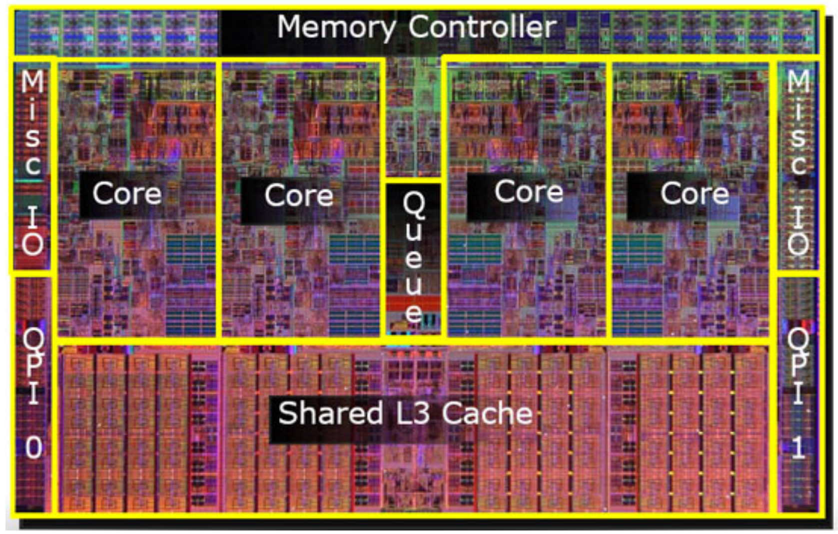
- Majority of a computer's hardware (measured in transistors)
  - Stores bits as charge on tiny capacitors
  - These capacitors "leak" and need to be "refreshed" about 60 times a second
  - They loose their charge when the power is removed
  - Trends: Capacity/Throughput

DRAM chip capacity

| Year | Size |
|---|---|
| 1980 | 64 Kb |
| 1983 | 256 Kb |
| 1986 | 1 Mb |
| 1989 | 4 Mb |
| 1992 | 16 Mb |
| 1996 | 64 Mb |
| 1999 | 256 Mb |
| 2002 | 1 Gb |
| 2004 | 4 Gb |
| 2010 | 16 Gb |

# CPUS

- Where all processing takes place, adding, multiplying, moving, etc.
- Runs at a faster speeds than either memory or I/O
- Large fraction of CPU is a special memory that "caches" frequently used data



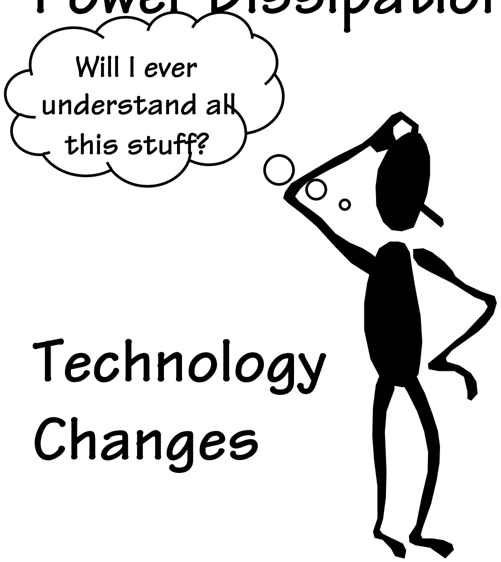

**Intel® Core i7® Extreme processor die**

- Multicore – trend towards more than one processing unit per CPU.

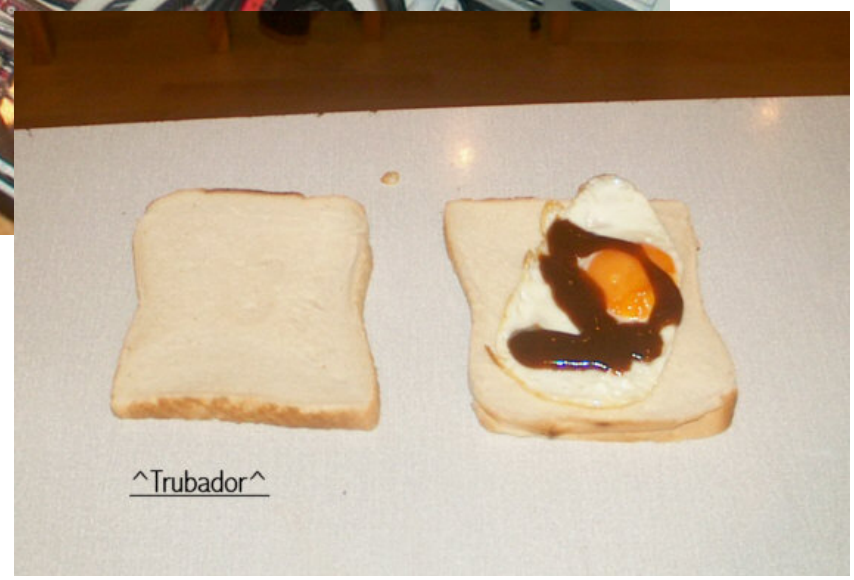
*The hottest chip you can get???*

# Issues for Modern Computers

- GHz Clock speeds
- Multiple Instructions per clock cycle
- Multicore
- Memory Wall
- I/O bottlenecks
- Power Dissipation



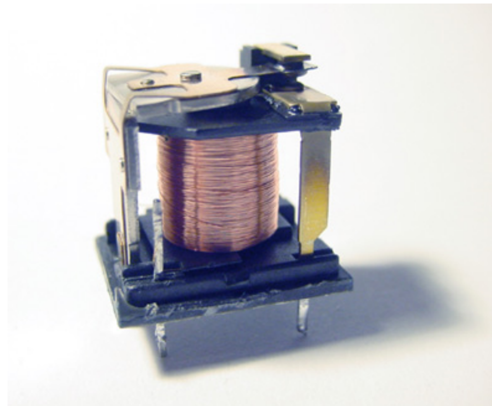

- Technology Changes

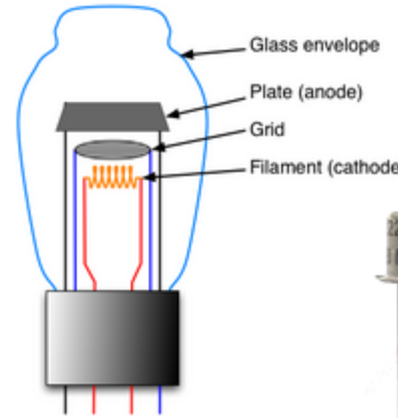


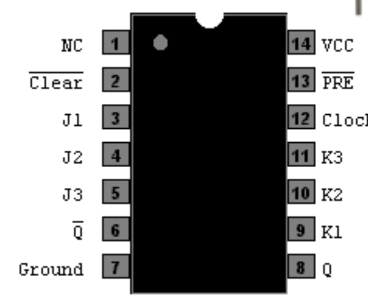

Courtesy Troubador

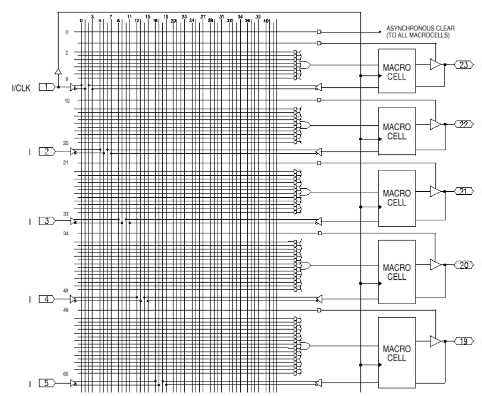
# Implementation Technology

- Relays
- Vacuum Tubes
- Transistors
- Integrated Circuits
  - Gate-level integration
  - Medium Scale Integration (PALs)
  - Large Scale Integration (Processing unit on a chip)
  - Today (Multiple CPUs on a chip)
- Nanotubes??
- Quantum-Effect Devices??

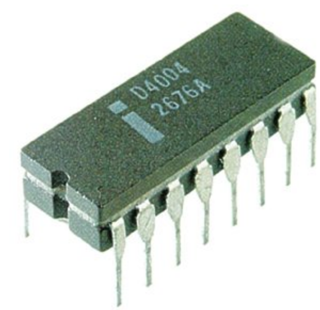




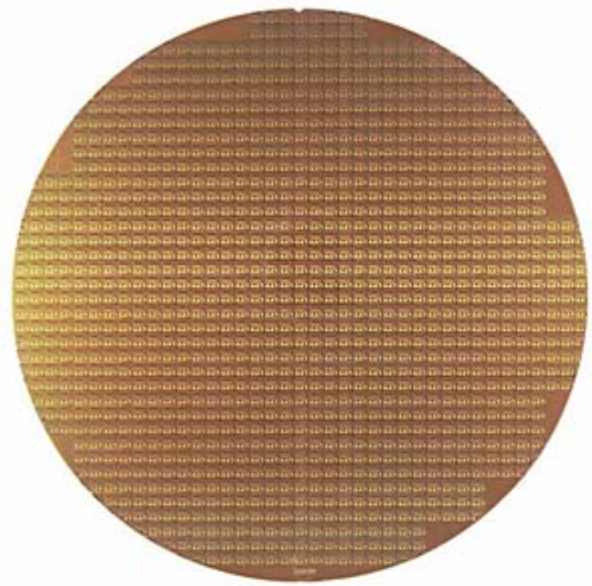



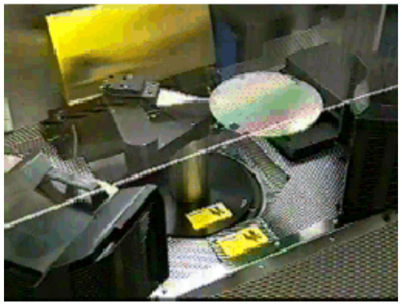




# Chips

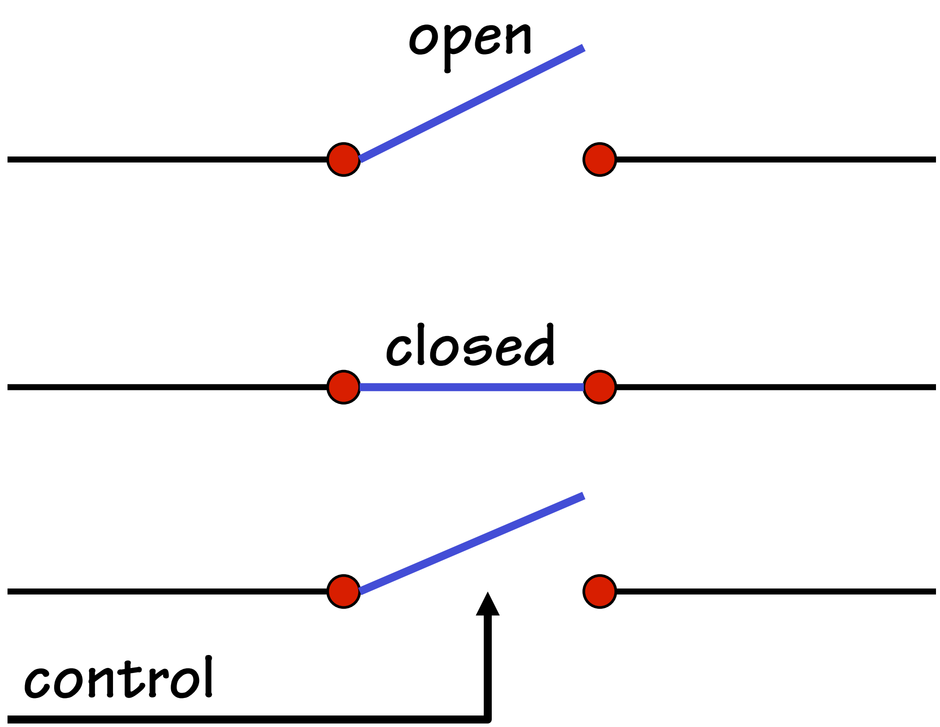
- Silicon Wafers
  - Chip manufactures build many copies of the same circuit onto a single wafer. Only a certain percentage of the chips will work; those that work will run at different speeds. The yield decreases as the size of the chips increases and the feature size decreases.
  - Wafers are processed by automated fabrication lines. To minimize the chance of contaminants ruining a process step, great care is taken to maintain a meticulously clean environment.





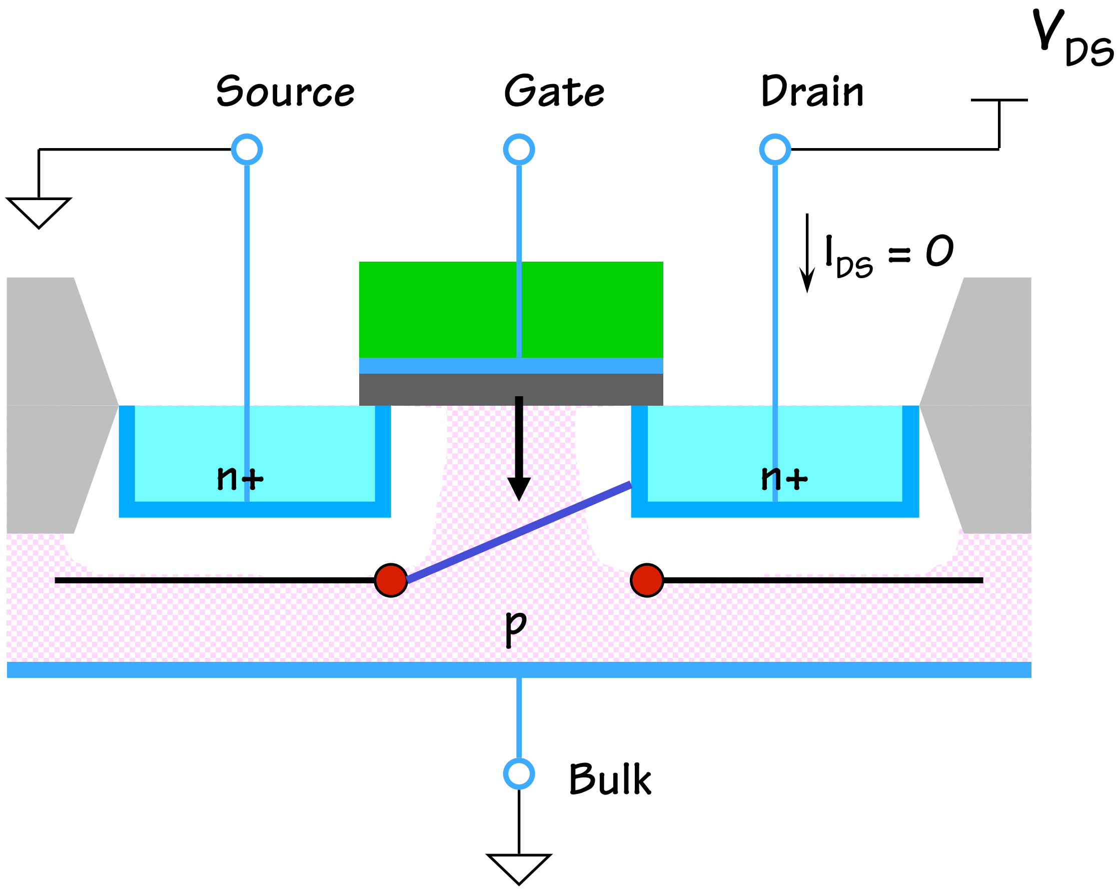
# Implementation Technology

- Common Links?
- A controllable switch
- Computers are wires and switches



# Field Effect Transistors (FETs)

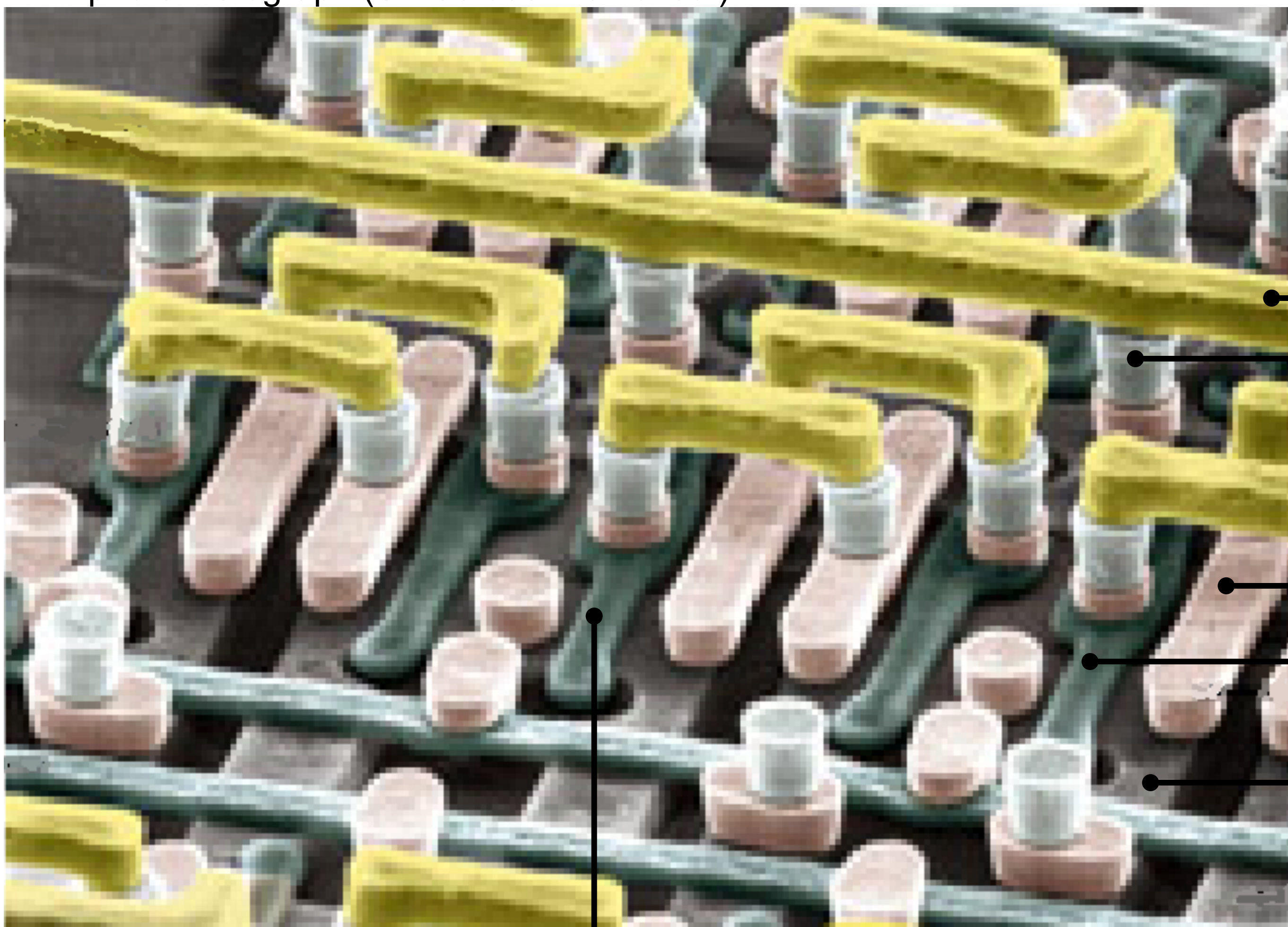
- Modern silicon fabrication technology is optimized to build a particular type of transistor. The flow of electrons from the source to the drain is controlled by a gate voltage.



# Chips

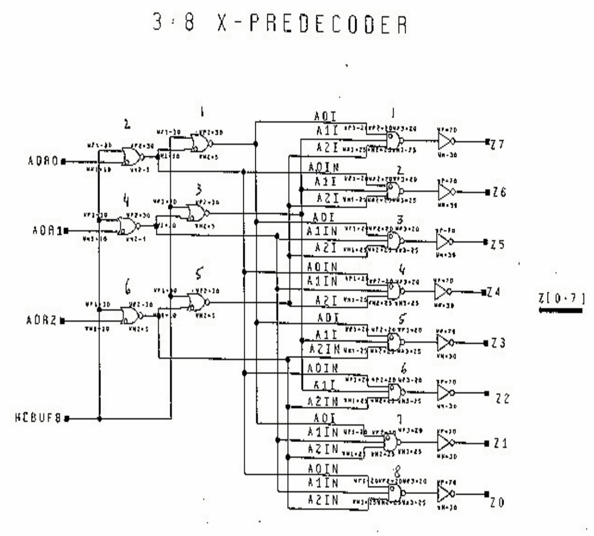
- **Silicon Wafers**

IBM photomicrograph (Si has been removed!)

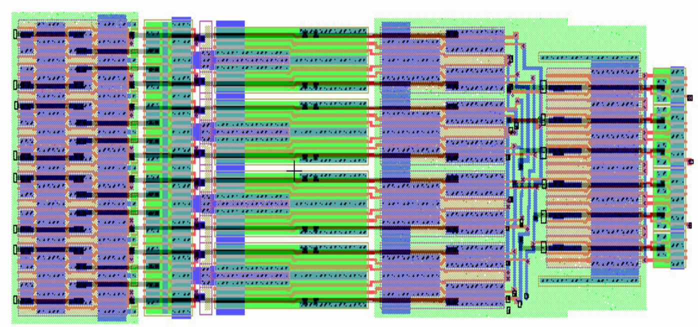


# How Computers WERE Designed

- 20 years ago
  - I/O Specification
    - Truth tables
    - State diagrams
  - Logic design
  - Circuit design
  - Circuit Layout




Chip Layout of 3:8 X Predecoder



# How Computers ARE Designed

- Today (with software)
- High-level hardware specification languages
  - Verilog
  - VHDL

## Verilog (One-Hot)

Following is the Verilog code for a 1-of-8 decoder.

```
module mux (sel, res);
  input [2:0] sel;
  output [7:0] res;
  reg [7:0] res;

  always @(sel or res)
  begin
    case (sel)
       3'b000 : res = 8'b00000001;
       3'b001 : res = 8'b00000010;
       3'b010 : res = 8'b00000100;
      3'b011 : res = 8'b00001000;
      3'b100 : res = 8'b00010000;
      3'b101 : res = 8'b00100000;
      3'b110 : res = 8'b01000000;
      default : res = 8'b10000000;
    endcase
  end
endmodule
```

## VHDL (One-Hot)

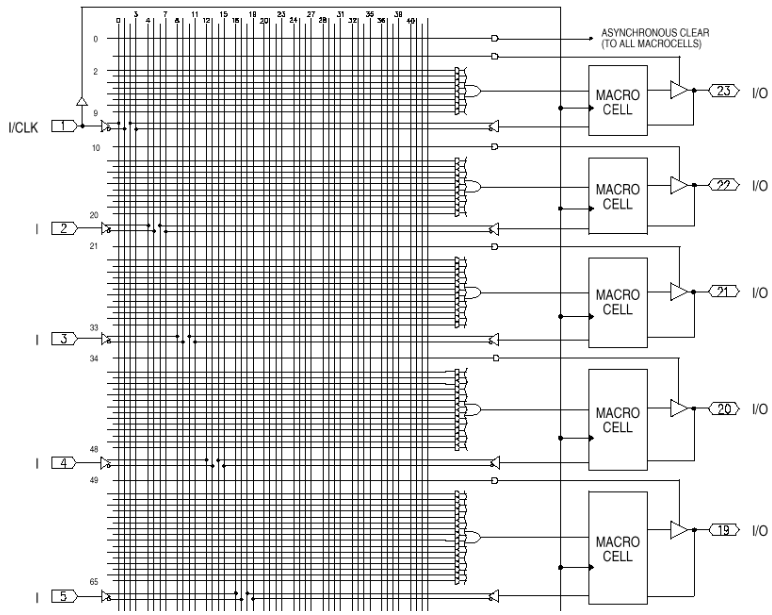
Following is the VHDL code for a 1-of-8 decoder.

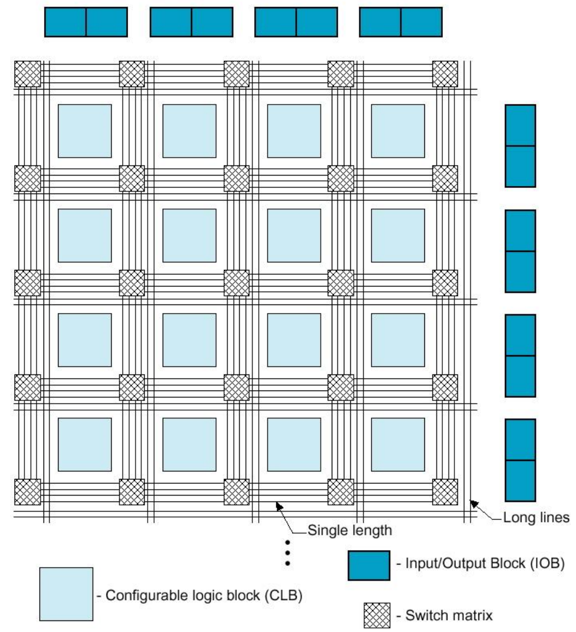
```
                library ieee;
use ieee.std_logic_1164.all;

entity dec is
  port (sel: in std_logic_vector (2 downto 0);
        res: out std_logic_vector (7 downto 0));
end dec;
architecture archi of dec is
  begin
    res <=  "00000001" when sel = "000" else
            "00000010" when sel = "001" else
            "00000100" when sel = "010" else
            "00001000" when sel = "011" else
            "00010000" when sel = "100" else
            "00100000" when sel = "101" else
            "01000000" when sel = "110" else
            "10000000";
end archi;
```

# Reconfigurable Chips

- Programmable Array Logic (PALs)
  - Fixed logic / programmable wires
- Field Programmable Gate Arrays (FPGAs)
  - Repeated reconfigurable logic cells



# Memory Concepts

- Memory is divided into addressable blocks, each with an address
- Addressable blocks are usually larger than a bit, typically 8, 16, 32, or 64 bits
- Each address has variable "contents"
- Contents might be:
  - Integers in 2's complement
  - Floats in IEEE format
  - Strings in ASCII or Unicode
  - Data structure de jour
  - ADDRESSES
  - Nothing distinguishes the difference

| Address | Contents |
|---|---|
| 0 | 42 |
| 1 | 3.141592 |
| 2 | "Lee " |
| 3 | "Hart" |
| 4 | "Bud " |
| 5 | "Levi" |
| 6 | "le " |
| 7 | 2 |
| 8 | 0c3c1d7fff |
| 9 | 0x37bdfffc |
| 10 | 0x24040090 |
| 11 | 0x0c00000e |
| 12 | 0x1000ffff |
| 13 | -100 |
| 14 | 0x00004020 |
| 15 | 0x20090001 |

# One More Thing…

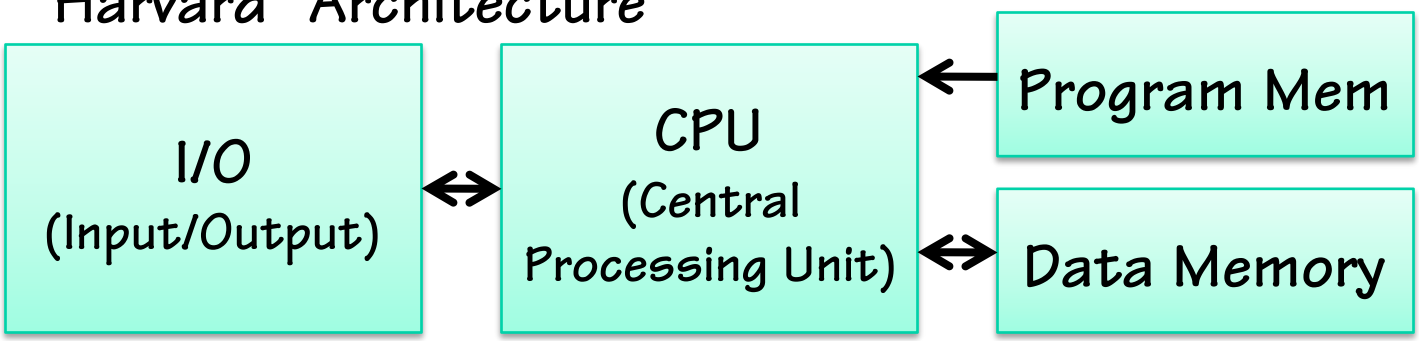
- Instructions for the CPU are stored in memory along with data
- CPU fetches instructions, decodes them and then performs their implied operation
- Mechanism inside the CPU directs which instruction to get next.
- They appear in memory as a string of bits that are typically uniform in size
- Their encoding as "bits" is called "machine language." ex: 0c3c1d7fff
- We assign "mnemonics" to particular bit patterns to indicate meanings. These mnemonics are called assembly language. ex: lui $sp, 0x7fff

| Address | Contents |
|---|---|
| 0 | 42 |
| 1 | 3.141592 |
| 2 | "Lee " |
| 3 | "Hart" |
| 4 | "Bud " |
| 5 | "Levi" |
| 6 | "le " |
| 7 | 2 |
| 8 | lui $sp,0x7fff |
| 9 | ori $sp,$sp,0x7fff |
| 10 | addiu $a0,$0,144 |
| 11 | jal 0x0000000e |
| 12 | beq $0,$0,0x0c |
| 13 | -100 |
| 14 | add $t0,$0,$0 |
| 15 | addi $t1,$0,1 |

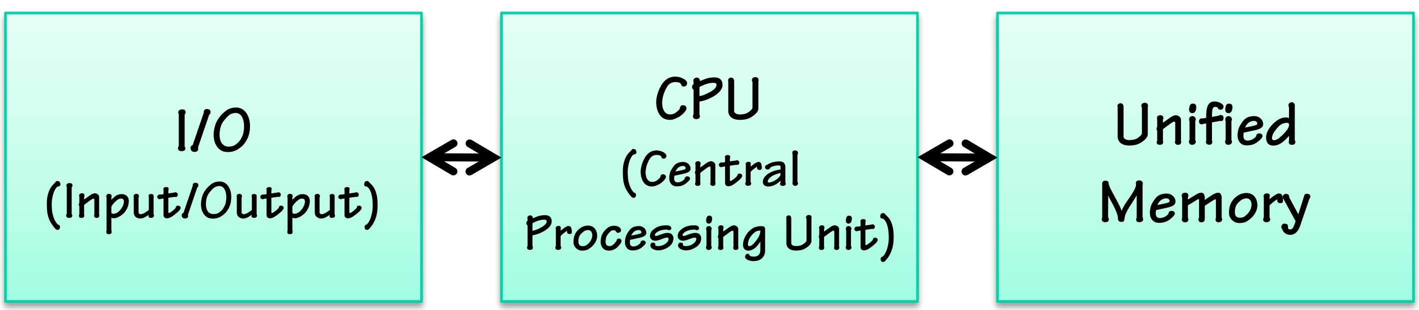
# A Bit of History

- There is a common debate over whether "data" and "instructions" should be mixed. Leads to two common flavors of computer architectures

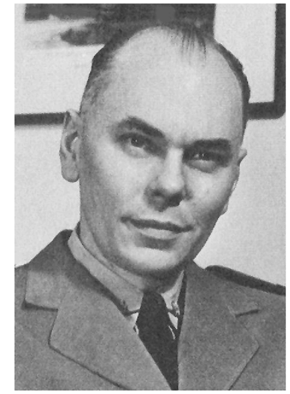
"Harvard" Architecture




"Von Neumann" Architecture



# A Bit of History



Howard Aiken:
Architect of the
Harvard Mark 1

- Harvard Architecture
  - Instructions and data do not interact, that can be different "word sizes" and exist in different "address spaces"
  - Advantages:
    - No self-modifying code
    - Optimize word-lengths of instructions for control and data for applications
    - Higher Throughput (i.e. you can fetch data and instructions from their memories simultaneously)
  - Disadvantages:
    - The H/W designer decides the trade-off between how big of a program and how large are data
    - Hard to write "Native" programs that generate new programs (i.e. assemblers, compliers, etc.)
    - Hard to write "Operating Systems" which are programs that at various points treat other programs as data (i.e. loading them from disk into memory, swapping out processes that are idle)

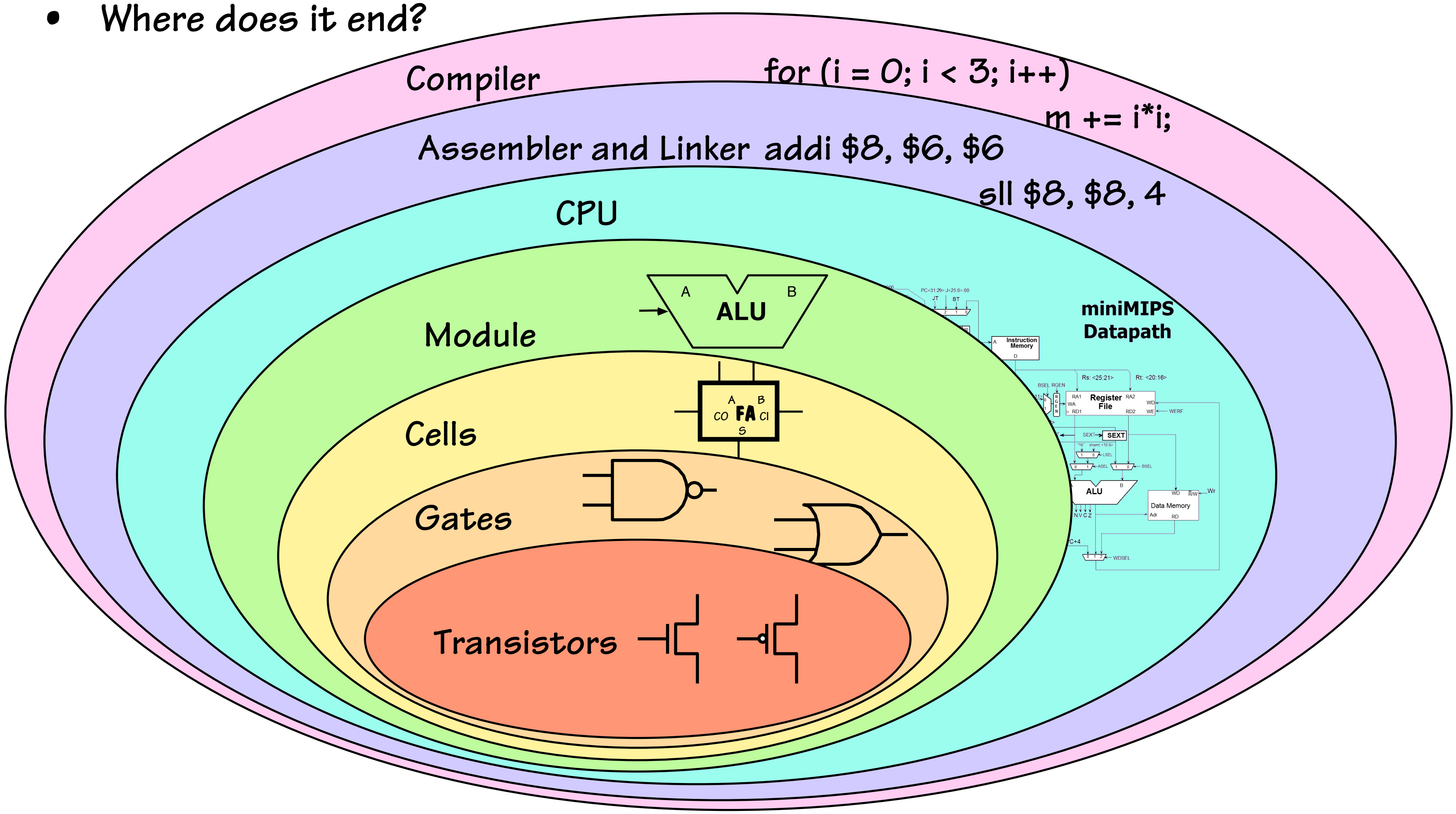
# A Bit of History

John Von Neumann: Proponent of unified memory architecture

- Von Neumann Architecture
  - Instructions and data are indistinguishable bits in a common memory that share a common "word size" and "address space"
  - Most common model used today, and what we assume in 411
  - Advantages:
    - S/W designer decides how to allocate memory between data and programs
    - Can write "Native" programs to create new programs (assemblers and compliers)
    - Programs and subroutines can be loaded, relocated, and modified by other programs (dangerous, but powerful)
  - Disadvantages:
    - Word size must suit both common data types and instructions
    - Slightly lower performance due to memory bottleneck (mediated in modern computers by the use of separate program and data caches)
    - We need to be very careful when treading on memory. Folks have taken advantage of the program-data unification to introduce viruses.

# Computer Systems

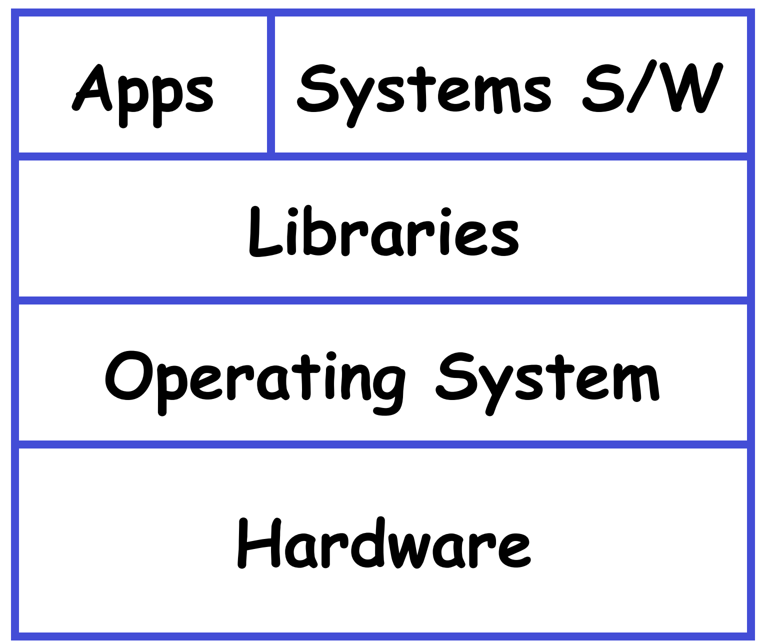
- What is a computer system?
- Where does it start?
- Where does it end?



# Computer System Layer Cake

- Applications
- Systems software
- Shared libraries
- Operating System
- Hardware – the bare metal

**Computers are digital Chameleons**



# Computers are Translators

- User-Interface (visual programming)
- High-Level Languages
  - Compilers
  - Interpreters
- Assembly Language
- Machine Language

```
int x, y;
y = (x-3)*(y+123456)
```

```
x:      .word 0
y:      .word 0
c:      .word 123456
    ...
    lw          $t0, x
    addi        $t0, $t0, -3
    lw          $t1, y
    lw          $t2, c
    add             $t1,
     $t1, $t2
    mul             $t0,
     $t0, $t1
    sw          $t0, y
```

# Computers are Translators

- User-Interface (visual programming)
- High-Level Languages
  - Compilers
  - Interpreters
- Assembly Language
- Machine Language

```
x:      .word 0
y:      .word 0
c:      .word 123456
    ...
    lw          $t0, x
    addi        $t0, $t0, -3
    lw          $t1, y
    lw          $t2, c
    add             $t1,
     $t1, $t2
    mul             $t0,
     $t0, $t1
    sw          $t0, y
```

```
0x04030201
0x08070605
0x00000001
0x00000002
0x00000003
0x00000004
0x706d6f43
```

# Why So Many Languages?

- Application Specific
  - Historically: COBOL vs. Fortran
  - Today: C# vs. Java
    Python vs. Matlab
- Code Maintainability
  - High-level specifications are easier to understand and modify
- Code Reuse
- Code Portability
- Virtual Machines

# Next Time

- A complete Instruction Set
- Assembly Language
- Machine Language