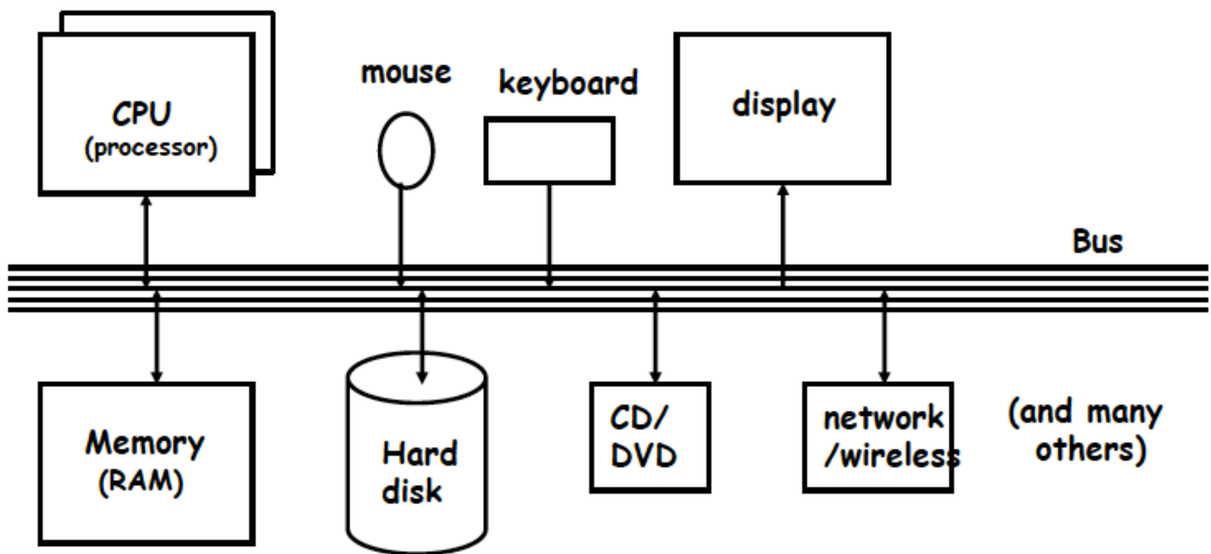
What's in a computer?

- **logical or functional organization: "architecture"**
  - what the pieces are, what they do, how they work
  - how they are connected, how they work together
  - what their properties are

- **physical structure**
  - what they look like, how they are made

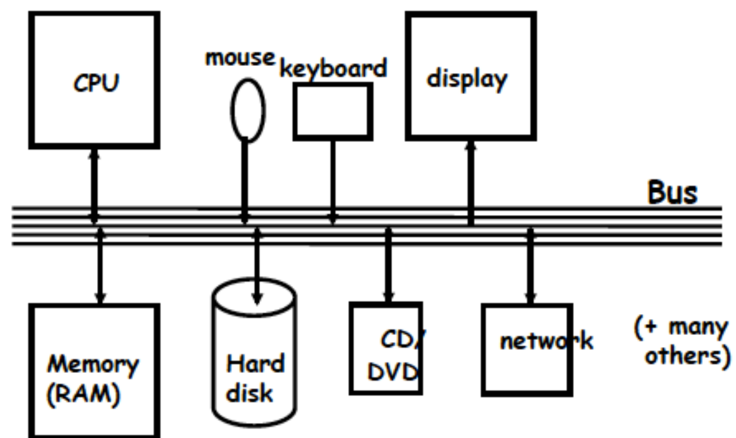
- **major pieces**
  - processor ("central processing unit" or CPU)
    does the work, controls the rest
  - memory (RAM = random access memory)
    stores instructions and data while computer is running
  - disks ("secondary storage")
    stores everything even when computer is turned off
  - other devices ("peripherals")

Block diagram of typical laptop/desktop



Block diagram of computer

- **CPU can perform a small set of basic operations**
  - arithmetic: add, subtract, multiply, divide, …
  - memory access: fetch data from memory, store results back in memory
  - decision making: compare numbers, letters, …, and decide what to do next according to result
  - control the rest of the machine



- **operates by performing sequences of very simple operations *very* fast**

- **instructions to be performed are stored in the same memory as the data is**
  - instructions are encoded as numbers: e,g., Add = 1, Subtract = 2, …

- **CPU is a general-purpose device: putting different instructions into the memory makes it do a different task**
  - this is what happens when you run different programs

Fundamental ideas

- **a computer is a general-purpose machine**
  - executes very simple instructions very quickly
  - controls its own operation according to computed results
- **"von Neumann architecture"**
  - change what it does by putting new instructions in memory
  - instructions and data stored in the same memory
  - indistinguishable except by context

    attributed to von Neumann (1946)

    (and Charles Babbage, in the Analytical Engine (1830's))
  - logical structure largely unchanged for 60+ years
  - physical structures changing very rapdily
- **Turing machines**
  - all computers have exactly the same computational power:

    they can compute exactly the same things; differ only in performance
  - one computer can simulate another computer

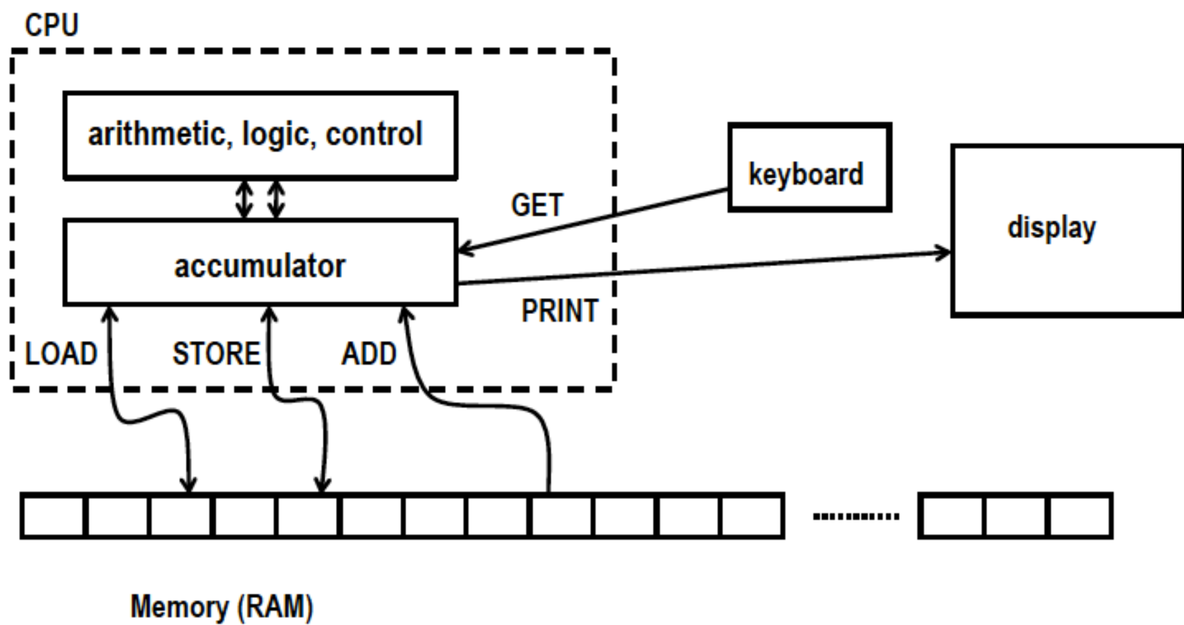
    a program can simulate a computer

A simple "toy" computer (a "paper" design)

- **repertoire ("instruction set"): a handful of instructions, including**
  - **GET** a number from keyboard and put it into the accumulator
  - **PRINT** number that's in the accumulator (accumulator contents don't change)
  - **STORE** the number that's in the accumulator into a specific RAM location (accumulator doesn't change)
  - **LOAD** the number from a particular RAM location into the accumulator (original RAM contents don't change)
  - **ADD** the number from a particular RAM location to the accumulator value, put the result back in the accumulator (original RAM contents don't change)
- **each RAM location holds one number or one instruction**
- **CPU has one "accumulator" for arithmetic and input & output**
  - a place to store one value temporarily
- **execution: CPU operates by a simple cycle**
  - FETCH: get the next instruction from RAM
  - DECODE: figure out what it does
  - EXECUTE: do the operation
  - go back to FETCH
- **programming: writing instructions to put into RAM and execute**

http://kernighan.com/toysim.html

Toy computer block diagram (non-artist's conception)



Toy Simulator

http://kernighan.com/toysim.html

(You must have Javascript enabled.) Type your program in the left window. **Labels must start in the first column and operators like GET or ADD must start anywhere but the first column (i.e., there must be a space or tab before them).** The simulator does not distinguish upper case from lower case, but is otherwise not robust, so be sure to spell instructions correctly and format code carefully.

Push RUN to run your program. A dialog box will appear when a GET is executed, and output from PRINT will appear in the right window. The simulator will stop if you Cancel a GET or don't enter anything.

```
    get
```

Accumulator:

RUN

Syntax reminder

```
get        get a number from keyboard into accumulator
print      print contents of accumulator
load Val   load accumulator with Val (Val unchanged)
store M    store contents of accumulator into memory location M (accumulator unchanged)
add Val    add Val to contents of accumulator (Val unchanged)
sub Val    subtract Val from contents of accumulator (Val unchanged)
goto L     go to instruction labeled L
ifpos L    go to instruction labeled L if accumulator is >= zero
ifzero L   go to instruction labeled L if accumulator is zero
stop       stop running
Num        initialize this memory location to numeric value Num (once, before program runs)
```

If `Val` is a name like `Sum`, it refers to a labeled memory location. If `Val` is a number like 17, that value is used directly. So `add Sum` means to add the contents of the memory location named `Sum` to the accumulator value (leaving `Sum`'s contents unchanged), while `add 1` means to add 1 to the accumulator value.

Syntax reminder

http://kernighan.com/toysim.html

```
get        get a number from keyboard into accumulator
print      print contents of accumulator
load Val   load accumulator with Val (Val unchanged)
store M    store contents of accumulator into memory location M (accumulator unchanged)
add Val    add Val to contents of accumulator (Val unchanged)
sub Val    subtract Val from contents of accumulator (Val unchanged)
goto L     go to instruction labeled L
ifpos L    go to instruction labeled L if accumulator is >= zero
ifzero L   go to instruction labeled L if accumulator is zero
stop       stop running
Num        initialize this memory location to numeric value Num (once, before program runs)
```

If `Val` is a name like `Sum`, it refers to a labeled memory location. If `Val` is a number like 17, that value is used directly. So `add Sum` means to add the contents of the memory location named `Sum` to the accumulator value (leaving `Sum`'s contents unchanged), while `add 1` means to add 1 to the accumulator value.

http://kernighan.com/toysim.html

```
start    get
         print
         store numb
         add 10
         add numb
         print
         get
         ifzero down
         goto start
down  stop
numb   0
```