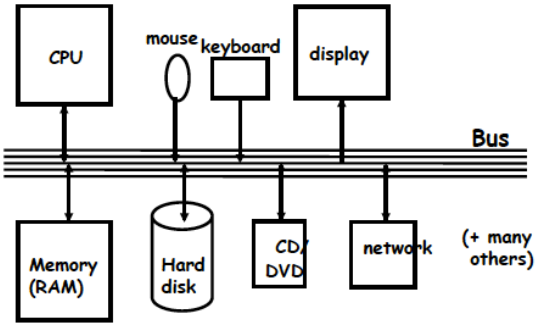
major pieces

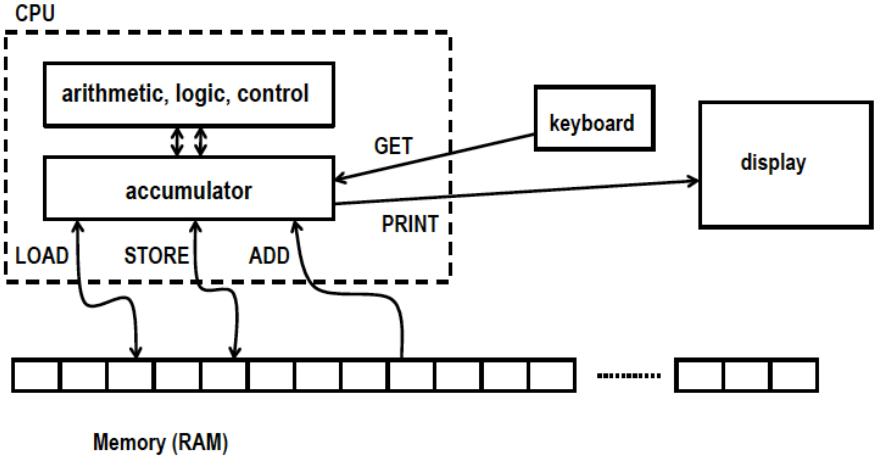
- processor ("central processing unit" or CPU)
  does the work, controls the rest
- memory (RAM = random access memory)
  stores instructions and data while computer is runni
- disks ("secondary storage")
  stores everything even when computer is turned off
- other devices ("peripherals")

Block diagram

- **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**



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



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

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.