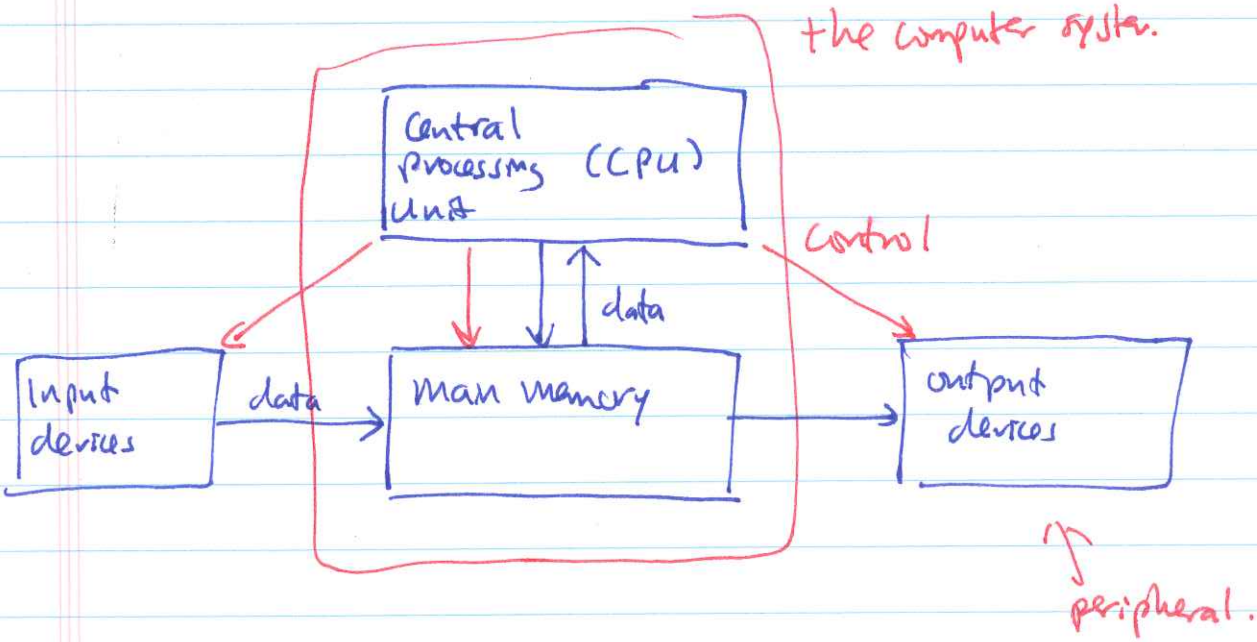
Introduction

Von Neumann

Abstract view of a computer system

schematically, a computer consists of 4 main components, interconnected as follows:



- CPU = central processing unit.
  Computation & process instructions in program.

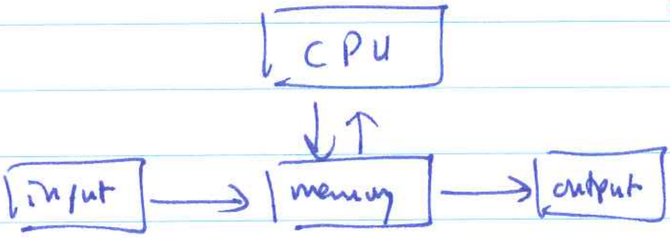
- Memory = store instructions of program.
  store input & output data.
  Store variables used by program.

- Input : allows user to enter data into the computer system

- Output : allows the computer to output data to user.

I/O are used for communication between Computer System & users.

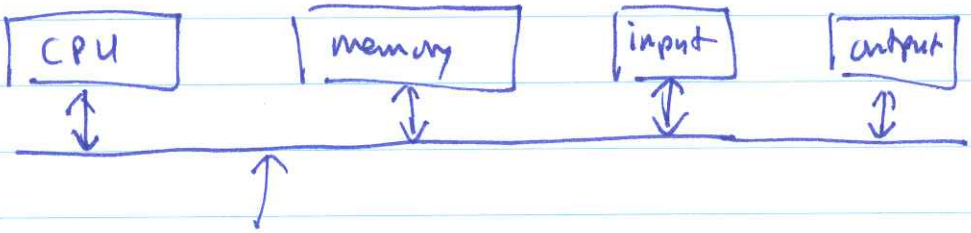
Note :- Although the diagram is given as:



Depicts logical communication channels.

In reality, there are many different ways to connect the components.

A popular way (cheap, found eg. in PC) is:

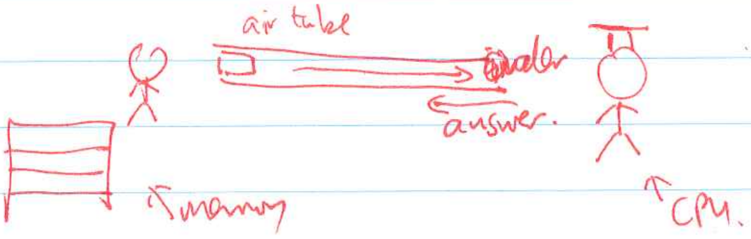


Common wires connecting all components.

(note: 1 line is infact many wires).

Compare :

CPU ↔ memory



Main memory :

- Purpose : store programs (instructions)
  store data.

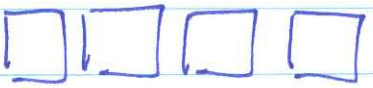
- Bit = Binary Digit

  = a memory element that can contain one of the values $\{\emptyset, 1\}$.

A bit is like a switch : it can be on or off.

A bit is the smallest unit of memory available. It's to small to be of practical use.

- Bits are grouped together to form larger memory elements, eg. :



4 bits.
(4 switches).

Each bit works independantly from the others. Each bit can take on the values $\{\emptyset, 1\}$.

4 bits can take on 16 diff. values:

0000
0001
0010
⋮ etc.

- In general, n bits together can take on 2^n diff. values.
- 8 bits are grouped together to form a "byte".

The number of diff. values that can be stored in a byte of memory is:

$$2^8 = 256.$$

(Why 8?)

A byte is sufficiently large (chunk of memory) to represent all characters in alphabet.

26 capital letters
26 small letters
10 digits.

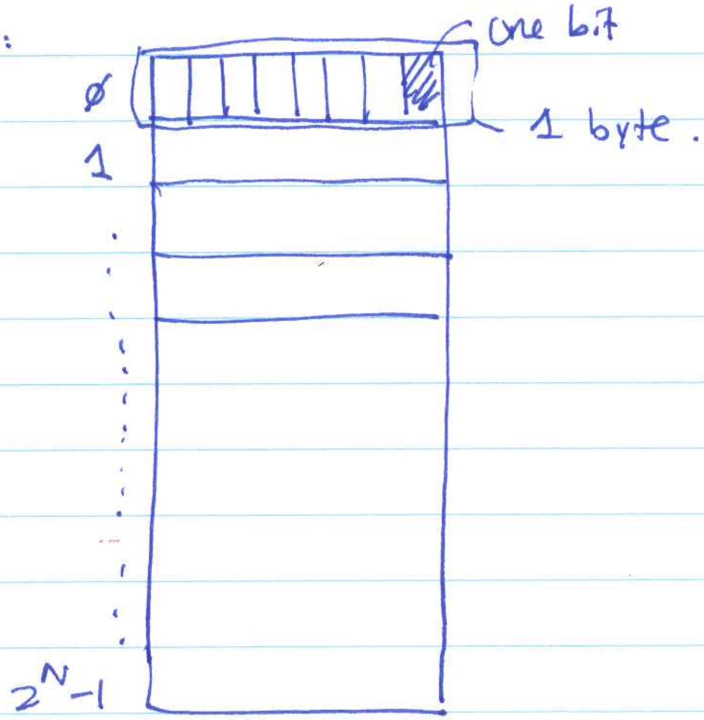
Each symbol (character) is represanted by a unique bit pattern.

eg: the letter

'A'	=	0100 0001
'B'	=	0100 0010
'C'	=	0100 0011
'a'	=	0110.0001
'b'	=	0110.0010
'c'	=	0110.0011

- The main memory is organized as follows:

main memory:



- Each byte is assigned with a unique "address".

  The address starts with ø up to a boundary. The total number of address is usually a power of 2, i.e., the number of diff. addr. = 2^N for some N.

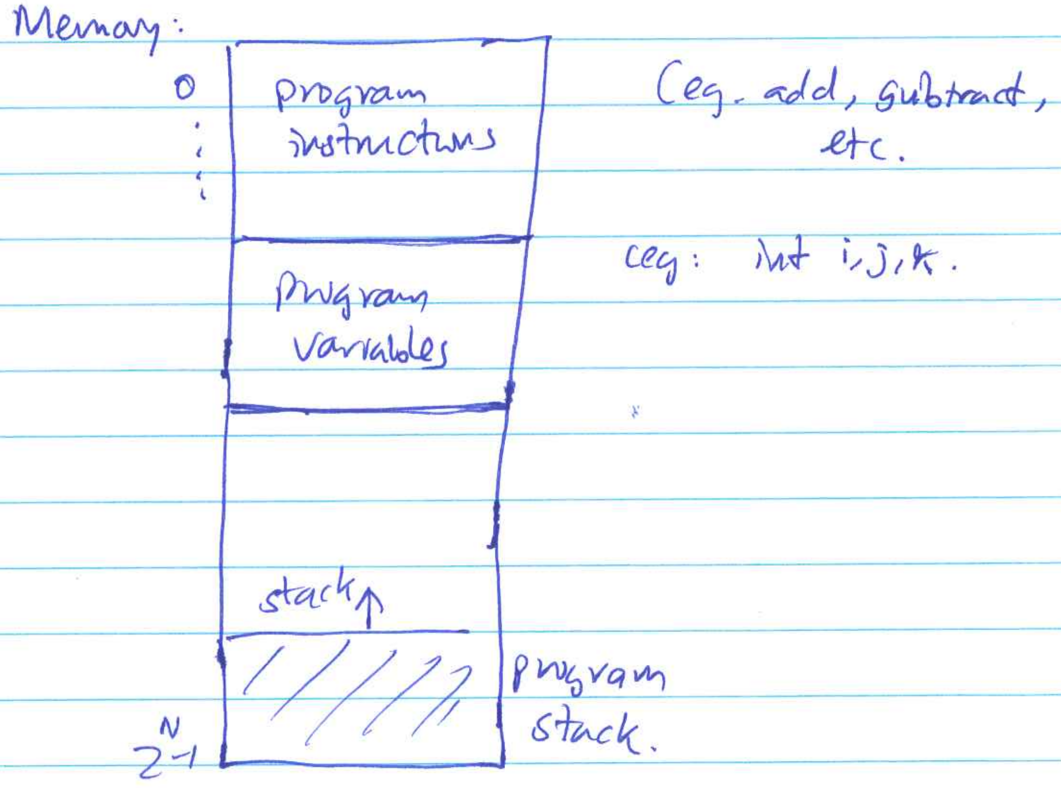
- The smallest memory unit that can be addressed (given an address) is usually one byte.

  - Such computers are called "byte addressable".
  (The smallest amount of data that can be retrieved / stored to memory is one byte).

Memory used to store: instructions
+
Data

Program organization in memory:

When a C or Pascal or Assembler (any program) is executed by the computer, the entire program is placed inside the memory, typically as follows:



- The program stack will be explained later in the course. It contains local variables & parameters for functions. It also enables function to return to its caller.

- The stack can grow & shrink. Prog. instr. & variables do not.

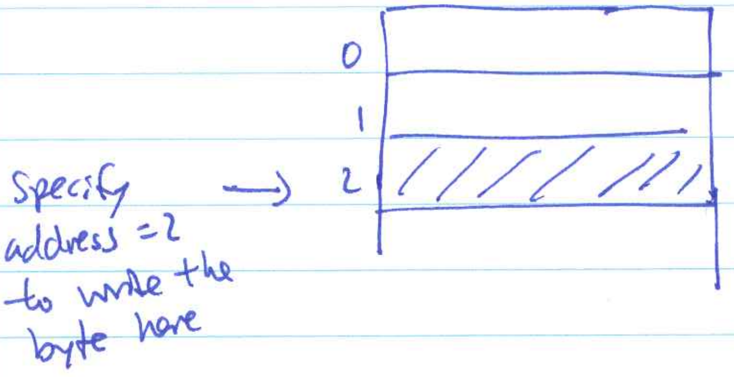
Physical Connections between CPU & Memory

- Jargon:

read memory = retrieve the data stored in the memory.

write memory = store new data into memory. The data stored previously in memory is erased (like a cassette tape).

- The CPU can read and write a certain memory location by specifying the corresponding address.



- To convey the address, the computer uses a set of wires, called address lines.

These address lines run from the CPU to the memory.

- Also, to convey the data that is stored in the memory to the CPU and the new data that the CPU wants to put in the memory, ~~a stuff~~ another set of wires are provided.

  This set of wires is called "data lines".

- Finally, there is a 3rd set of wires between the CPU & the memory that is used to convey control information.

  Example of control information:

  example control signals

  (1) read command
  (2) write command
  (3) ready status

  This set of wires is called "control lines".

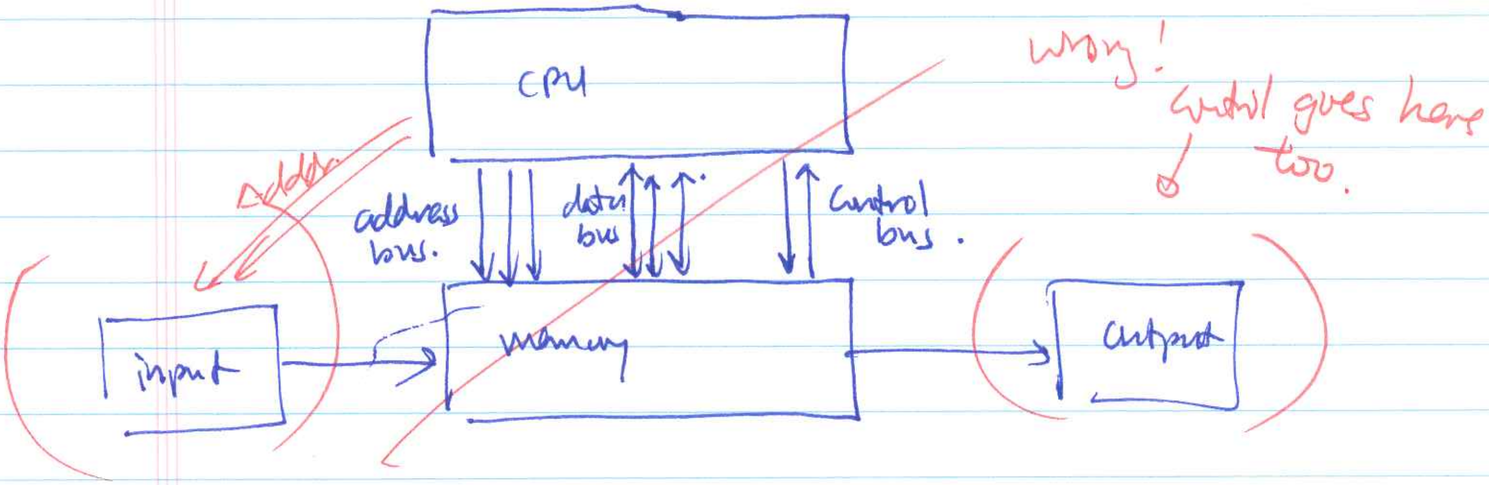
- A set of lines used for a similar purpose is called a bus.

  Hence:

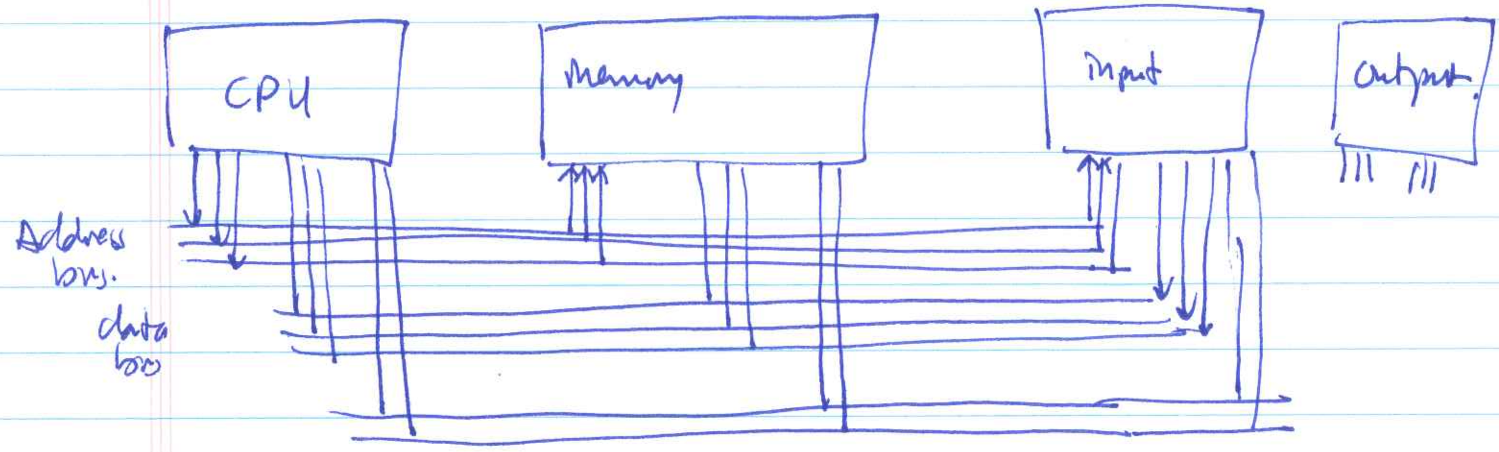
  address bus
  data bus
  control bus

  } together they are called the system bus.

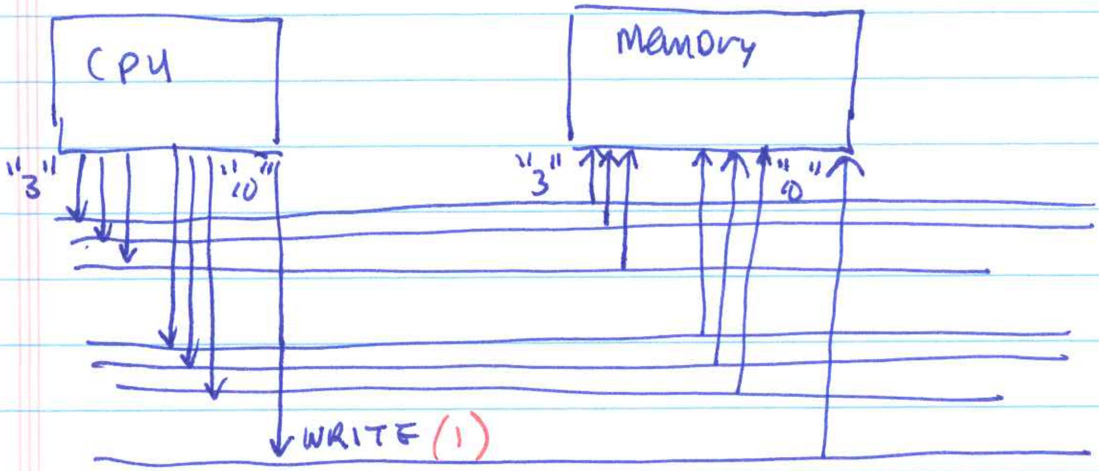
The actual connection between the CPU & memory is as follows:



Again, in reality, things can be connected in many diff. ways, in the IBM PC:



How the CPU writes the number 3 into ~~[illegible]~~ memory location 10 :



- We will later see :

  (1) How the CPU specifies the values 3 & 10

  (2) How the CPU specifies the operation (read or write).

- The CPU can request any item (data) stored any where in main memory and the main memory takes the SAME time to return the value.

  Such behavior is called "random access"
  → access time does NOT depend on the location where the data is stored.

- In contrast, sequential access devices have the property that access time depends on where the data is stored.

  eg. seq. access device : tape

  To access the first item on the tape, it's fast.
  To get to the last item, you need to fast forward the tape → takes longer time.

- Memory Size :

The memory size is given as the number of bytes.

Symbolic abreviations :

K = kilo
$= 2^{10} = 1024$.
$\approx 1,000$.

M = mega
$= 2^{10} \times 2^{10} = 2^{20} =$
$\approx 1,000,000$

- Word = the most convenient unit that the CPU can work with.

Depends on how the CPU is designed.

M68000 : 1 word = 16 bits
= 2 bytes.

SPARC : 1 word = 32 bits
= 4 bytes.

Most modern computers use 1 word = 32 bits.

Words are used for computation. (represent integers).

Programming Languages

- Program = a sequence of instructions that (when executed) achieves a certain effect.

  eg. solve an equation. } programs.
  compute balance,
  etc.

- A program can be written in : (expressed).

  (1) high level programming language (like C, Pascal)

  (2) Assembler code
  } very closely related.
  (3) Machine code.

- High level programming language :
  (0) Written using English-like statements & math. expressions.
  (1) Non~~e~~ machine specific.
  (2) Written ~~using~~ to be read easily by humans.
  (3) Programming language is general purpose to a certain extend.
  Some languge is better suited for certain problems.

~~Machine code~~

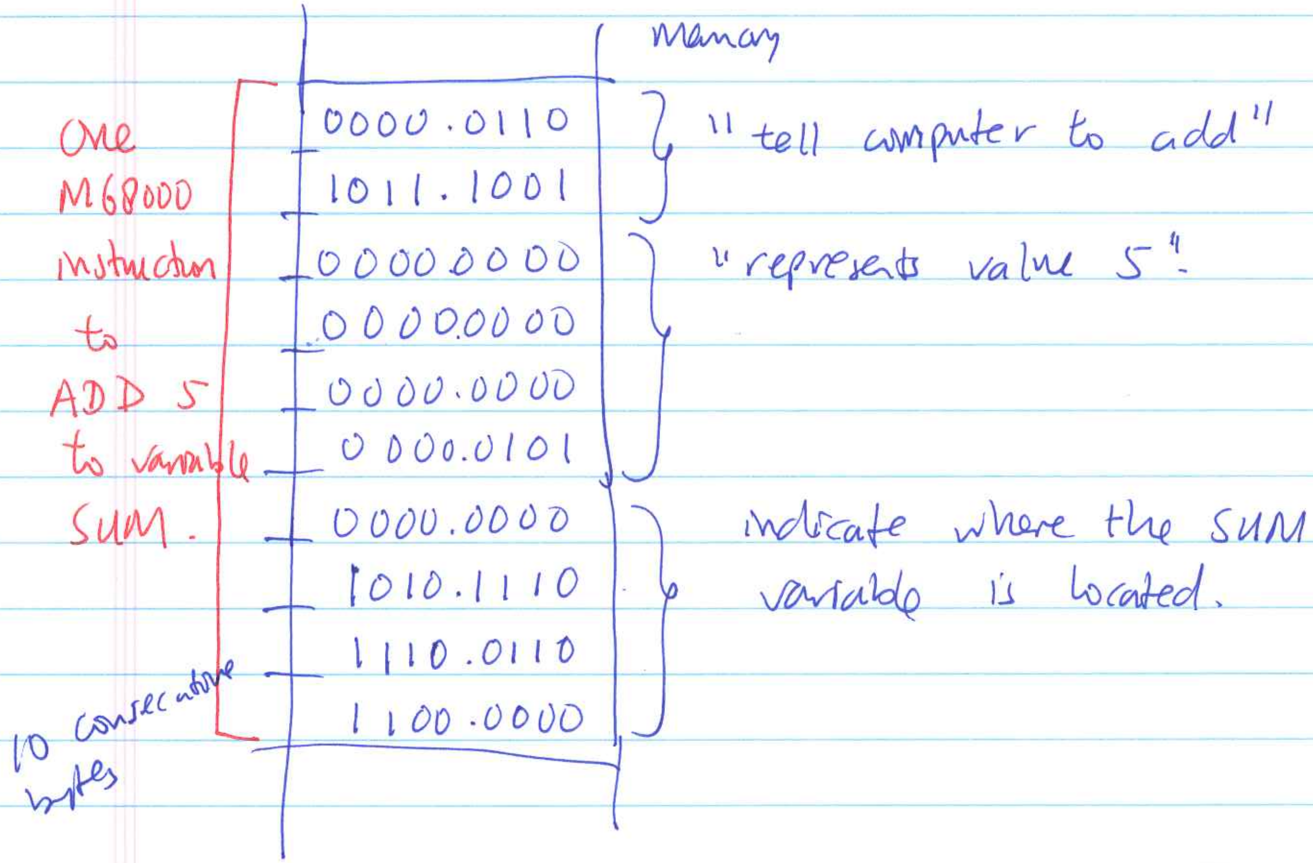
Example: sum = sum + 5 ;

adds 5 to the variable sum.

- Machine code = a sequence of bits that encodes a machine (computer) instruction.

  - it instructs the computer to perform <u>one</u> operation.

example: M68000 machine instruction:



- Obviously machine code is not meant to be read by humans!

Machine code is specifically meant to be executed (read) by the computer.

- Some applications need to use machine code to write it.

An area in computer science that frequently use machine code is "systems programming".

Systems programming : discipline of designing & implementing programs (software) that let people use the computer system more easily.

- Writting systems programs using machine code (strings of 0's and 1's) is NOT a viable way
  - easy to make error
  - un readable.

Huron Ontario
Michigan
Erie
Superior.
(HOMES)

- Assembler code = a mnemonic code (symbolic names) that represent machine code.
  - easier readable for humans.
  - ~~but~~ still very close to machine code:

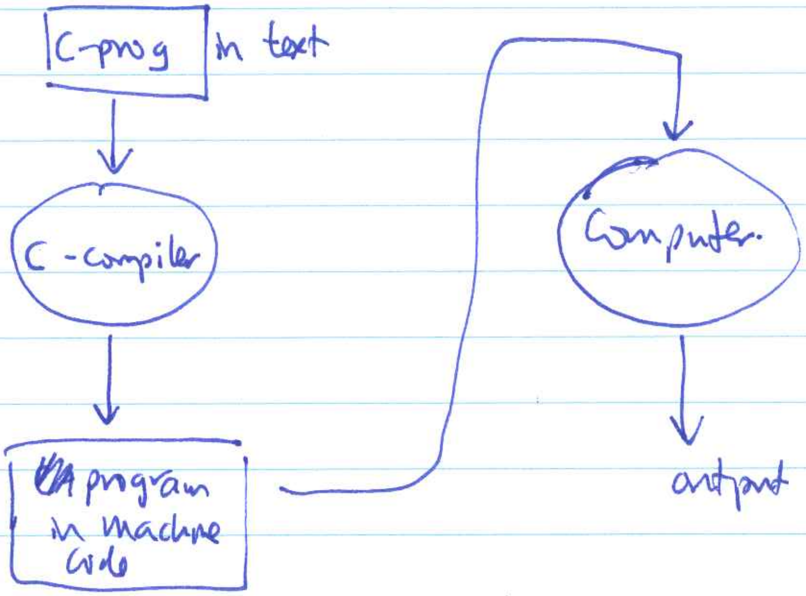
    there is a one-to-one correspondence between assembler code & machine code.

eg: ADD #5, SUM - add 5 to variable SUM.

From C-program to execution :

- A C-program needs to be translated to a program consisting of machine code before the computer can execute it. — the translator is C-compiler.

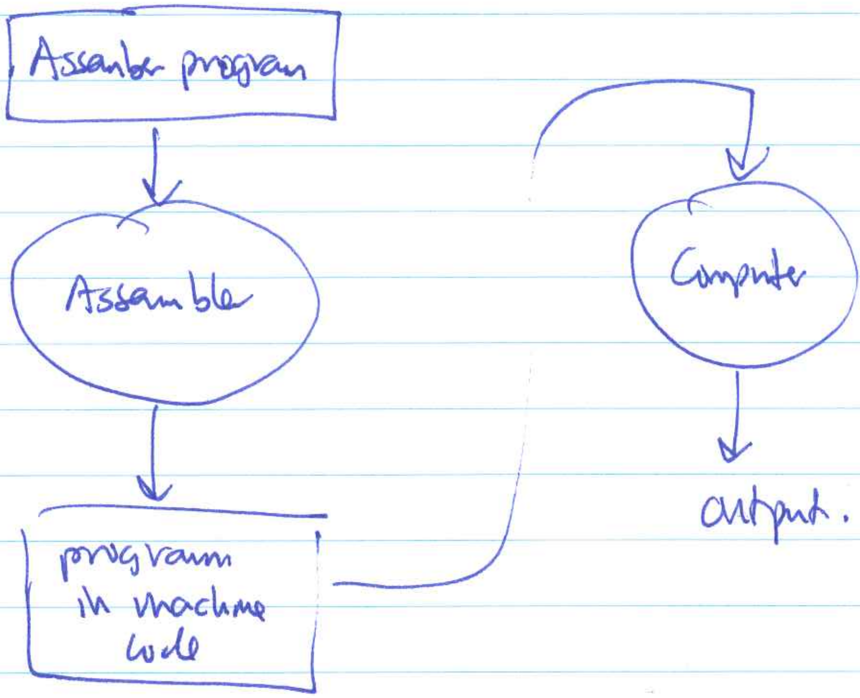
- Programming procedure :



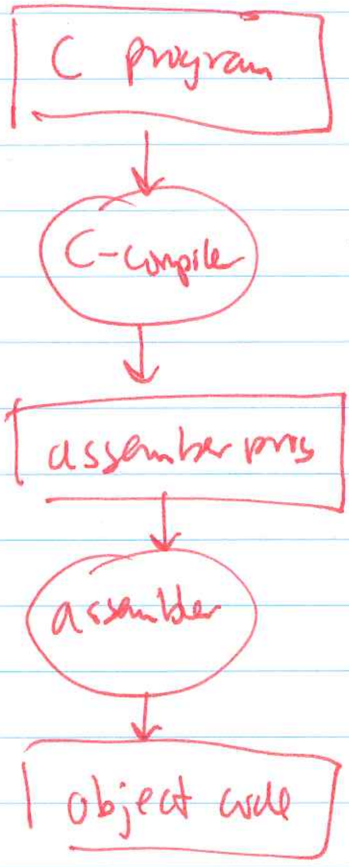
From assembler to execution.:

- An assembler program also needs translation to a program consisting of machine code before the computer can execute it — the translater is an Assembler.

Assembler is MUCH MORE SIMPLER than compiler.



More typically:



(can be executed by computer).