Introduction

A computer system is comprised of a collection of:

Processing elements - perform transformations on data items

- CPU
- GPU (graphics card)
- Disk controller

Storage elements – store data items for later use

- Main memory
- Cache memory
- Disk

Communication elements – transfer data items between processing and/or storage elements

- Internal Processor bus
- PCI Bus
- SCSI Bus
- Ethernet link

Programming a computer

We will consider a simple hypothetical “human powered” computer with a CPU (the human) and a main memory consisting of a wall of “post office” type boxes

Each box has a unique address or “box number”.

Addresses range from 0, 1, 2, ... , n-1 where *n* is the number of different boxes.

Each box can hold either:

An instruction to the human or
A real number

The human computer is limited his capacity to understand instructions but performs the instructions he can understand with 100% reliability.

The human computer instruction set

store value,box#
store 130,101 *! Stores the value 130 in box # 101*

add box#,box#,box#
add 101,102,110 *!Adds the values stored in box 101 and 102 and puts the answer in box 110*
This does NOT mean that the value 203 goes into box 110 we can't know what will go into box 110 unless we know the value presently stored in box 101 and box 102

sub box#,box#,box#
sub 100,200,300 *!Subtracts the value stored in box 200 from the value stored*

halt *!Program is complete, stop*

When the human computer is told to *start* he fetches his first instruction from *box 0* and then proceeds sequentially through box 1, box 2, ... until he encounters a *halt* instruction or a *jumpc* instruction

jumpc box#,op,box#,box#
jumpc 100,lt,101,10 *!if the value stored in box 100 is less than the value stored in box 100 then fetch your next instruction from box 10. If the instruction in box 10 is neither halt nor jumpc the instruction fetched after box 10 will be in box 11.*

Other conditions include

ne – not equal
eq – equal
le – less than or equal to
gt – greater than
ge – greater than or equal to

Adding the 1st 100 integers

Problem: Add 1+2+3+....+100 and leave the sum in box 100.

Box		
0	*store 100,200*	*!Store the upper limit in box 200*
1	*store 0,100*	*!Initialize the sum to 0*
2	*store 1,101*	*!Initialize the value to be added*
3	*store 1,102*	*!We need a value to increment the value to be added*
4	*add 101,100,100*	*!add contents of box 101 to box 100 leaving sum in 100*
5	*add 102,101,101*	*!increment the value to be added to box 100 by 1*
6	*jumpc 101,le,200,4*	*!as long as the value in box 101 is less than or equal to the 100 in box !200 continue to add*
7	*halt*	

Exercise for next time: The human computer doesn't have a multiply instruction. Write a program to multiply the value in box 100 by the value in box 101 leaving the result in box 102. (You may assume both values are integers)

This hypothetical program is written in a symbolic "machine language" sometimes called Assembly Language. It turns out to be the case that any result that may be computed on *any* computer may be computed (though much more slowly!) by the human computer. Thus, writing other sample programs for the human computer is a useful exercise as it provides useful practice in the art of specifying an *algorithm* precisely and correctly.

The human computer also illustrates and *extremely important concept: the distinction between the address of a memory cell (100) and the value it contains (1 + 2 + ... + 100)*

The human computer and the C programming language:

C is sometimes referred to has a “high level” assembly language because, of all modern programming languages, C code maps most directly and naturally to the machine level.

```
        limit = 100;
        sum = 0;
        addval = 1;
        incr = 1;
addit:
        sum = sum + addval;
        addval = addval + incr;
        if (addval <= limit) goto addit;
```

Note that in C, *a variable name* instead of *an actual address* is used to identify the *box* in which the variable resides, but the program itself is line by line equivalent.

A computer program that was capable of *translating* the C code above to Assembly Language is called a *compiler.*