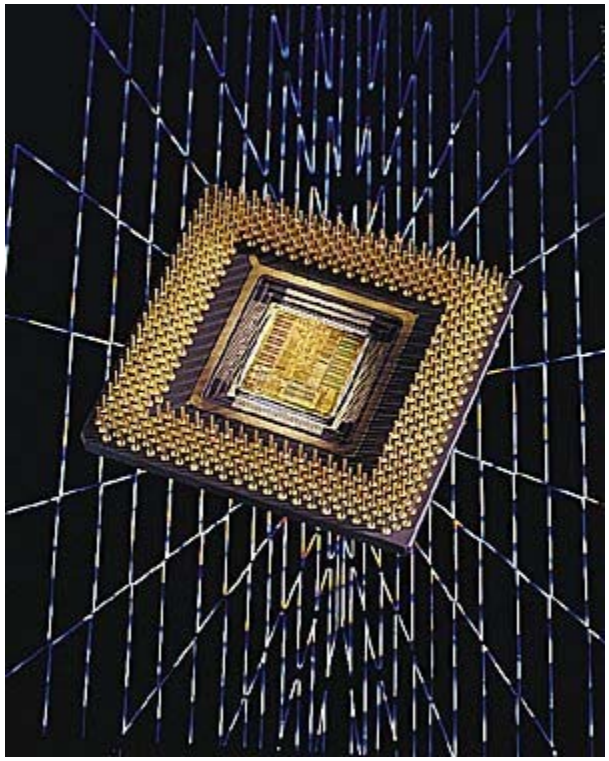
Compsci210 tutorial

Introduction to Assembly and LC-3 Simulator

Data representation part

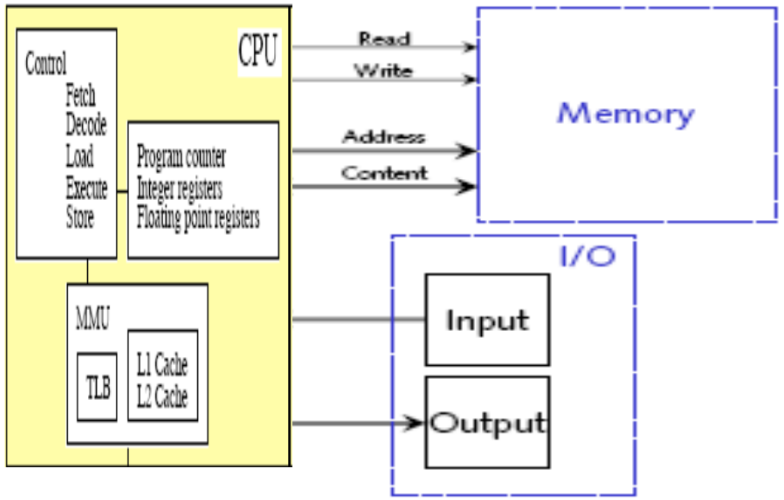
- Last tutorials we have learnt how to
  - Represent decimal numbers in binary forms (4 ways).
  - Add, subtract, multiply and divide numbers in binary form (2's complement).
  - Detect invalid overflow/underflow.
  - Understand bit wise operations like OR, AND, NOT…
  - Understand shift left (<<), shift right arithmetically (>>) and logically (>>>).
- We need to visualise what we have learnt:
  - Assembly and LC-3 Simulator

Central Processing Unit



- A **Central Processing Unit** (**CPU**), or sometimes just called **processor**, is a description of a class of logic machines that can execute computer programs.

CPU – memory - register



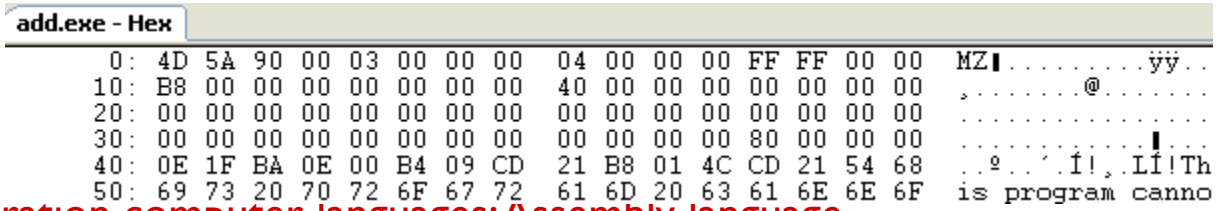
Registers	32 - 64	1 cycle
L1 cache	56 KB	2 cycles
L2 cache	512 KB - 2MB	6 - 10 cycles
External Memory	512MB - 1 GB	100 - 300 cycles
Disks	160 GB - 250 GB	10^7 cycles to seek

- CPU: 3.0 Ghz
- Bus: 667 Mhz
- Ram: 400 Mhz

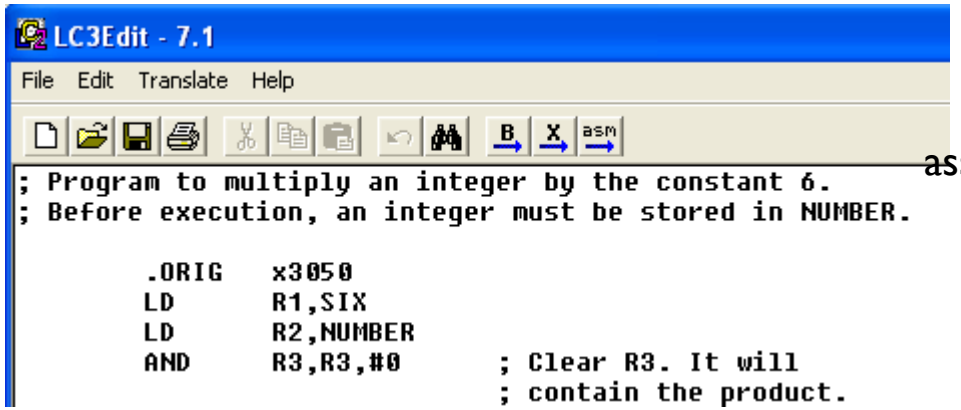
- Those are connected through bus (which is a subsystem that transfers data between computer components inside a computer)
- Connection speeds between them are different.
- Use registers to deal with calculation if possible!

A bit of history (1)

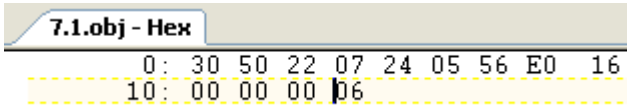
- First generation computer languages: Machine code.
  - Computer hardware is designed to understand and execute what is called "machine code" (instructions to tell the computer what to do).
  - A computer program: the bit pattern of the machine code to be loaded into the computer memory. It could be specified manually, using switches on the front panel for the computer



- Second generation computer languages: Assembly language.
  - Specifying a computer program as the bit pattern of the machine code was very time consuming, and error prone. A human readable/writable form of machine code was developed, namely "assembly language".
  - Another program, called an assembler, was developed to take a textual version of the machine code, and translate it into machine code.



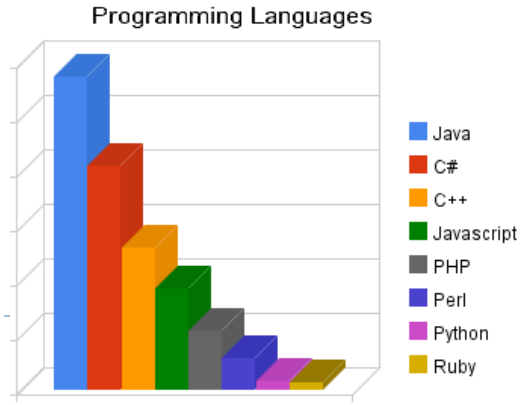
assembler



Translation from Assembly language to Machine code

A bit of history (2)

- Third generation computer languages: High level languages.
  - Assembly language is very low level, and depends on the computer architecture.
  - High level languages were developed, that were relatively machine independent, and more like the notation of mathematics. Fortran (~1955) and Basic (~1964): still had to build control statements out of one line if statements and goto statements, and there were no recursive functions.
  - Pascal (~1970) and C (~1972) were developed, with support for data structures.
  - 1971, Ken Thompson developed a FORTRAN compiler, instead ended up developed a compiler for a new high-level language he called B.
  - In 1972, Dennis Ritchie at Bell Lab, built on B to create a new language called C (next to B) which inherited Thompson's syntax. Most of the components of Unix were eventually rewritten in C. Because of its convenience and power, C went on to become the most popular programming language in the world over the next quarter century.
  - Modern languages (third generation languages), such as C++ (~1983) and Java (~1995), are object oriented languages (both are based on C). Third generation languages represent the main programming languages in use today.



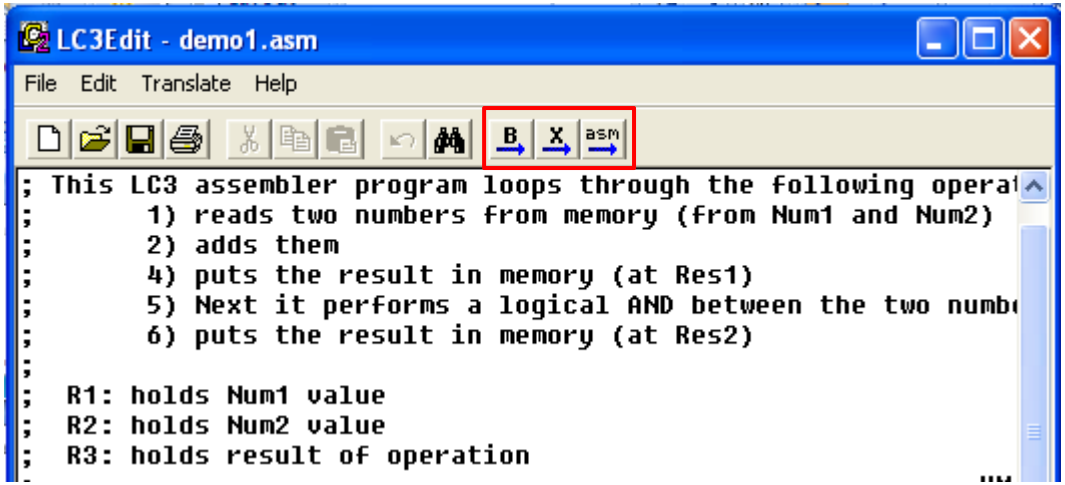
From: http://phpimpact.files.wordpress.com/2008/06/programming_languages.png

Install and run LC-3 Simulator

- Download links are provided in tutorial page
- After installation, you will see 2 exe programs (windows): LC3Edit.exe and Simulate.exe.

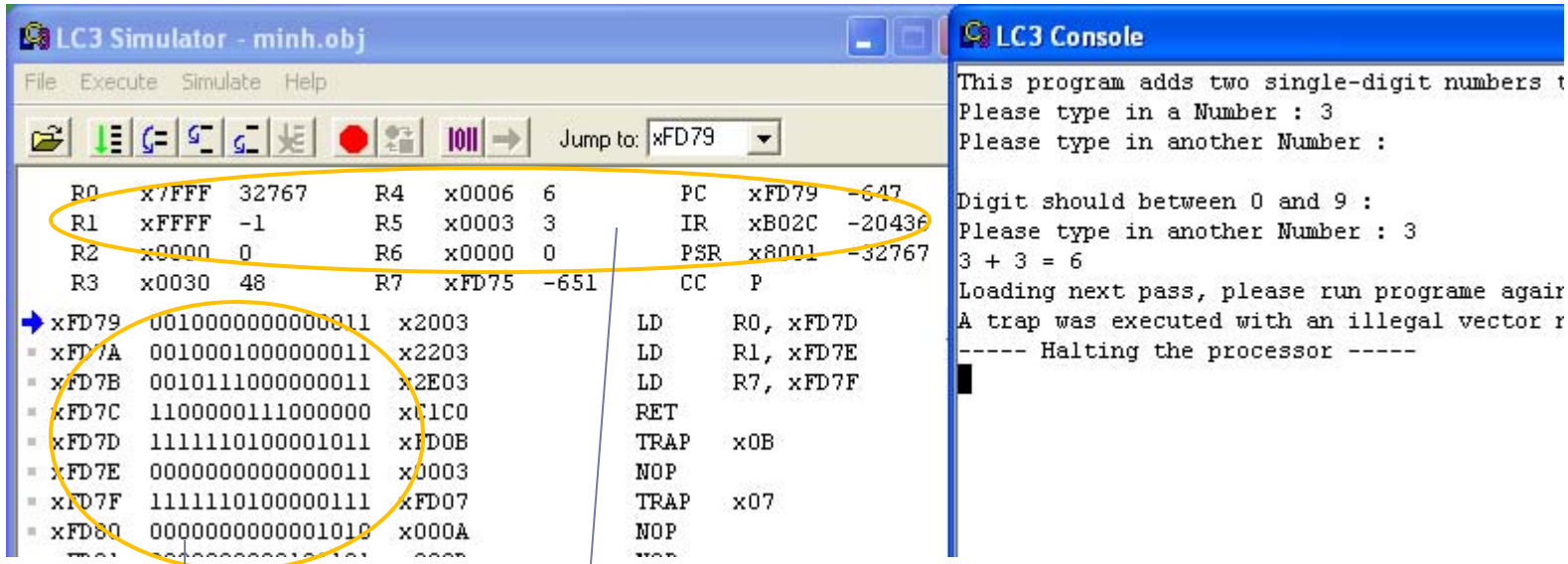
- LC3Edit is editor program (IDE).
- Simulate is LC3 simulator program (virtual computer which execute assembly code).

LC3Edit



- You can edit your program here using binary code, hexadecimal code and assembly code.
- After finish editing, you can export to .obj file which can be run by LC3 simulator.

LC3 simulate.exe



- Memory and values stored in memory
- Registers and values stored in register
- Include 2 frames: console (likes computer screen) and simulator (computer)

LC3 assembly programs

- Each program should be placed in it's own .asm file.
- The files should be entitled *.asm like example.asm...
- Each program should begin in memory at address x3000. This is accomplished via the .ORIG directive, which should be the first line in each file.
- The end of the program should consist of two lines: the penultimate line should contain the HALT instruction, and the last line in the file should contain the .END directive to inform the assembler that this is the end of the program.
- So… all of assembly files should be of the following form:
  - .ORIG x3000
  - ...
  - *your code goes here*
  - ...
  - HALT
  - .END
- See example programs.

Run example: AND.asm

```
        .ORIG x3000
;;; Test AND instructions
        ADD  R1,R1,#5
        ADD  R2,R2,#-2
        ADD  R3,R2,R1
        AND  R4,R3,#-1
        AND  R5,R1,R4
        ADD  R6,R6,#-1
        HALT
        .END
;;; Detail will be talked in tutorial
```

Run example: NOT.asm

```
        .ORIG x3000
;;;Test NOT instructions
        AND   R0,R0,#0
        NOT   R0,R0
        AND   R1,R1,#0
        ADD   R1,R1,#1
        NOT   R1,R1
        NOT   R1,R1
        ADD   R0,R0,R1
        HALT
        .END
;;; Detail will be talked in tutorial
```

Limited number of instruction sets in LC 3

- The LC-3 instruction set implements fifteen types of instructions, with a sixteenth opcode reserved for later use.
- Arithmetic instructions available include addition, bitwise AND, and bitwise NOT, with the first two of these able to use both registers and sign-extended immediate values as operands.
- The LC-3 can also implement any bitwise logical function, owing to the fact that NOT and AND together are logically complete.
- So **A OR B = NOT[(NOT A) AND (NOT B)]**
- Then AND, OR, NOT can be used to implement XOR

Exercise 1

- Interpret the instruction
  0x5fe0 = 0101 1111 1110 0000

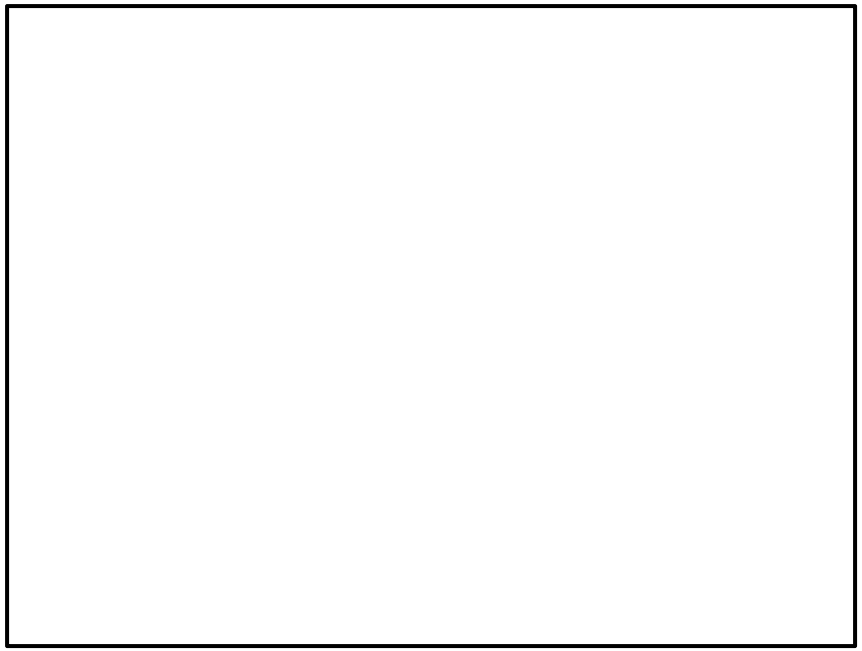
1. What is the opcode? What instruction is it? E.g. add, and, not?

2. Describe what the instruction does?

Exercise 2

- Use LC3 Assembly Instruction set table to convert the following code to Binary ISA codes:

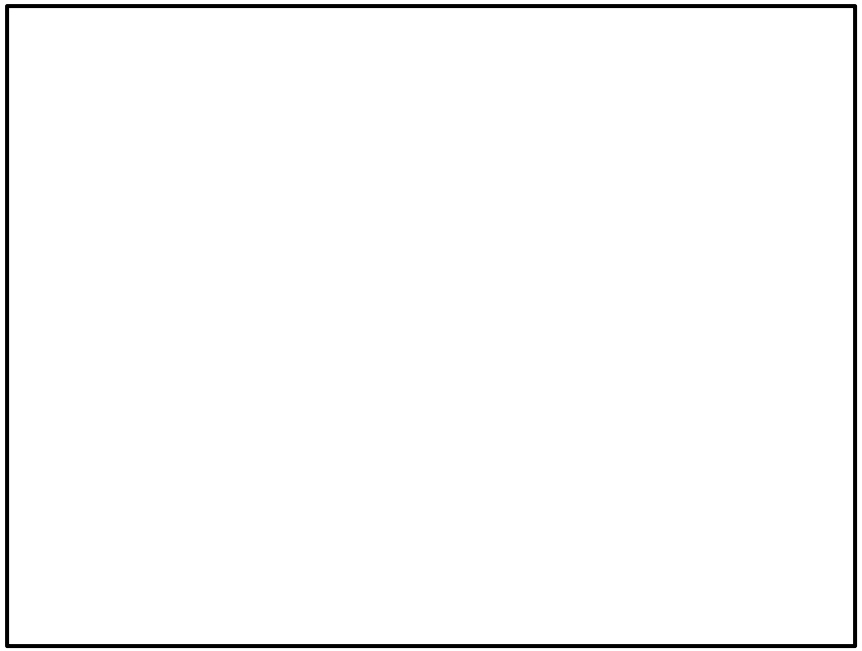
- ADD R1,R1,#5
- ADD R2,R2,#-2
- ADD R3,R2,R1
- AND R4,R3,#-1
- AND R5,R1,R4
- ADD R6,R6,#-1



Exercise 3

- Use LC3 Assembly Instruction set table to convert the following code to Binary ISA codes:

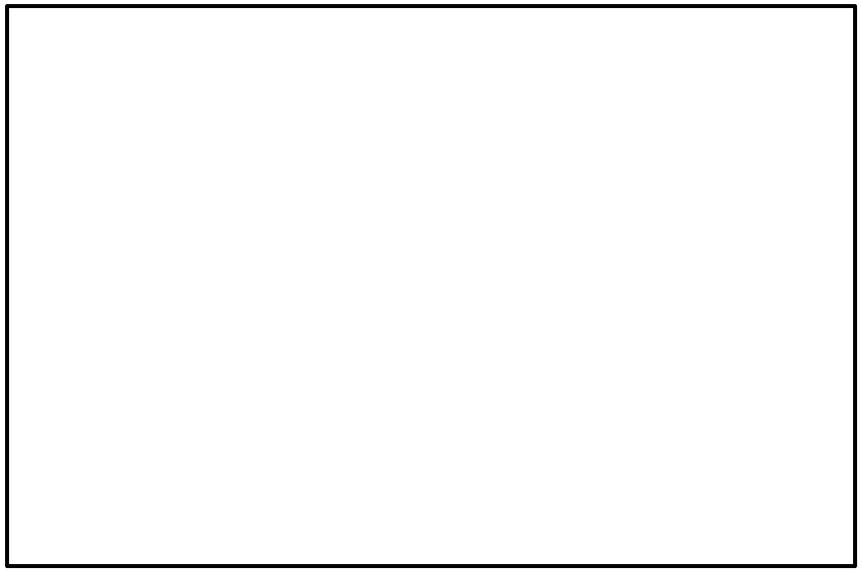
- AND R0,R0,#0
- NOT R0,R0
- AND R1,R1,#0
- ADD R1,R1,#1
- NOT R1,R1
- NOT R1,R1
- ADD R0,R0,R1



Exercise 4

- Use LC3 Assembly Instruction set table to convert the following code to Binary ISA codes:

```
LD      R2, Num1
LD      R3, Num2
ADD     R4, R2, R3
HALT
Num1    .FILL   5
Num2    .FILL   6
```



- What do the codes do?