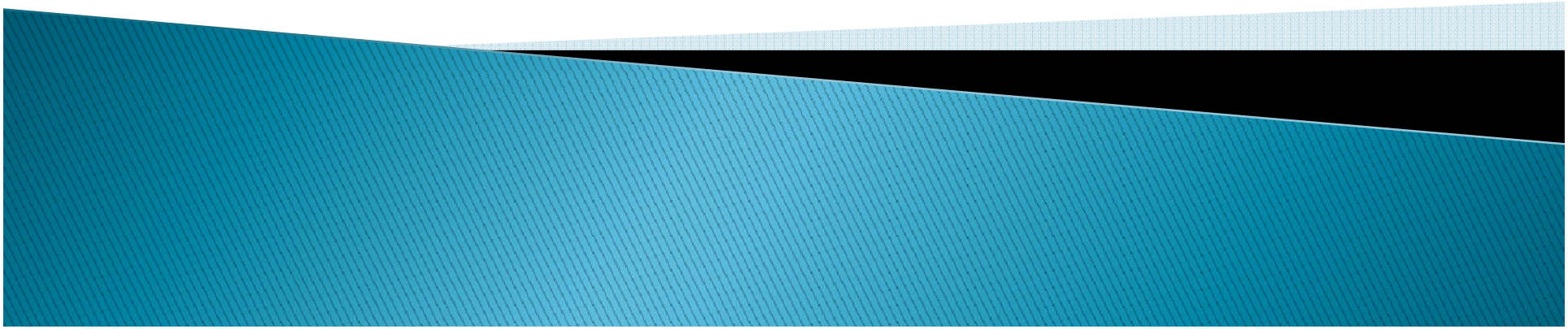
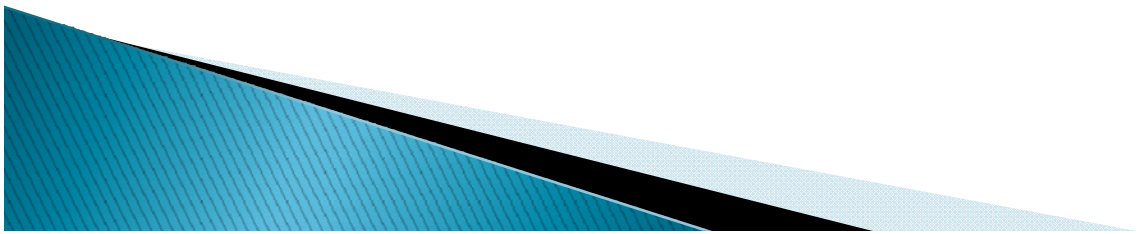
Computer Science 210 tutorial 3

Introduction to Assembly and LC-3 Simulator

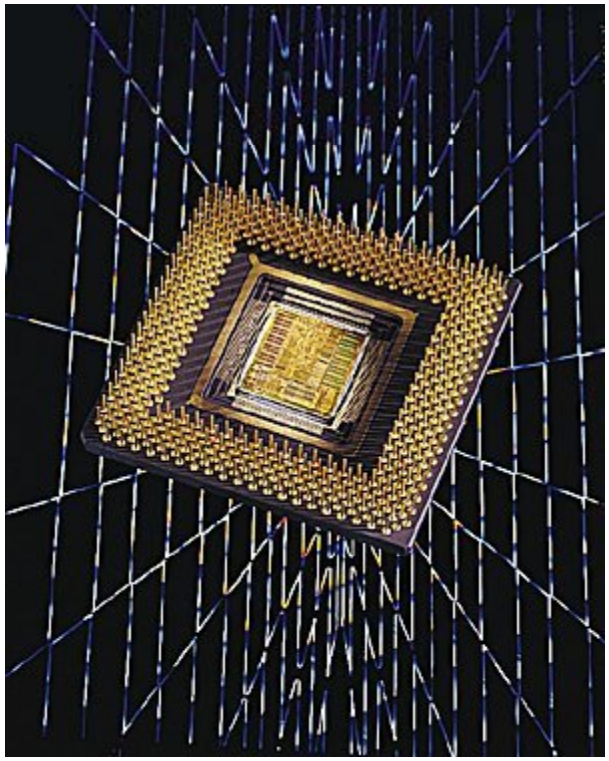


Data representation part

- 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

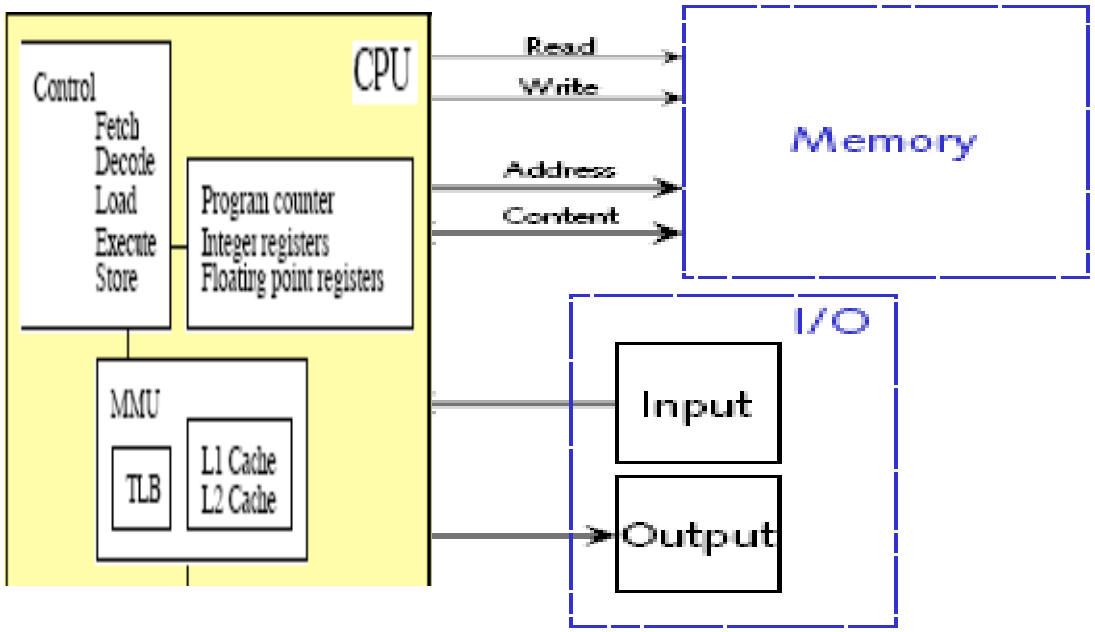


Central Processing Unit



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

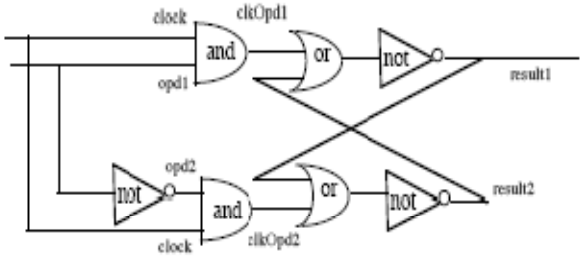
Inside Central Processing Unit (CPU)



- Internal memory inside CPU called registers, caches

Language levels

- Lowest



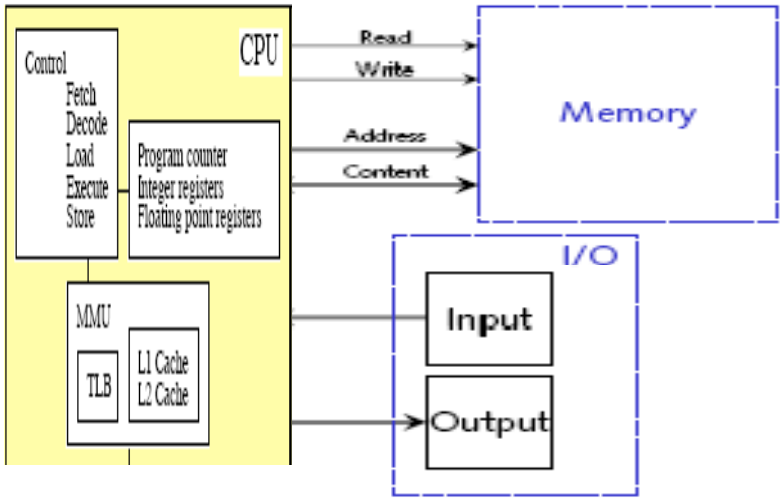
- 00111010101001
- Machine code

- Low
  - **ldiq $a0, CALLSYS_GETCHAR;**
  - **call_pal CALL_PAL_CALLSYS;**
- Assembly code

- Medium
  - C code

- High
  - C++/Java/.NET
  - …

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

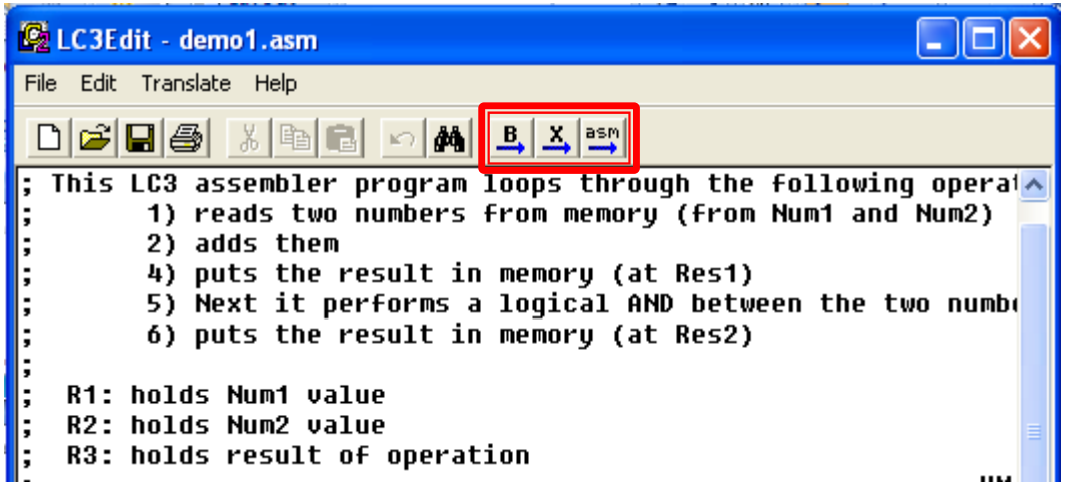
- Those are connected through bus (which is a subsystem that transfers data between computer components inside a computer)
- Connection speeds between them are different.
- Use registers to deal with calculation if possible!

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.

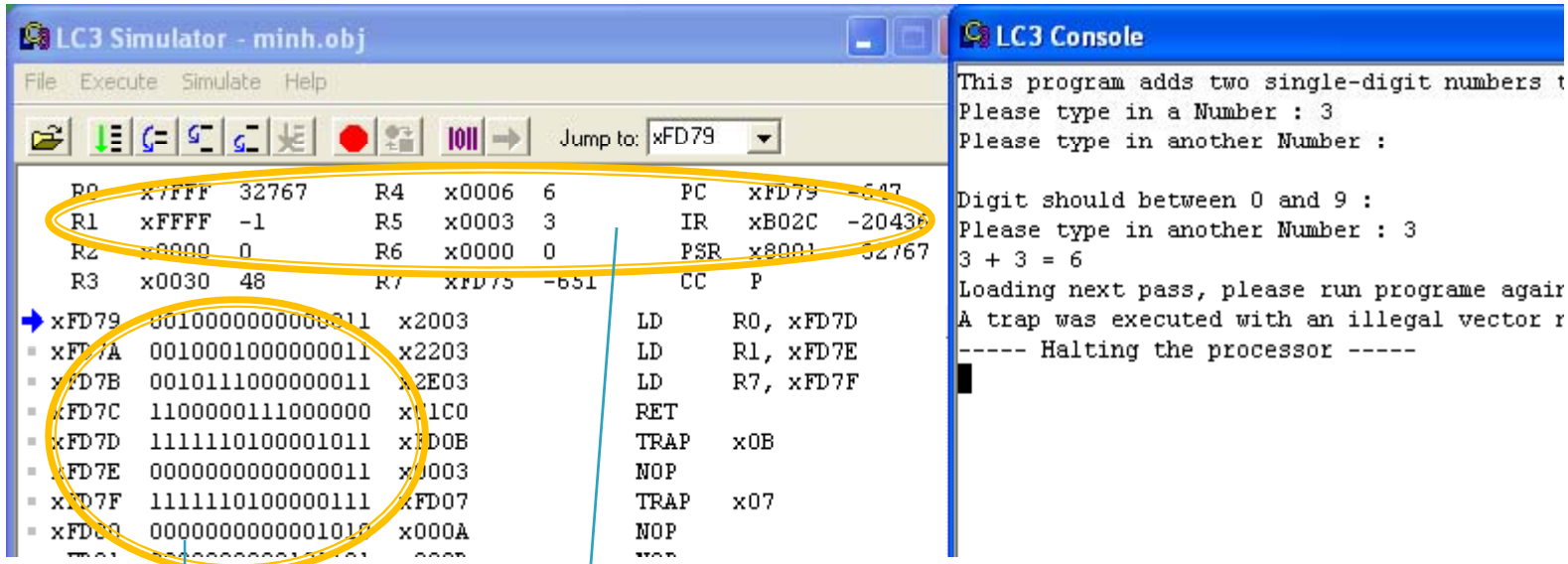
- LC3Edit is editor program (IDE).
- Simulate is LC3 simulator program (virtual computer which execute assembly code).

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.

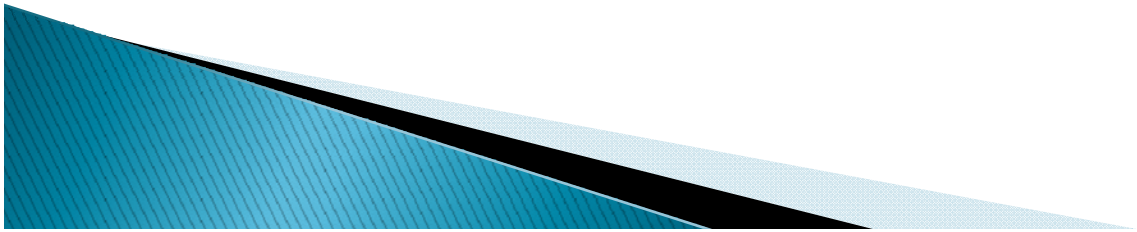
LC3 simulate.exe



- Registers and values stored in register
- Memory and values stored in memory
- 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.

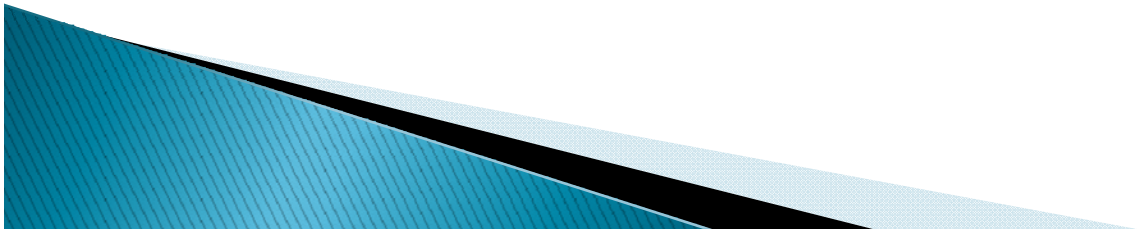


Run example: AND.asm

```
        .ORIG x3000
;;; Test AND instructions
        ADD  R1,R1,#5    ; R1:  0x5
        ADD  R2,R2,#-2   ; R2:  0xfffe
        ADD  R3,R2,R1    ; R3:  0x3
        AND  R4,R3,#-1   ; R4:  0x3
        AND  R5,R1,R4    ; R5:  0x1
        ADD  R6,R6,#-1   ; R6:  0xffff
        HALT
        .END
;;; Detail will be talked in tutorial
```

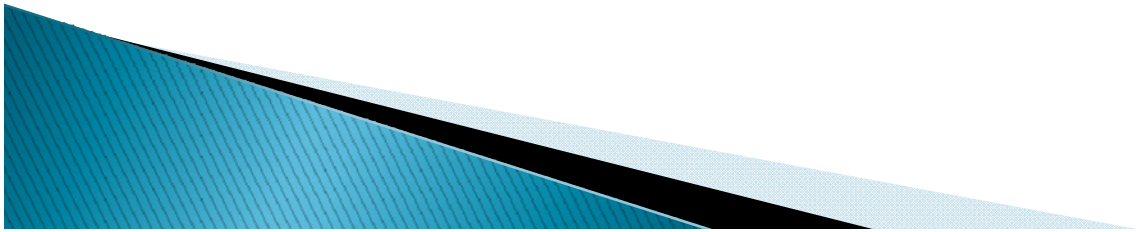
Run example: NOT.asm

```
        .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
```



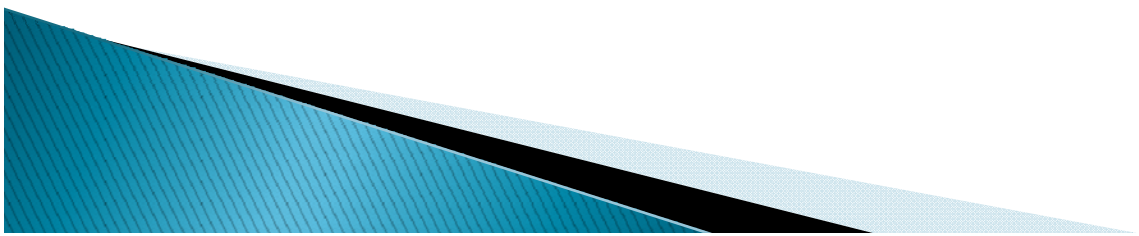
Run example: demo1.asm

```
        .ORIG       x3000
        LD    R2, Num1
        LD    R3, Num2
        ADD  R4, R2, R3
        DONE       HALT
         Num1      .FILL   5
         Num2      .FILL   6
        .END
```



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



Exercises

- Run some other examples yourselves such as **simple_calculator.asm**
- Modify examples with your knowledge of LC3 and assembly taught in class
- Try some previous data representation examples on LC3.
- Try to implement OR and XOR by modify demo1.asm