1. A Tour on Computer Systems

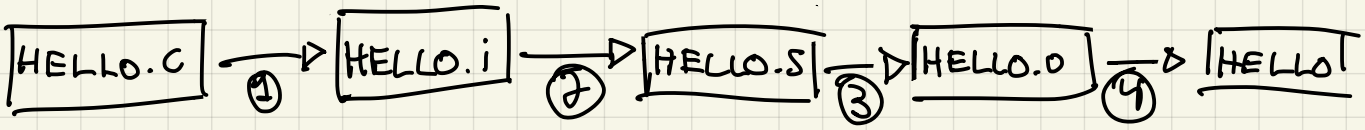
- Bit is the smallest unit stored within the computer.

Bit ⟶ Binary digit, either 0 or 1.
Nimble ⟶ 4 bits, 0 to F (16).
Byte ⟶ 8 bits, 0 to FF (255).

Sequence of bytes stored in a computer are either:

- Integer i.e. 30, 26,741, etc.
- Floating-point number i.e 20.52, 400.12, etc.
- Character string i.e. "Hi, world!", etc.
- Machine instruction i.e HLT, JMP, etc.

- On a compiled programming language, this is the compilation process:



(1) Preprocessing: modifies source code according to preprocessor directives.

(2) Compilation: translates the .i text file into an assembly-language program.

(3) Assembly: translates the .s file into machine language instructions. and packages them into a relocable object program (.o file).

(4) Linking: combines required object files together creating an executable program.

- A computer system is composed of:

- Buses: electrical circuits that transfers groups of bytes across the system called words (32/64 bits).
- I/O devices: Input/Ouput devices externalizes the computer through controllers and adapters.
- Main Memory: Temporary storage device that holds the program and its data while the processor is executing it.
- Processor: Component responsible to interpret and execute instructions. It contains a word-size storage called Program Counter (PC) where it points to some machine language instruction in the memory. It works in the following iteration:

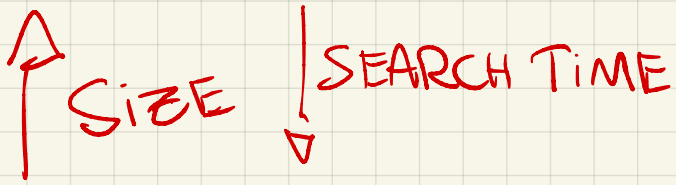
1. READS THE INSTRUCTION FROM MEMORY THAT THE PC POINTS TO.
2. INTERPRETS THE BITS IN THE INSTRUCTION.
3. PERFORMS THE OPERATION OF THE INSTRUCTION.
4. UPDATES THE PC POINTER TO THE NEXT INSTRUCTION.

Q: WHAT IS THIS ITERATION BASED ON?
A: IT'S BASED ON THE "INSTRUCTION SET ARCHITECTURE."

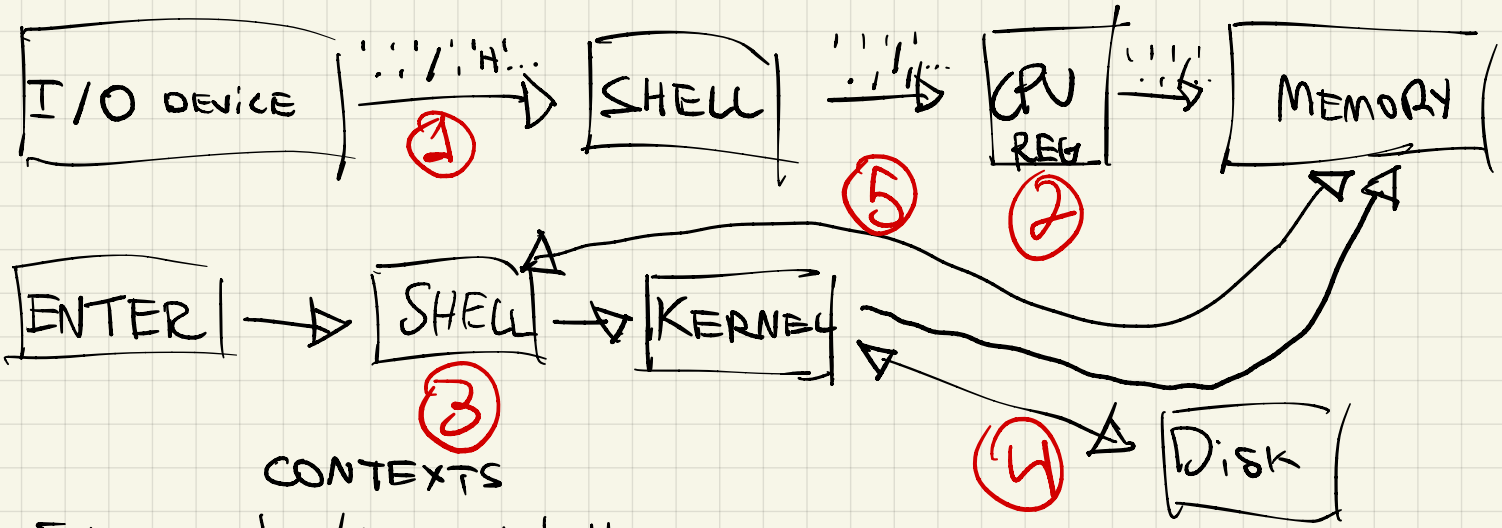
THIS CICLE IS SUMMARIZED INTO:

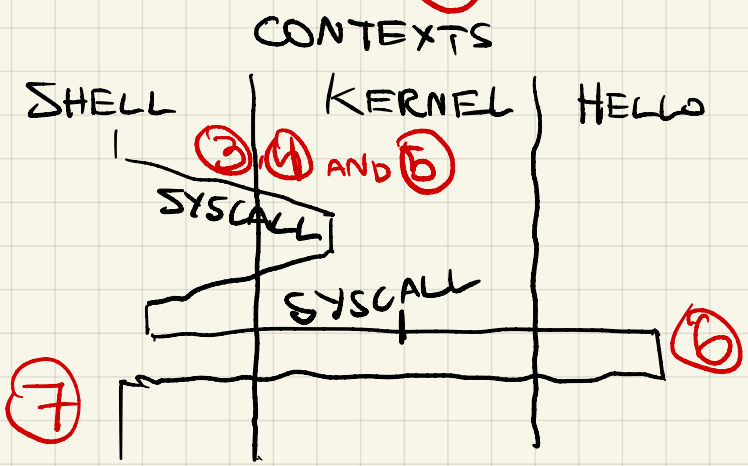
* LOAD
* STORE
* OPERATE
* JUMP

- THE COMPUTER USES CACHING TO REDUCE OVERHEAD IN COPYING CONTENTS FROM MEMORY/DISK THAT ARE REQUIRED CONSTANTLY, BY STORING THOSE INTO THE PROCESSOR'S CACHE, A WORKING MEMORY UNIT. WE HAVE L1, L2 AND L3 CACHES PRESENT INTO THE CPU.



- HOW A "HELLO.C" PROGRAM IS EXECUTED?



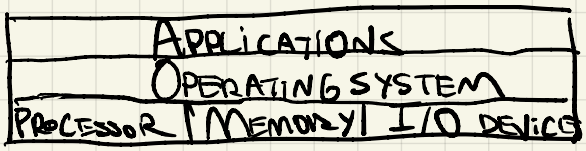


1. Each keystroke is detected by the shell program.
2. Each of those keystrokes is sent to CPU registers and is sent to the memory
3. When Enter key is pressed, the shell program makes a system call to the Operating System's Kernel.
4. The kernel will execute the system call instruction and give the execution back to the program. It'll load the hello program into memory.
5. The context is back to the shell program.
6. Shell makes another system call to Context Switch with the execution of the binary. So it executes its instructions.
7. After the binary's execution, it context switch back to the shell program.

- Programs don't interact with hardware components. They rely on services provided by the **operating system**, which is a layer between hardware and application programs.

Operating System purposes:

- Protect the hardware from programs' misuse.
- Provide equivalent interface from different kind of hardware



Operating Systems Abstractions

Abstraction	Components
Files	I/O devices
Virtual Memory	I/O devices and main memory
Processes	• I/O devices • Main memory • Processor.

- Processes
  - Abstraction of a program that's being executed.
  - Gives a feeling of exclusive usage of the processor entirely for the executing program.
  - It executes concurrently with other processes.
    - That's done by Context Switching.
  - Processes' states are saved in a structure called Context.
  - Can have multiple Threads.

- Threads
  - Abstraction for multiple executing programs that share the same context, code, and global data.

- Virtual Memory
  - Abstraction of exclusive usage of the main memory for the process.
  - May contain:
    + Program code and data.
      - Static size.
    + Heap
      - Dynamic size (may shrink or inflate).
      - Used for `malloc` and `free`.
    + Shared Libraries
      - Static size
      - Used to store the `clib` (C's standard library) and other libs.
    + Stack
      - Dynamic size (may shrink or inflate with usage).
      - Used by the compiler.
    + Kernel Virtual Memory.
      - Static size.
      - Used to store the kernel's code.
      - Only reachable through System Call invoke of the kernel.

- Files
  - Sequence of bytes.
  - All devices, including the kernel's I/O, are modeled as a file, even networks.

- Amdahl's Law
  - The overall performance of a system, including its optimization, is defined by how significant the optimized component is, and how much it sped up.

$$T_{new} = (1-\alpha)\cdot T_{old} + \frac{(\alpha \cdot T_{old})}{k}$$

$$= T_{old}\left[(1-\alpha) + \frac{\alpha}{k}\right]$$

Given speedup = S

$$\Rightarrow \boxed{S = \frac{1}{(1-\alpha) + \frac{\alpha}{k}}}$$

WHERE:

α = PERCENTAGE OF TIME CONSUMED BY COMPONENT
K = FACTOR

//

- CONCURRENCY
  - MULTIPLE AND SIMULTANEOUS ACTIVITIES
- PARALLELISM
  - USE OF CONCURRENCY TO MAKE A SYSTEM RUN FASTER.

2. Representing and Manipulating Information

- Binary is the de facto numeric representation system in computing.
  - Bits need to be grouped and have interpretation added to give meaning to them.
  - Important representations:
    - Unsigned;
    - Two complement;
    - Floating-point.

2.1. Information Storage

- Computers use blocks of 8 bits called **Bytes**.
  - All of the computer's memory is mapped in what's known as **virtual address space**.
  - To represent bytes, it's used the **Hexadecimal notation**, ranging from $00_{16} \rightarrow FF_{16}$.

- Word size = Pointer data size, either 32/64 bits.
  - Avoid types that are compiler dependent.

- A multi-byte object is stored as a sequence, with the smallest address of the bytes being used. I.e: int x = addr 0x100 (4B size ∴ 0x100, 0x101, ..., 0x103).
  - Big endian: most significant bytes come first.

0x100	0x101	0x102	0x103
01	23	45	67

= int 0x1234567 at 0x100

  - Little endian: least significant bytes come first.

0x100	0x101	0x102	0x103
67	45	23	01

= int 0x1234567 at 0x100

- Strings are sequences of characters terminated with null terminator (0x00).

- Basis of computing calculation → **Boolean Algebra**.
  - Operations: NOT, AND, OR, and XOR.
  - Can operate on **bit vectors** (string of 0s and 1s with 'w' lenght).
  - Can represent a finite set.

i.e

$A = 01001110 = \{0,3,5,6\}$
$B = 11100001 = \{0,2,4,6\}$

$A \cup B = 01001111 = \{0,3,5,6\}$
$A \cap B = \{0,6\}$

  - Bit Masking: &, | and ^.
  - Bit shifting: << and >>.

2.2. Integer Representations

- Intergers can be represented as signed (negatives, zero, and positives) and unsigned (positive's only).
- C's int types are fixed in size, except long that varies from arch (32/64 bits).
- Unsigned int representation equation:

Given $\vec{x} = [x_{w-1}, x_{w-2} \dots, x_0]$

$\therefore$

$$B2U_w(\vec{x}) \doteq \sum_{i=0}^{w-1} x_i \cdot 2^i$$

(Binary to Unsigned)

300 300 60 256 21,330 462 90

  - Every unsigned decimal has an unique unsigned bit representation that goes both ways.
- Signed int representation equation:

Given same $\vec{x}$

$\therefore$

$$B2T_w(\vec{x}) \doteq -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

(Binary to "Two-complement)

300 304,537 1,400 24,561 482 21-37

  - Follows the same principle as unsigned ints.
  - Two complement's - and + ranges are assymetrical, because 0 is non-negative.
- Converting signed to unsigned:

$$T2U_w(-\vec{x}) = -x + 2^w$$

i.e $T_2U_4(-10) = -10 + 16 = 6$

- Converting unsigned to signed:

$$U2T_w(u) = u_{w-1} \cdot 2^w + u$$