Nand2Tetris

The Tetris to Nand view

Why this part of the course?

We want to understand how run-of-the-mill computer systems work

To become better at programming them

At different levels in their stack

Tetris

- Example application
- Very interactive, computer should react fast to user input
- Both from peripherals (keyboard, mouse, touchscreen)
- And from application logic (react to current tetris construction)

Behind the screen :)

An Operating System runs the Tetris application, meaning:

The OS abstracts away the hardware capabilities in a standardized set of functionalities (for instance, input/output, access to memory)

The application developer has used these functionalities to write the source code

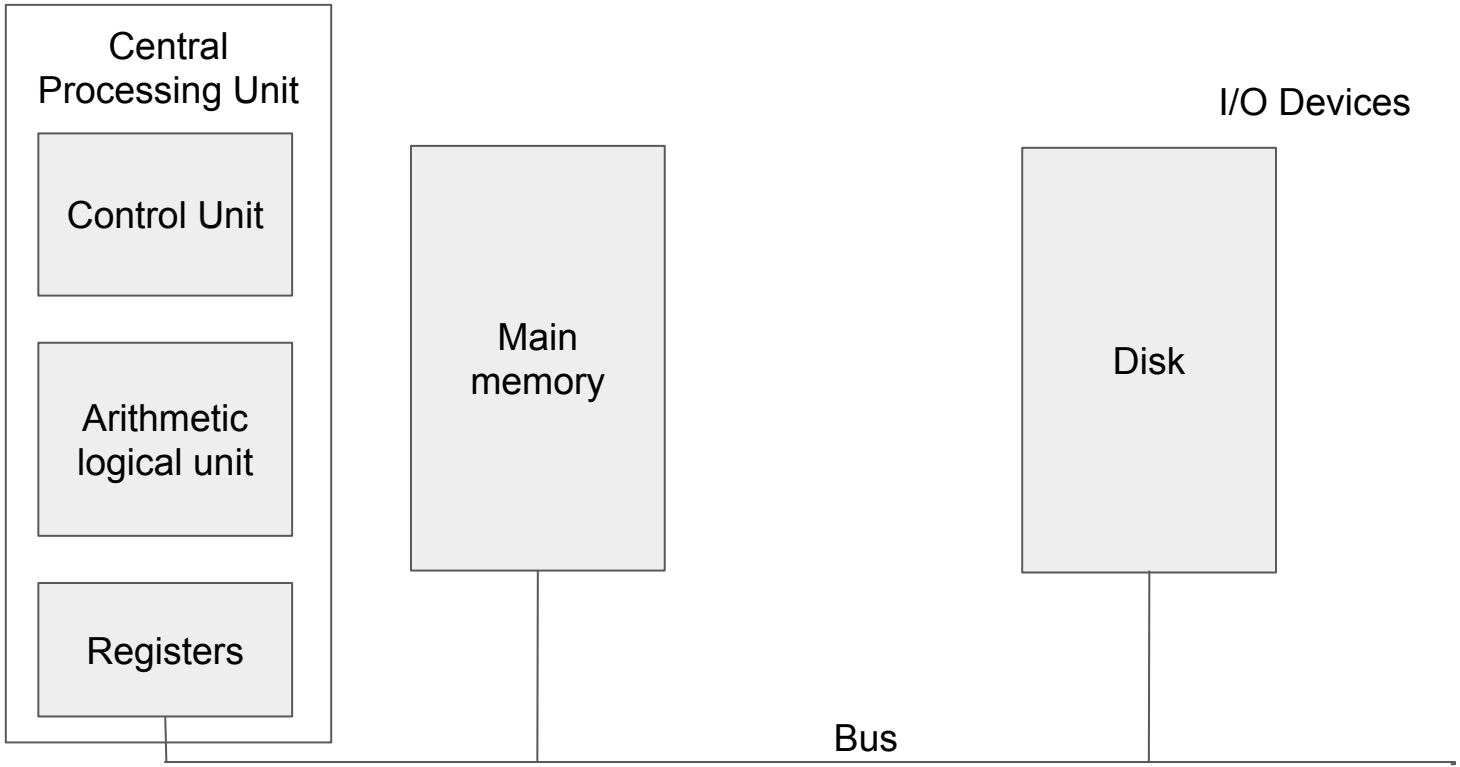
The source code has been compiled to lower level code (e.g., binary code)

The OS loads the binary code into memory

Execution starts

Von Neumann Architecture

Stored-program computer → instruction set stored in main memory, executed by the CPU



Central Processing Unit (CPU)

Control Unit

- fetches instructions from main memory
- determines their type

Arithmetic Logical Unit

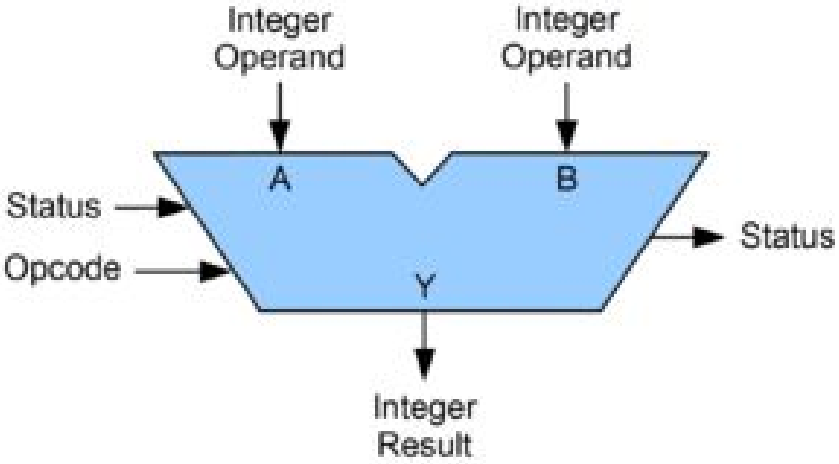
- Performs operations (addition, subtraction, Boolean AND etc)

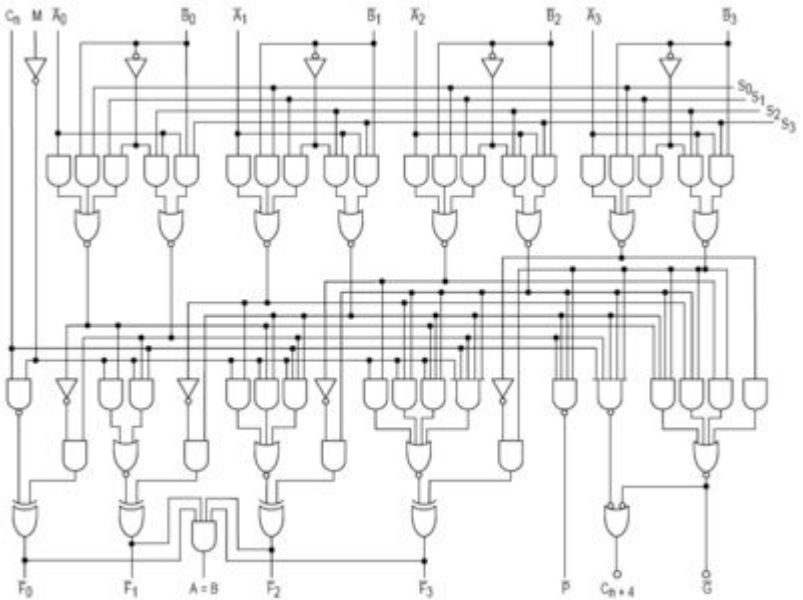
Registers

- Small, high-speed internal memory
- Each has a certain function, typically same size
- **Program Counter** → points at the next instruction that needs to be fetched
- **Instruction Register** → currently executed instruction

Arithmetic Logical Unit

- A and B are fed from registers
- The result is stored in a register



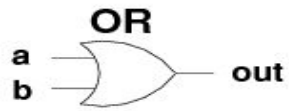


https://en.wikipedia.org/wiki/Central_processing_unit

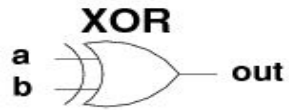
Some gate types

- We recognize on the right side truth tables
- These gates are put together in a combinatorial circuit, controlled via a control line and a multiplexor
- A multiplexor outputs one of multiple inputs

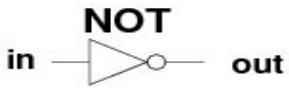
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



a	b	out
0	0	0
0	1	1
1	0	1
1	1	1



a	b	out
0	0	0
0	1	1
1	0	1
1	1	0



in	out
0	1
1	0

http://minnie.tuhs.org/CompArch/Tutes/week02.html

(Typical) Instruction types

- Register-register
  - Fetch operands from registers to the ALU input registers
  - Perform **data path cycle**
    - Run the two operands through ALU
    - Store the result in a register
- Register-memory
  - Units of data are fetched from memory to registers
  - In subsequent instructions, they become ALU inputs

(Typical) Instruction Execution

1. Fetch next instruction into the IR
2. PC = following instruction
3. Determine type of current instruction
4. If instruction uses units in data from memory, determine its location
5. If needed, fetch the unit of data into a CPU register
6. Execute instruction
7. Repeat from step 1.

Example http://www.hanshq.net/ones-and-zeros.html

```
movl $42, %eax  #x86 instruction moving integer 42 to register eax
ret
```

```
1011      1      000          00101010 00000000 00000000 00000000
Opcode    Width  Register     42
```

Big Endian Byte Order: The **most significant** byte (the "big end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

Little Endian Byte Order: The **least significant** byte (the "little end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

Nand2Tetris

Point your browsers to http://www.nand2tetris.org/software.php

- Install nand2tetris
- Open the Hardware Simulator Tutorial

Note that we may distinguish nuances in CS related to:

Emulate → imitate the internal mechanisms

Simulate → imitate the behaviour