1 Cognition, computation, computers, programs

1. Cognitive Science: mind/brain is analogous to software/hardware.

1.1. Various names: "the computational view of mind," "information processing psychology," "the computational theory of mind," or simply "computationalism."

1.2. The analogy is usually far from strict and there are many varieties.

1.3. For instance: "mind/cognition is computation" versus "mind/cognition is computable."

2. Why we need computationalism?

2.1. Compare the tasks:

- Predicting the trajectory of celestial bodies, say the motion of the earth in the next six hours.
- Predicting the next move of a chess player at a given state of the game.

2.2. In the case of chess, physical description and physical laws are helpless. The source of helplessness is twofold: (i) The state of a chess game has infinitely many physical realizations, and whether a physical state is a game state is dependent on the whether or not some cognitive agents interpret the "scene" as a chess game or not. Therefore a function from physical description to game state is at best extremely complex. (ii) Even if we find a way from physical description to game state, the physical description of rule-based behavior would be a function of the particular realization function (the mapping from physics-to-chess states) at that particular occasion.

3. Computation and cognition share an essential property: both operate on rules and representations, yet are based on (instantiated by) physical causal systems. What is happening in a computer and mind/brain are quite similar.

Levels (more on this below):

i. physical (device)

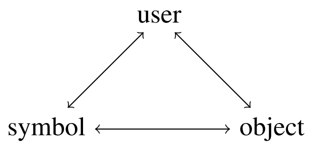
i. symbolic (syntactic)

iii. intentional (semantic)

1.1 What is (symbolic) computation?

1. Symbols (or signs) and signification are central concepts in language, logic, and computation.

1.1. Take signification as a three-part relation:



- A user uses a sign to **refer** to an object.
- A sign **denotes** an object.
- A user has certain **intentions** about an object.

1.2. Example: 00101101010011100000010.

2. A computational process is a sequence of manipulations performed over symbolic representations.

2.1. Example: the process by which you flip the digits of the representation above, one at a time is a computational process.

1.2 Hardware/software

1. Most computers around are based on von Neumann[1] architecture.

2. The main components of VNA is a **central processing unit** and a **memory**. You can think of these as the homes of **processes** and **data**.[2]

3. Memory is simpler, so let's start with it. It consists of a sequence of slots with the informational capacity of a **byte**[3]. Bytes are organized into **words**. The size of a word depends on the design; 4-bytes is a typical value. Therefore memory can be thought of as a sequence of words. Sometimes words are also called **cells**. Each cell of memory has a unique **address**.

[1] Named after the mathematician and physicist John von Neumann.

[2] But beware that the clarity of this distinction is not present in every model or architecture of computation.

[3] A byte consists of 8 bits. A bit is a binary digit which can either be 0 or 1. A single byte can hold 2^8 different values, can you see why?

Umut Özge – umozge@metu.edu.tr

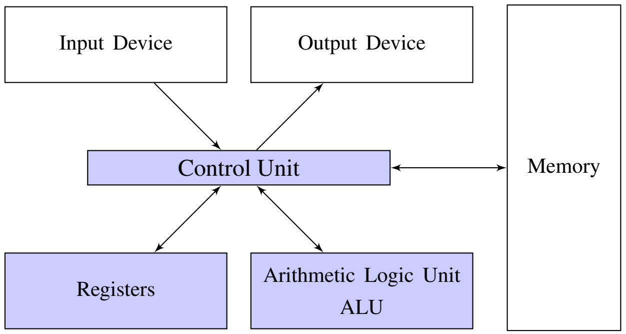


Figure 1: Von Neumann architecture

4. The two basic components, CPU and memory, communicate through three channels:
   i. A collection of wires called **address bus**;
   ii. Another collection of wires called **data bus**;
   iii. A single wire called **R/W line**, the status of which signals whether CPU wants to write to or read from the memory.

5. The address bus and the R/W line are one way channels. The value is always dictated by the CPU. The data bus is a two-way channel.

6. The two basic interactions between CPU and Memory go as follows:
   - CPU sets the address bus and R/W line to W. In that case it also sets the data bus. Obviously, this amounts to dictating to write the data to the given address in memory.
   - CPU sets the address bus and R/W line to R. In this case it is the memory that sets the data bus in accord with the data located at the address provided by CPU; this is reading from memory.

7. CPU itself also has some local memory slots. These are called **registers**. Access to registers is faster than access to memory. But there is a reason to keep the size of the CPU small, therefore there are a limited number of registers, which are used to store intermediate results of computations and some frequently used information.

8. The computation unit of CPU is **arithmetic logic unit** (ALU, for short.) ALU is responsible for arithmetic and logical operations.

9. CPU feeds on **instructions**. Some typical types of instructions are:
   i. Store the number at the register X at the memory address Y;
   ii. Fetch the number at the address X and store it at the register Y;
   iii. Add the number of address X to the number at address Y, and store the result at address Z;
   iv. Compute the bitwise *and* of the number at the address X and Y, and store the result at the address Z;
   v. If the contents of the register X and Y are not identical, go to address Z;
   vi. Jump to the address X;
   vii. and so on.

10. **Stored-program** concept
    a. Not only data but also instructions are represented as numbers;
    b. Programs are stored in memory; they can be read and written just like data.

11. A central aspect of a computational system is **flow of control**. Some instructions have *go to* or *jump*, which sends the control to another instruction. But other instructions lack such a mechanism for flow of control.

12. There is a special register called **program counter**, where the address of the next instruction is kept.

13. CPU operates through a sequence of cycles. Each cycle begins by fetching a binary description of what to do, an **instruction** from an address in Memory. Following this CPU understands, or technically speaking, **decodes** the instruction. The next step is to **execute** the instruction. This completes a single cycle, which is followed by reading the next instruction from Memory.[4]

14. Beside instructions Memory is the store for any sort of information that needs to be kept for reuse.

15. Most of our commercial computers are based on this architecture.

[4] Cycles are called 'fetch-decode-execute' or 'instruction cycle'.

1.3 Levels of programming

1. Computers operate on numbers, in the sense that the functional architecture of the machines get their instructions as binary numbers organized into expressions of **machine code**.

2. Given a functional architecture, say VNA, a straightforward specification would be to code the memory byte-by-byte (writing **machine code**); so that CPU "knows" what to do in each possible state of the process.

3. Example of an addition instruction:

   ```
   000000 10001 10010 01000 00000 1000000
   ```

4. A machine code instruction is organized into **fields** with different meanings.

5. One level up the machine code is the **assembly** language. The above machine code takes the form:

   ```
   add $s1,$t1,$t2
   ```

6. Assembly level lies between high-level languages and machine code.

   A simple while loop in C:

   ```
   while (save[i] == k)
           i = i + 1;
   ```

   Assuing that `i` and `k` are stored in registers `$s3` and `$s5`, and the array `save` starts at the memory address stored in `$s6`, the above C code is **compiled** to the following assembly code (for MIPS):

   ```
   Loop:  sll $t1,$s3,2
          add $t1,$t1,$s6
          lw  $t0,0($t1)
          bne $t0,$s5,Exit
          add $s3,$s3,1
          j   Loop
   Exit:
   ```

7. A computational process can be specified at various levels.