CSCI 120 Introduction to Computation
Inside a computer (draft)

Saad Mneimneh
Visiting Professor
Hunter College of CUNY

1 The Computer Architecture

Let's recall from Lecture 3 the architecture of a modern computer, illustrated again below.

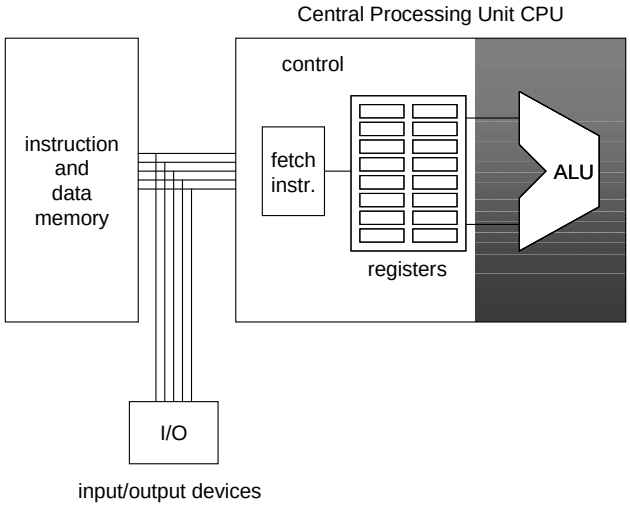

Figure 1: Computer architecture

In the figure above, we can identify five main components of a computer architecture:

1.1 Main memory

As explained before, the main memory contains both data and programs. This is basically the idea behind a stored-program computer. The program is treated as data, and is stored in memory. But how can the program be treated as data? To answer this question, we need to ask a second question: What is a program? Luckily, we know the answer to this second question. Since a program is simply the encoding of an algorithm, it consists of a sequence of computational steps that perform a certain task. Therefore, in order to store a program as data, we need to store this sequence of steps. Each step, called an instruction, will have a unique representation in bits such that the control unit of the computer will

recognize it when fetching it from memory. The set of all these instructions is called the **instruction set** of the computer (see Section 1.5).

1.2 CPU (Central Processing Unit)

The Central Processing Unit (CPU) is the *brain* of the computer. The CPU found in today's computers (e.g. Pentium and Celeron by Intel, and Athlon and Sempron by AMD) is a small flat chip (approximately two inches by two inches) that is connected by pins into a socket mounted on the computer's main circuit board, which is the *body* of the computer and usually referred to as the **motherboard**. The motherboard has a bus (collection of wires) that connected the CPU to other components such as memory and input/output controllers (keyboard, mouse, disks, monitor, etc...).

1.2.1 Control unit and registers

The control unit contains the necessary circuitry to coordinate the machine's activity. Instructions are fetched from memory for execution by the control unit. The control unit decodes the instruction and determines what needs to be done. If for instance an arithmetic operation is to be performed, the control unit forwards the desired operands to the arithmetic and logic unit ALU. Together, they form the Central Processing Unit, CPU. For temporary storage of information, the control unit uses a number of registers [modern form of Babbage s piles of disks :)]. These registers are used as place holders for the operands and the results of arithmetic or logical operations performed by the ALU (see below). A special register, called **the program counter**, holds the value of the memory location from which the next instruction needs to be fetch. Therefore, when the control unit fetches an instruction, it also increments the value in the program counter appropriately to point to the next memory location from which the next instruction will be fetched (unless the current instruction is a Jump, see Section 1.5 on instruction set). Another special register is the **instruction register** in which holds the instruction being fetched from memory.

1.2.2 ALU (Arithmetic and Logic Unit)

The ALU is responsible for performing the arithmetic operations such as addition, subtraction, multiplication, division, and relational operators such as $<$, $=$, $>$, as well as logical operations, such as the Boolean operations AND, OR, NOT, etc... Logical operations are usually performed bit-wise, e.g. 10010110 AND 11001001 = 10000000. The ALU operates on registers only. To perform operations on data stored in memory, the control unit transfers the data from memory into the registers, informs the ALU which registers hold the data, activates the appropriate circuit within the ALU to perform the desired operation, and informs the ALU which register should receive the result. The result can be then transferred from the register to memory if desired. This transfer of information (bits) to and from memory is done through a collection of parallel wires on the motherboard called the **bus**.

1.3 Bus

A nice analogy for describing the bus is the Hunter College bridge on the third floor that connects the East building to the West building and the West building to the North building. If we imagine that the West building is the CPU, the

East building is memory, and the North building is I/O, then we can visualize students as bits moving between these three components using the bridge. The bus on the motherboard acts in the same way. It is basically a collection of parallel wires. The width of the bus (i.e. number of parallel wires) depends on the size of a word in memory and on the memory capacity. For instance, to read from memory, the CPU puts on the bus the bit pattern of the address of the memory location to be read, together with the appropriate signals telling the memory that it is supposed to retrieve the data stored in that location. Similarly, when writing into memory, the CPU puts on the bus the bit pattern of the desired address of the memory location to be written, the data to be stored, and the appropriate signals telling the memory that it is supposed to store the provided data in that location. The following figure explains the concept:

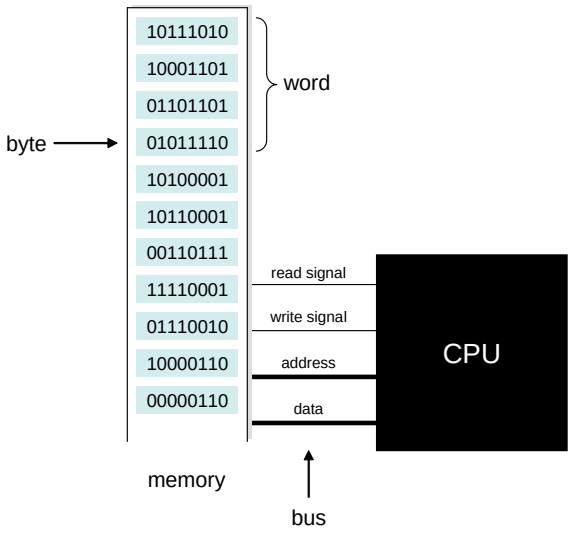

Figure 2: Bus

It is worth mentioning here that the computer's internal bus is called a **parallel bus** because data is transferred in parallel using all wires. In contrast, a **serial bus** consists of one wire where data is transferred serially one bit at a time, e.g. USB (Universal Serial Bus). USB is an also called an external bus because it is used for communication between the computer and an external device (e.g. flash drive). But this distinction is not clear, for instance, the bus that communicates data between the computer and an external hard drive is often on the motherboard (more on this later).

1.4 I/O controllers

Communication between a computer (CPU and memory) and peripheral devices such as printers, disks, mice, keyboards, etc... is normally handled through an intermediate device known as **controller** (see Figure 1). The controller may be permanently mounted on the motherboard (e.g. IDE controllers for hard disks), or may plug into a slot on the motherboard (e.g. graphic cards controllers for monitors). In either case, the controllers connects to the peripheral device by cables either within the computer case, or to a connector in the back of the

computer case, known as **port**. The device can then be attached to the port externally.

We may think of a controller as a small computer by itself. It has its own memory and CPU and is responsible for translating the data back and forth in forms compatible with the computer and the device attached to it. Therefore, each device has its own controller. However, standards have been developed to handle a variety of devices using a single controller. Such standards include USB (Universal Serial Bus). A USB controller can be used to interface between a computer and a USB compatible device. Today, most devices are USB compatible. This includes mice, keyboard, printers, external mass storage devices, scanners, digital cameras, mp3 players, etc...

As Figure 1 illustrates, a controller is connected to the bus in the same way memory is, so that the CPU can communicate data to the controller and vice-versa. Usually, the transfer of data to and from controllers is done using the same mechanism illustrated in Figure 2. To prevent conflict between a controller and main memory, however, each controller is designed to respond to unique set of addresses that main memory will ignore. This mechanism is called **memory-mapped I/O** because the computer's input/output devices appear as if they reside in various memory locations.

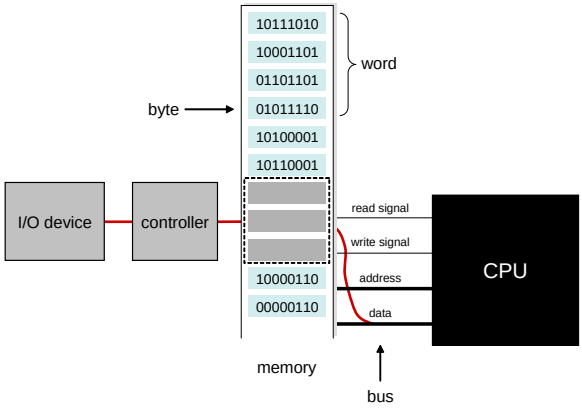

Figure 3: Memory-mapped I/O

A controller may also access memory directly using **Direct Memory Access (DMA)** (but more on this later).

1.5 Instruction set

Any computation must be modeled as a series of instructions. Therefore, another interesting question is how to come up with such an instruction set. There are two philosophies:

- Reduced Instruction Set Computer (RISC): The instruction set is very small with only a minimal number of simple instructions. With RISC, the machine is fast and efficient; however, a more complex operation would require a sequence of multiple instructions. Examples of RISC architectures: Power PC series (including those that apple call G4 and G5), developed by Apple, IBM, and Motorola.
- Complex Instruction Set Computer (CISC): The instruction set contains a large number of complex instructions, even though many of them are

technically redundant. Example of CISC architectures: Intel Pentium series.

Regardless of which philosophy is followed, any computation must be able to be carried out by a sequence of instructions from the instruction set. Here's an example: Assume we want to add two values stored in memory locations 108 and 109, and store the result in memory location 110. Since the ALU operates on registers only, the addition operation is usually carried out as follows:

1. copy the value (the bit pattern) stored in memory location 108 to one of the registers, say R
2. copy the value (the bit pattern) stored in memory location 109 to another register, say S
3. use the ALU to perform addition with R and S being the operands and store the result (the output of the ALU) in a third register, say T
4. copy the value (the bit pattern) from T to memory location 110
5. halt

The five instructions above are usually stored somewhere in memory in consecutive locations. Each instruction may occupy several bytes (or words). The control unit (see Section 1.2.1) is responsible for fetching these instructions one at a time in order and executing them.

With this addition example in mind, we can usually group the instructions of an instruction set in three groups:

- **data transfer**: This group consists of instructions that move data from one location to another. For example, steps 1, 2 and 4 of the addition operation illustrated above fall into this category. Although we use the terminology *transfer*, it is rare that the date being transferred is erased from the original location. For instance, memory location 108 still holds the same value after step 1 above. Therefore, the term *copy* provides a better description to this group of operations. Special terms are used for specific kinds of transfers or copies. Copying a value from memory to a register is called a LOAD instructions. Copying a value from a register to memory is called a STORE operation. Therefore, steps 1 and 2 are LOAD instructions, while step 4 is a STORE instruction. Some data transfers involve I/O devices (such as printers, keyboards, monitors, disks, etc...). However, these I/O transfers can be handled by the same instructions that request data transfer between the CPU and main memory (see Section 1.4 on I/O controllers).
- **arithmetic/logic**: This group consists of instructions that perform arithmetic operations, such as addition, subtraction, multiplication, division, and relational operators such as $<$, $=$, $>$, as well as logical operations, such as the Boolean operations AND, OR, NOT, etc... Logical operations are usually performed bit-wise, e.g. 10010110 AND 11001001 = 10000000. Step 3 of the addition operation illustrated above is an example of arithmetic/logic instruction.
- **control**: This group consists of instructions that direct the execution of the program rather than the manipulation of data. For example, step 5

of the addition operation illustrated above falls into this category. This group contains more interesting control instructions such as the JUMP instructions which are used to direct the control unit to fetch an instruction other than the one that appears in the next memory location. This is done by explicitly setting the program counter (see Section 1.2.1) to the desired memory location.