1 | Computers and Computer Programs

The prototypical example of a computer used in this book is the MIPS architecture (Microprocessor without Interlocked Pipeline Stages; used primarily in embedded systems, routers, game consoles, etc.), which is simple enough for novices.

The meaning of a **digital device** in this context is that at its most fundamental level, it stores and works with binary values (0s and 1s).

- No other value in a digital device (all other info are just sequences of binary values)
- Examples: computers, tablets, phones, medical equipment, etc.
- Each binary value is called a **bit**

The components we'll be discussing are two kinds: hardware and software.

There is an arbitrary distinction; historically, complex operations were implemented in software, and some operations were moved to hardware for efficiency. Anything which can be done with software can also be done with hardware.

Hardware Components

Those mentioned here are of a typical general purpose computer (like desktop or laptop). All of these components reside in the ***system unit***, which is the box housing the computer.

- **CPU (Central Processing Unit)** - where all calculations/computations are done, decisions made concerning the sequence in which computations are made.
    - Consists of registers (below), an arithmetic logic unit (ALU) capable of arithmetic and logical operations, a control unit, and other components connected w/ buses and wires
    - **Registers** - High speed memory in the CPU (see Figure 1.1); also storage elements with fast access time
        - Each stores 32 or 64 bits, and the bits are represented by 0s and 1s
        - Each register has a "register number" associated with it, starting from 0

- In MIPS architecture, 32 bits in a register, 32 general purpose registers, AKA ***programmable registers*** (values they contain can be altered at the programmer's discretion), and they:

    - Store intermediate results from arithmetic/logical computations
    - Used to move info from one location in memory to another
    - Store memory addresses

0	00000000000000000000000000000000
1	11011000010101010100010101010100
2	00010101000111101010101010010101
3	11010000011000100101010101010010
4	00000000000000001111111111111111
5	11111111111111110000000000000000
6	00000000000000000000000000000001
7	00000000000000000000000000000000

Figure 1.1: Possible values for 8 of the 32 CPU registers in the MIPS architecture

- **Program Counter (PC)** - Stores location of the instruction currently executing
    - ***Program*** (also see section below) consists of instructions stored in memory, and each instruction has a location (address)
    - The PC always knows which instruction needs to be executed next
    - Also necessary for the correct sequence of operations to take place

- **Datapath** - Hardware logic; components in the CPU which enable data to move between memory and registers
    - Performs computations
    - Makes decisions
    - Controls the flow of data in the CPU
    - Include registers, memory, ALU, PC, control unit, and connections necessary for these to work together

- **Memory**, AKA ***main memory*** or ***random access memory (RAM)***, stores data which is needed for CPU operations. It can be written.

    - Is **volatile**, meaning it requires power to retain data (ex. Not saving in MS Word, also think of it as temporary storage)
    - **Bit (Binary digIT)** is the fundamental unit (can be 0s or 1s); 8 bits = 1 byte
    - Is **byte addressable**, meaning each byte has a unique address

- Think of the byte to memory as one long sequential list of bytes
- An **address** is used to access a particular memory value when the CPU needs
- In MIPS architecture, 4 bytes to a **full word** (32 bits a word) and that's the **word size** (or **length**)
  - ***Word addresses*** increase by 4 (shown below)
  - ***Byte addresses*** increase by 1 (Difference between Byte Addressable Memory and Word Addressable Memory - GeeksforGeeks)

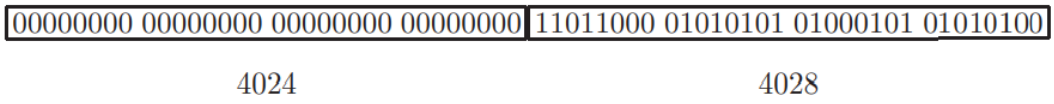

Figure 1.2: A portion of the MIPS memory, showing word addresses

- **Peripheral devices** - capable of retaining data when power is shut off (see non-volatile storage).

  Shown in conjunction with the CPU and Memory in Figure 1.3, which are placed next to each other since there is a constant exchange between them, and this needs to happen very quickly. Please note the directions of the arrows.

  - **Non-volatile storage (includes read-only memory, or ROM)**
    - Usually a ***non-removable magnetic disk*** (stores all system and application software inside computer); data here can be overwritten/removed
    - Data written to ***removable optical disks*** (like CD, DVD) aren't typically removed
    - Inside the System Unit (can't remove it)
    - Always there, retains information when powered off (permanent storage)

  - **Display (or monitor)**
    - ***Output device*** - send information from memory to the device (another example is a printer)
    - Changing something in memory changes something on the Display
    - Many modern computers include it in the System Unit
    - In most computers, a section of memory is set for info sent from memory to monitor (which is said to be ***memory mapped***)
    - Also technically can be used for both input and output

  - **Keyboard and mouse**
    - ***Input device*** - send information from device to memory (ex. joystick)
    - Changes in the keyboard/mouse changes something in memory

        - NOTE: The terms “input” and “output” are from memory’s POV

    - **USB Port (Universal Serial Bus)**
        - Shown as an input *and* output device
        - Can be connected to any number of devices that have the USB interface
        - Usually a flash drive for additional non-volatile storage; removable to transfer data to different computers
        - AKA ***flash memory***

Input and output can also take place through a wireless adapter and/or an ethernet port, which is useful for communication with other computers/internet.

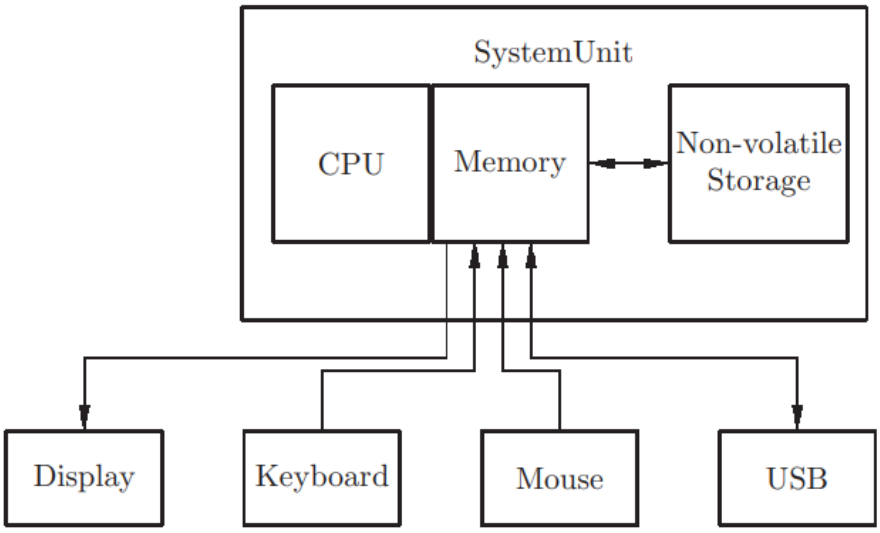

Figure 1.3: Diagram of a computer, with peripheral devices

Software Components

The CPU understands primitive instructions (such as addition and subtraction) and must be coded in binary.

- **Program** - a sequence of binary coded instructions in the computer’s memory
    - An **instruction** generally consists of an operation code (add, compare, etc.) and locations of the operands (typically registers)

- Ex. ***Register operands are coded in binary*** - When giving an instruction, we need to tell

what values are needed for the instruction; these values may be in the registers, and we need to fetch them with register numbers. The register number must be in binary; similarly, ***memory addresses are coded in binary***

In this way, we can claim that our Java programs are not actually "programs" since they're not written in binary; we can say they're just text files.

- **Machine language** - language of the binary coded instructions, with memory addresses for operands
    - When digital computers first built, Von Neumann decided that the instructions to the CPU should be stored in memory (called the **Von Neumann architecture**)
- **Assembly language** - symbolic ***mnemonics*** (memory; using a symbol to represent an instruction, like ADD instead of 000000100000) used instead of binary
    - Have an **assembler** (software) that translates assembly language program into machine language
        - Allows to use symbols for memory locations (like RESULT)
        - ***The MARS Simulator*** is an assembler/runtime system for MIPS architecture that allows you to type and edit programs, translate them to machine language, execute them, and has a debugger

So what are some others?

- **Programming languages and compilers**
    - **High level languages** allows for algebraic expressions (ex. a + b * c instead of separate instructions: MULT, ADD)
        - Examples: Java, C++, Python, Scheme
    - The **compiler** translates a program written in high level language to an equivalent program in machine language
- **Operating System** manages system resources as they're executed by the CPU. Examples include Windows, MacOS, Android, IOS, and Linux. Functionality includes:
    - Load programs into memory from a disk
    - Execute several programs at once (manages everything that needs to occur when programs executing)
    - Manage access peripheral devices (listed above)