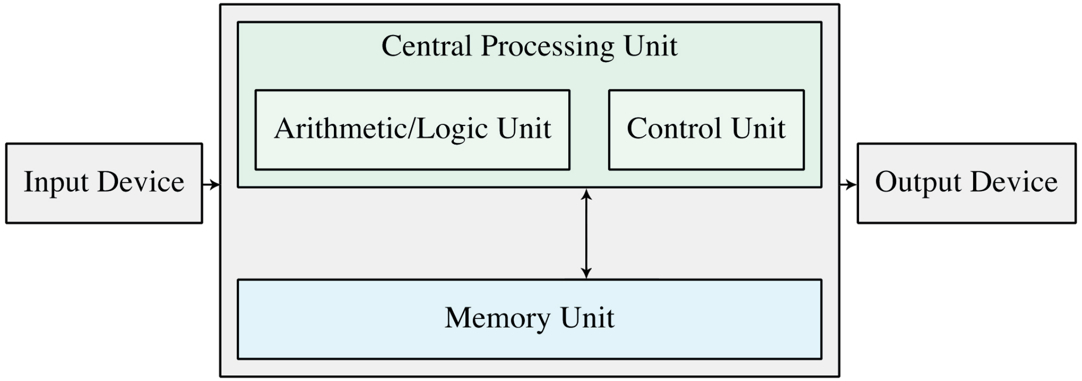
Computer Architecture & Compilation Process

Michael R. Nowak

Basic Architecture



- Majority of modern computers use the Von Neumann architecture
  - Store the program's instructions alongside its data in the memory unit
  - Partition the computer into two components: Central Processing Unit (CPU) and Memory Unit
    - The computational power resides in the CPU
    - The memory unit stores program code and data
    - The two are connected by a "bus"

Basic Architecture

- During each computation cycle, the machine retrieves the next instruction from the memory unit
- Subsequently executes the computation associated with the retrieved instruction
- This process continues until the machine is told to ‘halt’

Memory

- The smallest unit of memory is the ‘bit’, which can be in one of two states
- Computers use transistor circuits known as ‘flip-flops’ to store bits
  - It can either be on (1) or off (0)
  - Will stay in this state until it is reset

Memory

- Most computers offer ‘byte addressability’
  - Instead of interacting with bits individually, computers group eight bits together into a byte
  - Memory organizes these bytes into an array of bytes
    - Each byte in this array has a unique address
    - The CPU uses a byte’s address to read or update values encoded in its eight-bit pattern
  - The byte is the smallest addressable unit of memory in such a computer
    - To get a particular bit within a byte, shifting and masking can be used

Memory

- Most computers support reading of and writing to larger units of memory
  - Words (4-bytes) and half-words (2-bytes) are such units and span consecutive bytes in memory
    - A half-word provides $2^{16} = 65536$ patterns
      - This number is known as "64k", where 1 "k" of something is $2^{10} = 1024$
    - A word provides $2^{32} = 4294967296$ patterns (about 4 billion)
  - The base-address of these units is that of its byte with the lowest address

Memory

- Programming languages provide abstractions of these memory cells through variables and types
  - A variable is a named memory cell: we bind an identifier (i.e., give a name) to a memory cell by associating that identifier with the base-address of the respective memory cell
    - It's easier to reference a memory cell by its declared name opposed to having to remember its address
  - When we name a memory cell, we must always specify its type: the type determines the number of units of memory (i.e., bytes) composing it and how its bit-pattern is to be interpreted

Central Processing Unit

- The components that we will concern ourselves with are the registers and Arithmetic Logic Unit (ALU)
- Registers store values used in, and results of, computations by ALU
  - The number of registers in the CPU is small compared to the size of memory; however, registers can be accessed more quickly than memory
    - There are a number of specialized registers too, such as the Program Counter (PC) which holds the address of the instruction being executed
- ALU performs the actual computations, including arithmetic and logical operations

Instruction sets

- The actual things that the computer hardware can do is specified in the instruction set
- A Reduced Instruction Set Computer (RISC) processor provides limited and primitive facilities, such as
  - Loading a register from memory
  - Storing the contents of a register to memory
  - Moving to a different part of the program
  - Arithmetic operations
  - Logical operations

Machine language

- CPU instructions are stored in memory in sequential order; processing proceeds word by word from
- Each instruction is encoded as a binary sequence of numbers; the language of these instructions is known as machine language
- For instance, using the MIPs machine language, we could write the equation `wage = rate * hours` as:

```
100011 00000 00010 0000000000000000   # Load rate, register 2
100011 00001 00011 0000000000000000   # Load hours, register 3
000000 00010 00011 00100 00000 011000 # Multiply registers 2 and 3;
                                        store the result in register 4
101011 00100 00101 0000000000000000   # Store value of register 4
```

Assembly language

- Assembly language has an assembly instruction for each machine language instruction
- Unlike machine language, assembly language is entered as mnemonics (i.e., words) that describe what they do
- For instance, we could write the equation `wage = rate * hours` as:

```
lw   $s0, $s2, 0
lw   $s1, $s3, 0
mult $s2, $s3, $s4
sw   $s4, $s5, 0
```

- In order for the assembly language to be understood by the computer, we use an assembler to translate from assembly language to machine language

Higher-level languages

- It is hard for a programmer to express ideas in machine language and assembly language
- Higher-level languages use more complete mnemonics and allow more complex organization of ideas
- In C++, provided that wage had been declared, and rate and hours had been defined, we could simply write the following statement in our program:

```
wage = rate * hours;
```

Compilation

- The compiler translates high-level programming language statements into an appropriate sequence of instructions in machine language
- Several low-level instructions are typically required to express a single high-level statement
- The C++ compiler process proceeds by
    1. Preprocessing the source file
    2. Translating the source code to assembly code
    3. Translating the assembly code to machine language (object code)
    4. Linking necessary object code together into an executable file

C++ Compilation