Hierarchy of Computer Architecture
Final Project, CIS-242-AA-CRN94413

Leigh Johnson
johnsonl6@my.smccd.edu
Cañada College

December 1, 2023

Contents

1 Hierarchy of Computer Architecture

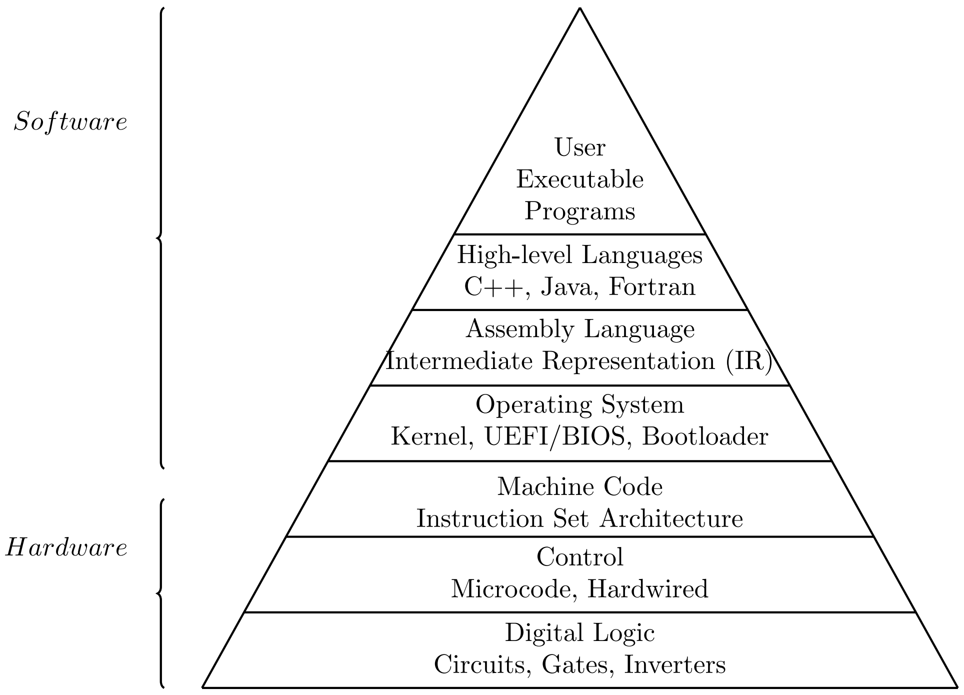
Software
Hardware
User Executable Programs
High-level Languages C++, Java, Fortran
Assembly Language Intermediate Representation (IR)
Operating System Kernel, UEFI/BIOS, Bootloader
Machine Code Instruction Set Architecture
Control Microcode, Hardwired
Digital Logic Circuits, Gates, Inverters

Figure 1: **Abstract Levels of a Computer System**

Computer architecture hierarchy refers to the structured organization of components within a computer system. This hierarchy typically starts with the smallest, simplest elements like transistors and builds up to more complex structures such as logic gates, microprocessors, and finally an entire computer operating system capable of running user programs. The components become more integrated and complex at each level of this hierarchy.

2 Digital Logic

The **Digital Logic** layer includes the physical components of the computer, like circuits, gates, and wires. Logic gates are the fundamental building block and implement operations using **Boolean Logic**.

2.1 Boolean Algebra

Named after mathematician George Boole, Boolean laws and identities can be used to reduce/simplify a logical statement. In the context of computer science, simpler circuits are preferred because they consume fewer resources (energy, money) and are simpler to build.

Name	AND form	OR form
$Identity$	$1x = 1$	$0 + x = x$
$Null$	$0x = 0$	$1 + x = 1$
$Idempotent$	$xx = x$	$x + x = x$
$Inverse$	$xx' = 0$	$x + x' = 1$
$Null$	$0x = 0$	$1 + x = 1$
$Commutative$	$xy = yx$	$x + y = y + x$
$Associative$	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
$Distributive$	$x + yz = (x + y)(x + z)$	$x(y + z) = xy + xz$
$Absorption$	$x(x + y) = x$	$x + xy = x$
$DeMorgan's$	$(xy)' = x' + y'$	$(x + y)' = x'y'$
$DoubleComplement$	$(x)'' = x$	$(x)'' = x$

Table 1: Boolean Algebra Identities and Laws

2.2 Truth Tables

Also known as **Karnaugh Maps (K-Maps)**, a **Truth Table** can be represented in two formats:

- **Sum-of-Products form** - collection of ANDed variables (product terms) that are ORed together (sum of all product terms)
- **Product-of-Sums form** - collection of ORed variables (sum terms) that are ANDed together (product all summed terms)

Consider the following example of a function *F(x,y,z) = x'yz + xy'z + xyz*, which outputs true (1) if the majority of inputs are true:

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 2: Truth Table representation for a majority function:
F(x,y,z) = x'yz + xy'z + xyz.

2.3 Logic Gates

Using **Boolean Algebra**, we can construct the next basic building block of digital circuitry: a **Logic Gate**. The most basic logic gates are:

- **AND Gate -** true (1) if both inputs are true, otherwise false (0).
- **OR Gate -** true (1) if either input is true, false (0) if both inputs are false.
- **NOT Gate -** true (1) if the input is false (0), false if the input is true. Also known as an **Inverter**

Two other gates, **NAND** and **NOR**, produce complementary output to AND and OR gates. The NAND gate is called a **universal gate** because any electronic circuit can be constructed using only NAND gates.

3 Hardware Control

Logic Gates are combined to build different types of circuitry and modules. The simplest units are **Integrated Circuits (ICs)**, a chip consisting of the necessary transistors, resistors, and capacitors to implement various gates.

3.1 Hardwired Control

3.1.1 Combinational Circuit

Combinational Circuits use a stateless circuit design, which accepts input values and (almost) instantly produces an output. One of the simplest examples is the **Half Adder**, which outputs the sum of two bits.

3.1.2 Sequential Circuit

The output of **Seqential Circuits** varies depending on the internal state (memory) of the circuit. The state changes are governed by a textbfClock, which produces a regular periodic electrical wave. Sequential circuits typically incorporate a **Feedback Loop**, which loops output back to input terminals.

A **Finite State Machine (FSM)** depicts the relationship between the state of **Flip-Flop circuity** and the system clock. FSMs can only be in one state at a time.

3.2 Microcode

Microcode is a characteristic of **Complex Instruction Set Computers (CISC)**, which are a layer of low-level instructions needed by higher-level machine code instructions. Machine code is an intermediary between the machine language and the hardware, allowing more flexible control of the processor's operations. A **Microprogram** is a sequence of microcode instructions.

3.3 Hardware Components of a Computer

3.3.1 Central Processing Unit (CPU)

The **Central Processing Unit (CPU)** is responsible for fetching program instructions, decoding each instruction, and executing the decoded operations. These steps are known as the **Fetch-Decode-Execute Cycle** or **Instruction Cycle**.

3.3.2 Memory

Memory refers to the devices that store data temporarily or permanently.

- **Random Access Memory (RAM)** - a volatile (temporary) form of memory.
- **Read-Only Memory (ROM)** - a nonvolatile (permanent) form of memory that is read-only.
- **Flash Memory (NAND, NOR)** - a non-volatile (permanent) form of memory that can be erased and rewritten.

3.3.3 Storage

A storage device is used to store data. There are many different kinds of storage mediums available for computers:

- **Magnetic Tape**
- **Optical Disks**

- **Hard Disk Drives (HDD)**
- **Solid-State Drives (SSD)**

3.3.4 Bus (Communication)

3.3.5 MARIE Reference Architecture

The CPU in the MARIE reference architecture (an example of **Von Neumann architecture**) consists of the following components:

- **Arithmetic Logic Unit (ALU)** - performs logic operations (e.g., less-than or equals comparisons) and arithmetic (add, subtract).
- **Accumulator (AC)** - a specialized register used for storing the output of the last executed operation.
- **Input / Output Registers** - specialized registers used to store user input and program output.
- **Memory Buffer Register (MBR)** - specialized register used to store data being transferred to/from main memory.
- **Memory Address Register (MAR)** - specialized register used to store the memory address of data to be fetched from main memory.
- **Control Unit (CU)** - responsible for sequencing operations, manipulating the **Program Counter**.
- **Program Counter (PC)** - specialized register used to keep track of the next instruction to be executed.
- **Instruction Register (IR)** - specialized register used to hold the instruction currently being decoded/executed.
- **Main Memory** - also known as the datapath. Stores program data.

4 Machine Code

4.1 Instruction Set Architecture(ISA)

An Instruction Set Architecture is an agreed-upon interface between *software running on a machine and the* hardware that executes it.

4.1.1 Complex Instruction Set Architecture (CISC)

4.1.2 Reduced Instruction Set Architecture (RISC)

5 Operating System

5.1 Firmware Interfaces

5.1.1 BIOS

5.1.2 UEFI

5.2 Kernel

6 Assembly Language

7 High-level Language

8 Userland