Computer Architecture and Design

Jacob Denson[1]

October 18, 2014

[1]Influenced by lectures given by Jose Amaral

Chapter 1

Computer Arithmetic

How does a computer calculate? To understand this, a more specific question needs to be asked. How does a computer represent objects that it needs to calculate. A computer cannot conjure up physical objects and maneuver them around to get results, and neither can the human brain. Both require the tools of symbolic representations to solve problems.

The most basic symbolic representation that one can think of is a number. When one sees the '2' written on this page, it does not physically show what it represents, a collection of two entities. It is just a generally agreed upon arbitrary representation of the number two. The rest of the single digit numbers follow this pattern exactly. However, from the number 10 onwards, our number system introduces a tool that makes the symbolization of numbers much more efficient: we construct all other numbers past 9 from sequences of the digits in the range 0 to 9. We can do this because of the internal relations between the numbers constructed, the operations of addition and multiplication. We define a sequence of digits in the following manner:

$$a_n a_{n-1} \dots a_2 a_1 = 10^n a_n + 10^{n-1} a_{n-1} + \cdots + 10a_2 + a_1$$

Every whole number can be represented as a unique sequence of digits 0 through 9 this manner, provided that $a_n \neq 0$. In fact, given any positive integer B, it can be proven that every number can be written uniquely in the form

$$B^n b_n + B^{n-1} b_{n-1} + \cdots + Bb_2 + b_1$$

where each number b_i is an integer from 0 to $B - 1$, and $b_n \neq 0$. Given an arbitrary set of symbols to represent each of these numbers, we can write every number as $a_n a_{n-1} \dots a_2 a_1$. We call our choice of B the base of the representation that we choose. The choice of base 10 in our *decimal* number system has no real advantage apart from the fact that the human hand has 10 fingers.

A computer is an electrical device. Everything in it thus acts as if it were a switch, turned on and on. Anything a computer can work with thus must be written in this binary way. We have shown above that an arbitrary base can be

used to represent any positive number. Like a human hand's 10 fingers, a computer's brain only has two. This provides us with the strategy of representing numbers in a computer via the binary system, where numbers are represented with a base of 2. The symbols we use for the digits of our system are off and on switches in the computer. An electrical flow is the digit one, an absense of flow is the digit zero. Given enough space to put eletrical flow, we can represent any number. We call these digits bits, short for binary digits. In the text, we use the normal symbols 0 and 1 to represent bits, and precede any number that uses the binary system with **0b**. As an example, $\mathbf{0b}\,101 = 5$.

The economic advantages to using the binary numbers in a computer system become apparent immediately. Given a number m, the number of bits required to represent the number is $\lfloor lg(m) \rfloor$. This follows as m is between $2^{\lfloor lg(m) \rfloor}$ and $2^{\lceil lg(m) \rceil}$, so, writing c as $2^{\lfloor lg(m) \rfloor}$, we see that

$$m = 2^c a_c + \cdots + 2a_2 + a_1$$

and this is the unique method of representing the number in binary. Thus c bits a_1 through a_c are required to represent the number.

An alternative notation to a number base system is tally notation, where we make one mark for each item that the number represents. We could represent this with switches in the following way. A number would be representing by the same number of on switches that it represents, terminated with an off switch. Though easy to understand, a number n would require $n + 1$ bits to be represented on a computer, much less efficient than the binary notation. The method we use is actually the most efficient possible. You can see this as every sequence of bits corresponds to one and only one number. If there was a more efficient way, numbers would have to share sequences of bits.

Another important based number system for representing computer numbers is the hexadecimal system, using base 16. We use the numbers 0 through 9 for the first few digits, and then let 10 be A, 11 be B, and so on, letting 15 be F. The reason why hexadecimal is so useful is that computers divide sets of bits into 8 bit chunks called bytes. A sequence of 4 bits fits quite nicely into 1 bytes, so it is quite easy to convert between the two once you get the hang of it.

Given two numbers in binary representation, is there an easy way to add the two numbers using the notation? If we have two numbers of the form

$$m = a_n a_{n-1} \ldots a_2 a_1 \qquad\qquad n = b_n b_{n-1} \ldots b_2 b_1$$

we can add them using the idea of the equation

$$m + n = (a_n + b_n) + (a_{n-1} + b_{n-1}) + \cdots + (a_1 + b_1)$$

We add the digits pairwise, then carry the remainder up into larger order bits to form the new sequence that represents $m + n$. This is the same method that is done in elementary arithmetic. The extension of this idea to binary digits is quite easy, and left to the reader to discover for himself. The asymptotic time that the algorithm for adding two n bit numbers turns out to be $\Theta(\log(n))$.

An important distinction to note between our theoretical description of binary numbers and the physical implementation is that on a computer, numbers are normally represented with a fixed number of bits. Customarily, these are sets of 32 bits or 4 bytes, and thus all the numbers we represent must fit in a set of 32 bits. If, given two numbers which when added result in a number greater than or equal to 2^{32}, we are unable to describe the sum of these numbers. This is called overflow – the bits required overflow the 32 bits that are given to represent the number. In most computer systems, this happens silently, we just ignore the overflow. Thus for two binary numbers,

$$m = \mathbf{0b}\underbrace{11\ldots11}_{32 \text{ times}} \qquad\qquad n = \mathbf{0b}\,1$$

the addition of these two numbers results in

$$m + n = \mathbf{0b}\,0$$

The problem we see is that, in the integers $\mathbf{Z}$, no two positive numbers result in zero. However, if we consider the numbers we are working with to be $\mathbf{Z}/t\mathbf{Z}$, where t is one number more than the maximum number we can represent on our computer, then this limitation is removed. It follows that we can consider n to be $-m$. We use this idea to extend our representation of integers to the 'signed' numbers, integers less than 0. Take an arbitrary number $x \in \mathbf{Z}/t\mathbf{Z}$, such that

$$x = \mathbf{0b}\,a_n a_{n-1} \ldots a_1$$

Then $-x \in \mathbf{Z}/t\mathbf{Z}$ is the number such that $x + -x = 0$. Given the sequence of bits above, define x' by

$$x' = \mathbf{0b}(-a_n)(-a_{n-1})\ldots(-a_1)$$

When we add x and x', we get the binary sequence $\mathbf{0b}\,11\ldots11$, so it follows that $x + x' + 1 = 0$, and thus $-x = x' + 1$ when we consider the numbers we are representing in binary to be elements of $\mathbf{Z}/t\mathbf{Z}$. Given a fixed number of bytes, we can represent $-x$ by the number we have just formed! We 'flip the bits', and add 1. We call this system the 'two's complement method'. The advantages of this system is that the way of adding these numbers is to use the simple method defined above. The disadvantage is that we lose half the amount of numbers we can represent in this system, as the largest half of the numbers we represent become the negatives of the smaller half.

In this chapter we have defined a notation that is the most useful part of the book. We have also solved problems given the constraint which forms a law of computing: every atom of information in a computer must be represented in a binary way. The two's complement method is an incredibly useful tool to represent integers. However, additional methods must be used to represent fractional numbers - it is not simple to just add a 'fractional' part to every number. We leave the method we use to represent fractionals, the 'floating point' representation, to another chapter.

Chapter 2

Components of a computer

To understand the limitations of computation on a physical machine, we must understand the components of the machine that limit what we are able to do. However, the physics that govern the machines we use is too overwhelming for a computing scientist to know, so we introduce the concept of a computer architecture, an organization of a computer into groups of parts independant of the physicality of the system. Every piece of a computer can be put into these groups, so we can consider the architecture to be 'every' computer. This abstraction enables the scientist to concentrate on the ideas more important to him or herself.

The architecture we describe divides components into three groups, with various other subgroups. They are listed below. They are the processor, the memory, and Input/Output devices (I/O). The processor performs the operation of the computer, and the memory stores information for the processor to use. Input/Output devices provide interaction with the computer and the outside world.

The Processor or Central Processing Unit (CPU) is the 'brain' of the computer. We divide this group into two main sets. The Datapath of the processor is the part of the processor which performs operations on data fed into the machine, such as additions of number and logical analysis. Control tells the rest of the computer what operations to perform.

Memory is divided into two subsections – main/primary memory, and secondary memory. The main memory holds the program while it is being run. Examples include Dynamic Random Access Memory and Cache memory. DRAM holds temporary information for the processor to use while computing, whereas Caches stores information used in immediate computation. Secondary memory holds information that is used in between computation periods. Examples include hard drives and DVDs. Normally, main/primary memory is volatile – when it turns off all information is lost. The advantage gained from this is usually much more speed. Secondary memory must be non-volatile, storing memory while the computer is turned off.

Processors in a computer are manafactured from silicon, used because it is an

excellent semiconductor. Normally, silicon does not conduct electricity, but in the presence of other elements, silicon can become either an excellent conductor, or a switch between a conductor or an insulator. A transistor falls into the last category, allowing for the choices in computation that allow a computer to function. To manafacture the processors, wafers of silicon are diced up. Because of the precision required, some of the processors become defective in the process. The fraction of processors that survive is called the yield of the batch created.

Input output devices are quite self explanatory – they allow the computer to interact with the outside world. Any device which can become a link between the computer user and the computer is an input device. Connection from the computer to the user is output. Common I/O devices are mice, computer screens, cameras.

When you are using a computer, even the architecture abstraction is abstracted for you. An operating system while manages I/O, memory and storage and scheduling tasks does all the dirty work for the user. The increasing layers of abstraction allow anyone to use a computer regardless of their knowledge on the process within.