Computer Science 202
Introduction to Programming
The College of Saint Rose
Fall 2012

Topic Notes: Computer Structures

We begin the meat of the course by considering the fundamentals of computer structures. We will examine the components of modern computers and learn about how the components work together to allow the computer to perform its tasks.

We will not attempt a formal definition of the term *computer*, but we will consider the major functions of a computer:

1. to *gather* input data from users or devices,
2. to *process* data into information,
3. to *output* data and information, and
4. to *store* data and information.

The terms *data* and *information* are often used interchangeably, but we should be more precise here.

- *Data* (the plural form of "datum") are unorganized facts and figures, without an obvious meaning.
- *Information* is obtained by processing and orgainzing data in a meaningful way.

A good example of this: your personal data such as your name, address, phone number, photograph, and so on are fairly useless on their own, but if you organize them to associate your name with your address and phone number to be able to answer a question like "What is John Smith's phone number?" we have created information.

We will spend much of our time early this semester looking more carefully at how computers gather, process, output, and store data and information.

Representing Data and Information

Before we go any further, it is necessary to think about how we can represent data and information for use by a computer. We think of our computers dealing with numbers, text, pictures, sounds, videos, and more. All of these things must be encoded in a way that a computer can gather, process, output, and store it.

The fundamental idea here is that as a computer is really just a collection of electric circuits. Those circuits transmit and store electrical signals that are either off or on. Those signals encode the only two values a computer understands: the values 0 and 1.

Anything more complicated that we wish to use must be encoded using a sequence of these 0's and 1's, which are known as *bit*s, short for **b***inary dig***its**.

The function of any computer can be boiled down to this: it takes a collection of bits, some of which are data and some of which are instructions on what to do to that data, and performs those instructions, producing a new collection of bits.

Binary Basics

Question: how high can you count on one finger?

That finger can either be up or down, so you can count 0, 1 and that's it.

(Computer scientists always start counting at 0, so you should get used to that...)

So then... How high can you count on one hand/five fingers?

When kids count on their fingers, they can get up to 5. The number is represented by the number of fingers they have up.

But we have multiple ways to represent some of the numbers this way: 1 0, 5 1's, 10 2's, 10 3's, 5 4's and 1 5.

We can do better. We have 32 different combinations of fingers up or down, so we can use them to represent 32 different numbers.

Given that, how high can you count on ten fingers?

To make this work, we need to figure out which patterns of fingers up and down correspond to which numbers.

To keep things manageable, we'll assume we're working with 4 digits (hey, aren't fingers called digits too?) each of which can be a 0 or a 1. We should be able to represent 16 numbers. As computer scientists, we'll represent numbers from 0 to 15.

Our 16 patterns are base 2, or *binary*, numbers. Again, we call the digits *bits*.

Each bit may be 0 or 1. That's all we have.

Just like in base 10 (or *decimal*), where we have the 1's (10^0) place, the 10's (10^1) place, the 100's (10^2) place, etc, here we have the 1's (2^0), 2's (2^1), 4's (2^2), 8's (2^3), etc.

As you might imagine, binary representations require a lot of bits as we start to represent larger values. Since we will often find it convenient to think of our binary values in 4-bit chunks, we will also tend to use base 16 (or *hexadecimal*).

Since we don't have enough numbers to represent the 16 unique digits required, we use the numbers 0–9 for the values 0–9 but then the letters A–F to represent the values 10–15.

```
 0  0000
 1  0001
 2  0010
 3  0011
 4  0100
 5  0101
 6  0110
 7  0111
 8  1000
 9  1001
10  1010  A
11  1011  B
12  1100  C
13  1101  D
14  1110  E
15  1111  F
```

Any number can be used as the base, but the common ones are base 2, base 8 (*octal*, 3 binary digits), base 10, and base 16.

A byte (8 bits) of data can be written as a decimal number in the range 0 to 255, as 8 binary bits, or as two hexadecimal symbols.

Two bytes of data: decimal in the range 0-65535, 16 bits, or 4 hex digits.

Decimal-Binary-Hex Conversions

Since we will encounter values in decimal, binary, and hexadecimal at various times, we should think about how to convert among these representations.

We'll first look at the easiest case: converting back and forth between binary and hex. Each hex digit represents 4 bits and each group of 4 bits can be represented by a hex digit.

So to convert the hex value `7A4D` to binary, we convert each digit:

```
0111 1010 0100 1101
```

And to go back the other way, we can start with

```
0100 1011 0000 0110
```

(conveniently written in groups of 4 bits) to obtain `4B06` in hex.

Converting from binary or hex to decimal is also pretty straightforward. We just add up the values represented by each digit. Start with a binary value:

`01011011`

This binary string is interpreted:

$$0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

which is 64+16+8+2+1 = 91 in decimal.

Notice that this is **exactly** the same way we interpret a decimal value. The value 2872 in decimal is really:

$$2 \times 10^3 + 8 \times 10^2 + 7 \times 10^2 + 2 \times 10^0$$

It's a similar idea converting from hex. Here, we'll look at the place values in base 16. Starting with the hex value `30C7`:

$$3 \times 16^3 + 0 \times 16^2 + 12 \times 16^1 + 7 \times 16^0$$

We'll probably grab a calculator for this:

$$3 \times 4096 + 12 \times 16 + 7 = 12288 + 192 + 7 = 12487$$

The remaining conversions start from decimal. We'll first convert decimal to binary. We assume here that we are going to store our result in 8 bits, but the procedure can be extended to larger values.

We will use 108 (decimal) as our value.

Let's remember what each position in our 8-bit binary value represents:

The first bit is the number of $2^7 = 128$'s in the number.
The second bit is the number of $2^6 = 64$'s
The third bit is the number of $2^5 = 32$'s
The fourth bit is the number of $2^4 = 16$'s
The fifth bit is the number of $2^3 = 8$'s
The sixth bit is the number of $2^2 = 4$'s
The seventh bit is the number of $2^1 = 2$'s
The eighth bit is the number of $2^0 = 1$'s

We work in order from left to right. If the number is greater than or equal to the value stored in a position, we place a 1 in that position and subtract the value for that position from the number. Otherwise, we place a 0 in that position.

So for our number 108, we start with the 128's place. 108 is smaller, so we place a 0 there:

```
0???????
```

Next, we notice that 108 is larger than 64, so we place a 1 in the 64's place and subtract 64 from our number. Subsequent steps will work with the number 44.

```
01??????
```

44 is greater than 32, so we have a 1 in the 32's place, and are left with 12 to work with.

```
011?????
```

There are no 16's in 12, so we put a 0.

```
0110????
```

There is an 8 in 12, so we place a 1 and subtract, leaving us with 4.

```
01101???
```

And there is a 4, so we place a 1 in the 4's and are left with 0.

```
011011??
```

The 0 remaining will not contain any 2's or 1's, so we will be placing 0's in the final two positions.

```
01101100
```

And there's our answer.

We will follow a similar procedure to this to convert decimal to hex, but it's a little more tricky since we're dealing with powers of 16.

Here, we will assume that we want the answer in 4 hex digits, and we'll start with the decimal number 19,832.

First, we need to remind ourselves what the place values are in hex:

$16^3 = 4096$
$16^2 = 256$
$16^1 = 16$
$16^0 = 1$

So we begin by figuring out how many 4096's there are in 19,832. If we divide 19,832 by 4096, we get 4, with a remainder of 3448. So our first hex digit will be a 4.

```
4???
```

And we continue working with that remainder, 3448. Note that you can also think about subtracting $4 \times 4096 = 16384$ from our starting number to get that remainder.

How many 256's are there in 3448? Well, $\frac{3448}{256}$ gives us 13, with a remainder of 120. This means we need to use 13 as our second hex digit. Recall that the character we use to represent 13 is D.

```
4D??
```

Continuing on, we are left with 120 and we need to fill in the 16's place. $\frac{120}{16}$ gives 7 with a remainder of 8. So we have our last two digits, and the hex representation of our number is

```
4D78
```

It's All Binary

So we have seen how we can represent unsigned (*i.e.*, non-negative) whole numbers in binary. However, that is just one of many kinds of data that we will wish to store and process.

- Integers (including negative numbers): the representation is similar to what we use for unsigned numbers. The usual representation is called *2's Complement*, but the details are not our concern.
- "Real" numbers with fractions/decimal points: there are many choices of how to represent non-intergral values. The representations almost always used are called *floating point* representations. Not all values can be represented exactly – approximations are needed.
- Characters: representing text. The two most common mappings of binary values to characters:
  - The *American Standard Code for Information Interchange* (ASCII) – provides a set of one-byte representations for English characters, numerical digits, and common punctuation.
    See: `http://en.wikipedia.org/wiki/ASCII`
  - *Unicode* – introduced about 20 years ago – representations are 2 bytes per character, allowing the inclusion of many international characters.
    See: `http://en.wikipedia.org/wiki/List_of_Unicode_characters`

  Text such as that in the document you are reading now is represented as a sequence of ASCII or Unicode characters.

- Other media, such as sound, images, and video each have many possible representations. When we refer to an "MP3" audio file, or a "JPEG" image, or an "MPEG" movie, we refer to a file that follows a specific set of rules about what bit patterns correspond to the sound, image, or video being represented.
- Computer instructions: both the data and instructions are stored as binary values in the computer's memory. We will look at this briefly in the next section.

Units for Measuring Data

While the fundamental unit of information is the bit, the smallest unit of data usually considered is a group of 8 bits, called a *byte*. A byte is enough to store any one of 256 (=2^8) values. Typically, a byte can represent a number or a letter or a special character (like a `!`). We will look more carefully later in the semester at how bytes can be used to represent specific number or characters.

To represent anything beyond the simplest data, we will need to use lots and lots of bytes. A page of text requires a few thousand bytes. So a several hundred-page book requires over a million bytes.

This leads us to some prefixes to indicate numbers, many of which you have likely heard used in this context:

Unit	Abbrv.	Size	
Kilobyte	KB	$2^{10} = 1024$ bytes	2.4% more than 10^3 (thousand)
Megabyte	MB	2^{20} bytes	4.8% more than 10^6 (million)
Gigabyte	GB	2^{30} bytes	7.3% more than 10^9 (billion)
Terabyte	TB	2^{40} bytes	9.9% more than 10^{12} (trillion)
Petabyte	PB	2^{50} bytes	12.5% more than 10^{15} (quadrillion)
Exabyte	EB	2^{60} bytes	15.2% more than 10^{18} (quintillion)
Zettabyte	ZB	2^{70} bytes	18% more than 10^{21} (sextillion)
Yottabyte	YB	2^{80} bytes	21% more than 10^{24} (septillion)

See: `http://www.unc.edu/~rowlett/units/large.html`

Note that each entry in the table is 1000 times larger than the previous. These are some incredibly huge numbers!

These prefixes are used elsewhere in describing sizes in computer technology. A camera's "megapixels" indicates how many millions of pixels are in the images that the camera can capture.

Hardware vs. Software

You have certainly heard the terms *hardware* and *software*. All of the physical components of the computer are considered hardware. The programs that use the hardware to perform specific tasks make up the software.

We will discuss software in more detail later. For now, note that there are two major categories of software:

- *Application software* consists of the programs you use all the time, like an email client, a web browser, and a word processor.
- *System software* or the *operating system* is the interface between the hardware and software. This is what allows you to use the same (or at least very similar) applications on very different underlying hardware.

Hardware Components

We will break hardware into four categories for now: *input devices*, *output devices*, *processors and memory*, and *persistent storage*.

An additional category of *network devices*, which allow computers to communicate with each other, will be a major topic for us a bit later.

Input Devices

Some input devices are very common, others are more specialized.

- *keyboard*: enter typed data and commands
- *mouse* or *touchpad*: select items on a screen
- *microphone*: input audio
- *scanner*: capture digital images of physical media
- *digital camera* or *webcam*: capture capture digital images or video
- *stylus*: a “pen” to “write” on a screen

Output Devices

There is also a wide variety of output devices that you may or may not be familiar with.

- *monitor*: video output (previously CRTs, now LCDs)
- *printer*: produce physical output
- *speakers*: audio output
- *projectors*: really just another type of monitor

Processors and Memory

The brains of your computer – the part that can perform calculations and store the data used by those calculations – are the processor and memory.

If you open up a desktop or laptop computer, you will find a circuit board which has attached to it many other smaller circuit boards and other devices. This is the *motherboard*. It contains slots to install a *central processing unit* (CPU), *random access memory* (RAM) and expansion devices such as sound, video, modem, or network cards. There will also be external ports, where devices can be connected.

We begin with RAM, sometimes called *main memory* or *primary storage*. Modern computers have hundreds of megabytes or up to a few gigabytes of RAM. Data stored in RAM can be accessed quickly for use in computation. However, it is a *volatile* storage, so its contents remain intact only as long as the computer remains on.

The processor is what performs the computations by executing a sequence of *instructions*, which are simple commands like "add these two numbers" or "compare these two numbers". These commands as well as the data on which they operate are stored in RAM. A modern processor is capable of performing billions of these simple operations each second.

Measuring Speed

But.. how fast is fast, when discussing RAM and processors?

The basic unit is a *cycle* – this is how long it takes for one instruction to be executed by the processor, or a value to be stored or retrieved by the memory system.

We want to measure the number of *cycles per second*, and the unit here is *hertz*, abbreviated "Hz".

A speed of 10 Hz would indicate that there are 10 cycles per second, meaning each cycle takes 0.1 seconds.

Even the earliest computers were capable of a few thousand cycles per second. Today's computers do billions.

Our prefixes come into play here again. For many years, processor speeds were measured in *megahertz* (MHz) and today most are measured in *gigahertz* (GHz).

So let's consider a processor that operates at 1 GHz. This means it can perform one billion cycles per second, and hence performs each cycle in one billionth of a second. That is an unimaginably short amount of time. There is a *clock* keeping the operations of the computer synchronized that switches from a 0 value to a 1 value a billion times per second.

Just as we saw the prefixes for large multiples of a value, there are some common prefixes used to represent fractional parts, in this case, of seconds.

Unit	Abbrv.	Length
second	s	10^0 (one)
millisecond	ms	10^{-3} (one thousandth)
microsecond	μs	10^{-6} (one millionth)
nanosecond	ns	10^{-9} (one billionth)
picosecond	ps	10^{-12} (one trillionth)

See: the SI multiples section at `http://en.wikipedia.org/wiki/Second`.

While these infinitesimal time scales are very hard to comprehend, we can think about some things we think are fast and compare them to the speed of a computer.

Consider typing at a keyboard. The world's fastest typists can input around 100 words per minute. Given an average word size of 6 characters (5 letters plus a space), how many characters per second would these typists be producing?

Contrast this with the number of instructions a 1 GHz processor is processing. In the time it takes for a very speedy human to type a character, how many instructions has the processor executed?

The "Hertz" rating for a processor is an important factor in determining how much computational power a computer has. We will consider other factors later in the semester.

Persistent Storage

Since RAM is a volatile storage, any data that needs to be stored when the computer is power off, or any data that is too large to fit in RAM, must be stored in a *persistent storage* device.

There are a number of technologies, varying in size and cost, that provide persisent storage.

- *magnetic disks*: bits of data are stored in densely-packed magnetic particles on a surface
  - *floppy disks*: becoming a legacy technology due to small storage capacity (1 MB)
  - *hard disks*: most common current technology, capacities now exceed 1 TB for both internal and external drives
- *optical discs*: compact discs (CDs), digital video discs (DVDs), Blu-ray disc – capacities from around 700 MB for a CD to 25 GB for a Blu-ray
- *flash storage*: key/thumb drives, memory cards – now can store 64 GB or more

---

Device Connection Interfaces

Next, we consider how devices (or, *peripherals*) can be connected to a computer. You have likely seen how many different devices connect. For the novice, this is just a matter of matching the right cord to the connection that fits the cord. But we will spend a bit of time considering the common interfaces on modern computers and which devices typically use them.

A peripheral is (typically) connected via a cord that ends with a *connector*. The connector has a shape that matches with a *port* on the computer. Usually, the connector has some number and configuration of metal pins, and these pins make contact with matching pin slots on the port.

It is through these connections that the peripheral device and the computer communicate using electrical signals. At most one value can be transmitted on each pin at a time, so any meaningful amount of data will need to be transmitted *sequentially*, forming a *bitstream*.

The speed of an interconnect is determined by how quickly this bitstream can be transmitted, its *transfer rate*.

- *data transfer ports*: two-way data communication
  - most common today: *universal serial bus* (USB)
  - legacy: the traditional *serial port* and *parallel port*
  - fast: *FireWire*
- *network ports*
  - *Ethernet port*: local area network connection
  - *modem port*: phone line (also becoming legacy)
- *audio/video ports*
  - *video graphics array* (VGA): connect older monitors
  - *S-video* (super video): PC/TV interconnect
  - *digital video interface* (DVI): digital LCD connection
  - *High-Definition Multimedia Interface* (HDMI): home theater

The transfer rate varies widely among these devices. These are measured in *bits per second* (bps).

Older serial ports operated at 115 Kbps and parallel ports could operate at 500 Kbps. The original USB ports operated at 12 Mbps and USB 2.0 increased that to 480 Mbps. USB 3.0 will operate at 4.8 Gbps.

FireWire devices can operate at 400 Mbps or 800 Mbps.

See: `http://www.coolnerds.com/Newbies/Ports/ports.htm`

Architecture Basics

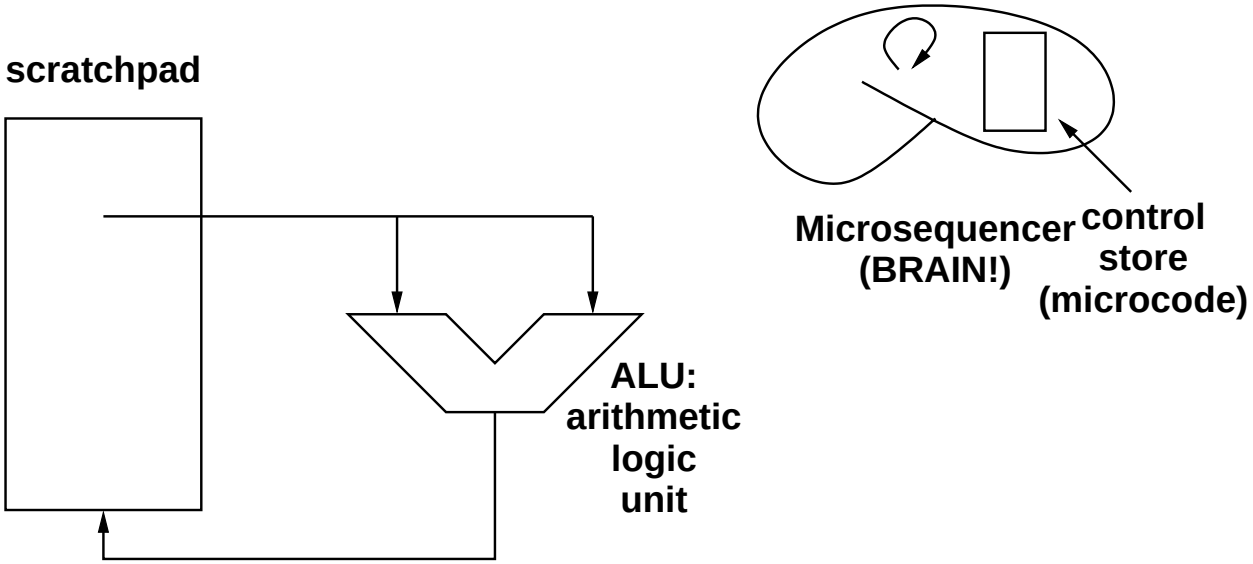
Modern computers use the *vonNeumann architecture*.

Idea: a set of instructions and a loop:

1. Fetch an instruction

2. Update next instruction location
3. Decode the instruction
4. Execute the instruction
5. GOTO 1

Basic picture of the system:



The ALU knows how to do some set of arithmetic and logical operations on values in the scratch-pad.

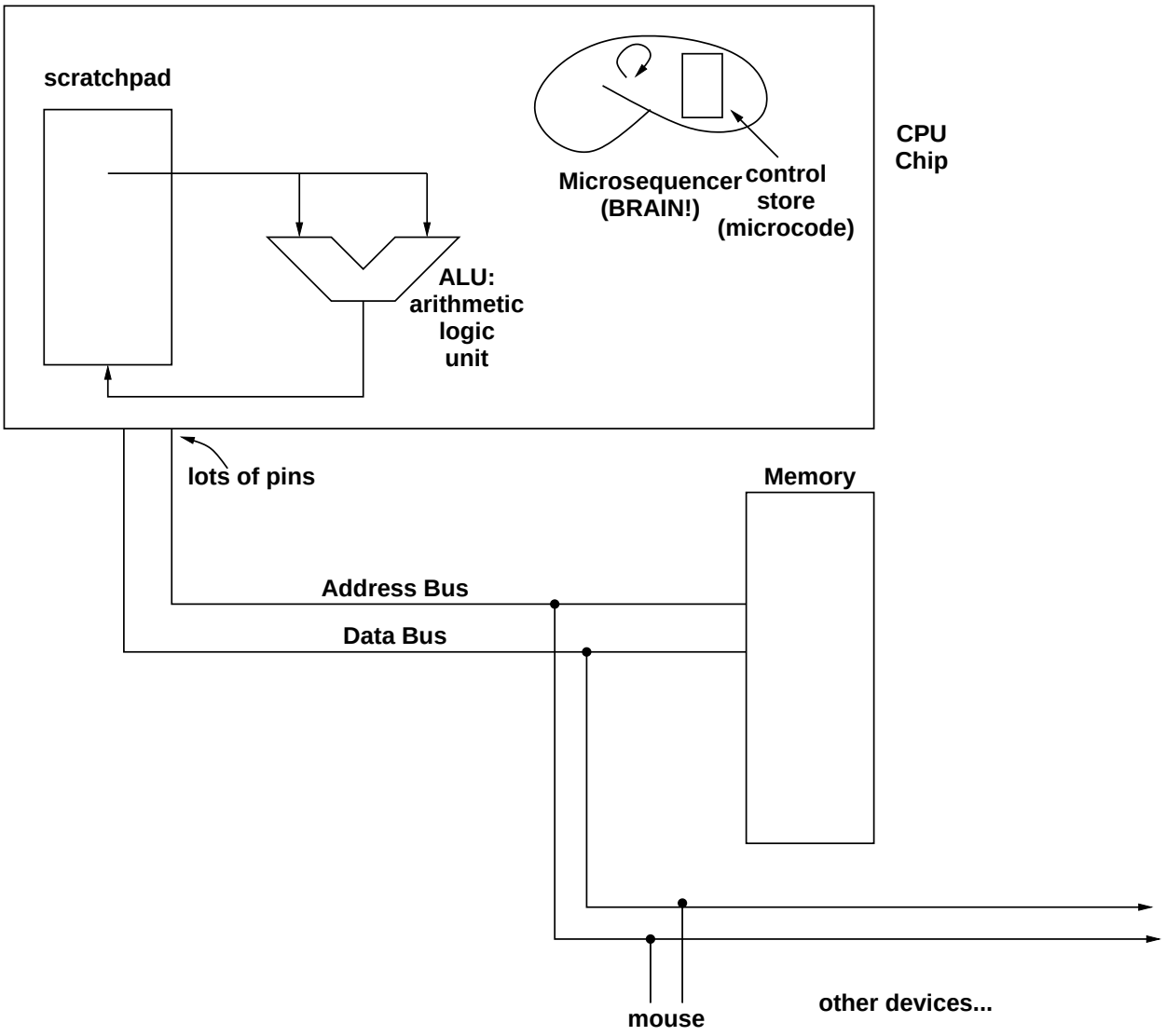
Usually the scratchpad is made up of a set of *registers*.

The micro-sequencer “brain” controls what the ALU reads from the scratchpad and where it might put results, and when.

We will not be concerned about the details of the micro-sequencer.

This is what makes up the *central processing unit (CPU)*.

We can expand this idea a bit to include memory and other devices.



The CPU interacts with memory and other devices on *buses*.

These buses are just wires that carry the electrical signals that represent the data.

If you dig deep down into any modern computer you use, it boils down to this basic process.

All of the instructions, data, and control signals come down to binary numbers.

For example, if we happened to be using a MIPS processor in our computer, and the CPU encounters this instruction

```
00000010001100100100000000100000
```

the processor will know that it means to take the numbers found in registers named `$s1` and `$s2`, add them together, and put the result into a register named `$t1`.

Somehow, the fact that this is an `add` instruction and which registers are involved is encoded in this particular 32-bit value.

Take a computer organization/architecture course to learn all of the details.