Computer Hardware
Jim Williams
HONP-112
Week 5

What we will cover
- Input and output hardware
- Processing hardware
- Storage hardware

What is Hardware?
- The term “Hardware” refers to any physical component of a computer system.
- We can look at hardware from the “top down”, starting with large devices and working our way down to the individual circuits/gates.
- In this class our discussion will not be too detailed/technical but will hopefully introduce some basic concepts.
- Remember that the examples given will not be complete, nor will they describe the many complexities found in real-life machines.
Four operations of a Computer

- Before we continue, please be aware that there are four main operations that a computer can perform:
  - Input
  - Output
  - Storage
  - Processing
- Hardware is built to allow for these operations
- We will spend a brief time on input and output, and focus more on processing and storage.

Input

- Input is an operation that converts human information into a form that the computer can “understand.”
- Remember that computers only “understand” combinations of high and low voltage states.
- So for us to use the computer for something useful, we need a way to get our information into the computer, but in a form that the computer can work with.

Input Hardware Examples

- Keyboard
  - Each key is mapped to a unique 1-byte pattern (example: the extended ASCII chart)
- Mouse
  - Each click/movement translates into binary instructions
- Microphone/Sound Card
  - Sound picked up from the air and fed into a sound card where an Analog-to-Digital converter circuit converts it to binary patterns.
- Scanner
  - A printed image is broken into dots, each dot is assigned to a bit pattern representing its shade.
So where is this ASCII Chart??

- Search the web for an ASCII chart and find one that you think is easy to read and understand.
- You should print this out and keep it with you for the remainder of this class. It is much easier looking up entries in the chart when it is on paper.
- You will need this when asked to interpret character bytes for various problems we will discuss.

Output

- Output is an operation that converts computer data into a form that humans can “understand.”
- Remember that computers only “understand” combinations of high and low voltage states. However, that is all “gibberish” to us.
- Therefore, we need a way to get the data from the computer into a form we can work with.

Output Hardware Examples

- Monitor
  - The digital information we need to see and understand is converted to a pattern of dots that make up the image on the monitor screen.
- Speaker/Sound Card
  - Binary patterns are fed into a sound card where a Digital-to-Analog converter circuit converts it to sound that it played through the speaker.
- Printer
  - A document (page, photo, etc.) in digital form is decoded by a printer which prints the pattern onto a sheet of paper that we can read.
Storage

- Storage refers to the ability to save data for later use, and retrieve data when it is needed.
- There are three main types of storage devices used in most systems today
  - Magnetic
  - Optical
  - Solid State
- Think of some examples.

Storage Types

- Magnetic: 0s and 1s are represented using patterns of magnetically-charged particles. These are read by one or many "heads."
  - Examples: Hard disks, floppy disks, tape.
- Optical: 0s and 1s are represented using high and low spots ("lands" and "pits") which are read by a laser light beam. The amount of light reflected back translates into a 0 or 1.
  - Examples: DVD, CD, etc.
- Solid State: 0s and 1s are stored by setting multiple semiconductor devices to one of two states (off/on).
  - Examples: "Flash drive", "thumb/USB" sticks, etc.

Storage: Media vs. Devices

- Think of a radio program you enjoy listening to as "data", the radio waves as the medium by which the program is delivered, and a radio receiver as a device.
- Apply this idea to computer storage:
  - A storage medium is a "delivery system" where we can store data.
  - A storage device is a piece of hardware that can access the medium (and hence the data that the medium contains).
Storage: Examples of Media

- A floppy disk
- A DVD
- A hard disk platter
- A digital tape
- A “thumb” drive

Notice that some of these are “removable” and others are not (like the hard disk platter).

Storage: Examples of Devices

- A floppy disk drive
- A DVD drive
- A hard disk driver controller card
- A digital tape deck
- A “thumb” drive port (USB controller, etc.)

NOTE: In the case of the hard disk, the medium and the device are packaged together in a single piece of machinery. But they are actually two parts.

Storage: Reading and Writing

- Some storage methods allow us to both read and write data.
- Others methods only allow us to read data (referred to as “read-only”).
- Can you think of some real life examples of both?
How does storage work?

- We will example a hypothetical example of a storage methodology.
- Our objective will be to learn the following:
  - How a typical storage medium is organized
  - How a group of data (file) is identified
  - How a file is retrieved/read
  - How a file is written

Compare to a book

- A book has a table of contents
- Based on the table of contents, you find the page of the chapter you are looking for.
- Because the pages all have numbers, and are numbered in order, you can then “jump” directly to the page you want, instead of starting at the beginning and reading through every page until you find what you are looking for.

Random Access

- Looking up information based on a table of contents is called “Random Access.”
- This means that we can jump to a specific location (it can be any number of possible locations, hence the term “random.”)
- This is the opposite of Sequential access, meaning we have to start at the beginning and go through each “page” until we find what we are looking for.
Let’s say I want to listen to a song on an album
One album is on CD. So I can just select the track number and I will “jump” directly to the track I want (random access).
One album is on tape. So I need to fast-forward or rewind to get to the song I want (sequential access).
In our storage example, we will examine Random Access storage.

Disk Storage
In the following example we will study disk storage.
This is a type of random access storage.
So we know we need some sort of table of contents.
We also know that the data on the disk itself must be grouped/organized in some way so that it can be identified as a single file.

Disk Storage – Table of Contents
A disk table of contents will contain the start and end “addresses” for the data that makes up a single file.
So when we want to open a file, the table of contents can look up the file name, get the start/end addresses, and then read in the file for us to use.
These addresses in our example will each identify a 4-byte “chunk” of 0s and 1s.
Again – this is only theoretical. Real life examples are too large and complex for this class.
Let's look at ExampleFile.txt. As we see it starts at address 4 and continues into address 6.

From Address 4 to Address 6, read in the data. For our case, assume 00000000 means “nothing” or “unused space.”
What does the file “mean”

<table>
<thead>
<tr>
<th>Address</th>
<th>4-Byte chunk of data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11101101 11111000 00000101 01101010</td>
</tr>
<tr>
<td>4</td>
<td>This</td>
</tr>
<tr>
<td>5</td>
<td>(space) class</td>
</tr>
<tr>
<td>6</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>7</td>
<td>11111111 01011011 10101011 00001000</td>
</tr>
</tbody>
</table>

- Using the ASCII chart (because we were told this is a TEXT file) – look up the character represented by each byte...write it in

<table>
<thead>
<tr>
<th>Address</th>
<th>4-Byte chunk of data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11101101 11111000 00000101 01101010</td>
</tr>
<tr>
<td>4</td>
<td>This</td>
</tr>
<tr>
<td>5</td>
<td>(space) class</td>
</tr>
<tr>
<td>6</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>7</td>
<td>11111111 01011011 10101011 00001000</td>
</tr>
</tbody>
</table>

- So the binary digits in the file represent the phrase “This class”.

Hold on a minute...

- As you can see it can become confusing to work with bytes when they are listed as 8-bit patterns of 0s and 1s.
- So let’s take a quick diversion here, and moving forward let’s use a simpler representation that is easier for us to read and write.
- Using base-16 (hexadecimal, or “hex” for short) notation allows us to write and read bytes in a more compact manner, allowing less room for error with all those 0s and 1s.
Remember our rule…

- As you already know there are $2^8$ possible patterns for a byte.
- If we looked at the byte as a binary numeric value, hopefully we will understand that the only possible values can start at 0 (all 0s) and end at 255 (all 1s).
- We cannot have any values higher or lower than that.

An example byte…

- Consider this byte: 10011101
- Calculate its numeric quantity:
  - $1 + 4 + 8 + 16 + 128 = 157$
- So, forgetting about what this byte REALLY represents in its context (i.e. a character, a number, something else...), lets just consider it to be the quantity 157 for a moment.

An example byte…

- Still working with the byte: 10011101
- Let’s instead now use Base–16 notation for the same byte.
- This would allow us to represent the same byte using many less digits.
- What is the same byte written as Base–16? There is a very easy way to do this.
Still working with the byte: 10011101

Step 1. Break the byte into its two HALVES:
  - 1001, 1101

Step 2. Determine the numeric quantity (in base-10) represented by each HALF this time.
  - 9, 13

Step 3. Now for each quantity of the halves, instead of a decimal number, use a single base-16 digit instead:
  - 9, D

So the byte 10011101 can be written instead as 9D. That is much simpler.

We are postulating that 9D in hex is the same as 10011101 in binary.

Now you already determined that the quantity represented by 10011101 is 157.

So let's examine what quantity is represented by 9D in hex:
  - (D * 1) + (9 * 16) = (13 * 1) + (9 * 16) = 13 + 144 = 157.

As you just proved, 10011101 in binary, and 9D in hex, are numerically equivalent symbols.

In most texts – and in many real-life design applications – Base-16 representation is the most common way to represent bytes.

You will see this notation come up again and again. Look at an ASCII chart for example (I hope you have researched this by now...).

Also, commonly used for representing shades of color on web pages, network addresses, computer instructions, contents of memory, etc. (You do not have to understand all the specifics now, but I am giving you a heads-up).
OK back to our example (using Base-16 this time)

<table>
<thead>
<tr>
<th>Address</th>
<th>4-Byte chunk of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ED F8 05 6A</td>
</tr>
<tr>
<td>4</td>
<td>54 68 69 73</td>
</tr>
<tr>
<td>5</td>
<td>20 63 6C 61</td>
</tr>
<tr>
<td>6</td>
<td>73 73 00 00</td>
</tr>
<tr>
<td>7</td>
<td>FF 5B 48 08</td>
</tr>
</tbody>
</table>

- From Address 4 to Address 6, read in the data. For our case, assume 00 means "nothing" or "unused space."

What does the file mean?

<table>
<thead>
<tr>
<th>Address</th>
<th>4-Byte chunk of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ED F8 05 6A</td>
</tr>
<tr>
<td>4</td>
<td>54 (T) 68 (h) 69 (i) 73 (s)</td>
</tr>
<tr>
<td>5</td>
<td>20 (space) 63 (c) 6C (l) 61 (a)</td>
</tr>
<tr>
<td>6</td>
<td>73 (s) 73 (s) 00 00</td>
</tr>
<tr>
<td>7</td>
<td>FF 5B 48 08</td>
</tr>
</tbody>
</table>

- Look up the hex values in an ASCII chart this time (easier to do). You will see that they "map" to the same characters as when we used base-2 notation. I wrote the characters in next to the hex digits. The file reads "This class."

Remember our table of contents

<table>
<thead>
<tr>
<th>File Name</th>
<th>Start address</th>
<th>End Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnotherTermPaper.doc</td>
<td>7</td>
<td>413</td>
</tr>
<tr>
<td>CuteDogPhoto.jpg</td>
<td>5632</td>
<td>5946</td>
</tr>
<tr>
<td>ExampleFile.txt</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>IllegalDownload.mp3</td>
<td>679</td>
<td>4672</td>
</tr>
<tr>
<td>Lesson2.ppt</td>
<td>423</td>
<td>523</td>
</tr>
</tbody>
</table>

- What if I wanted to add more to ExampleFile.txt and save my changes to the disk? So far the file reads "This class." I want to say more than that. Where would the new information get stored on the disk?
ExampleFile.txt uses addresses 4 through 6. I only have two bytes left there. So what do you think will happen when I add for example 11 more bytes that cannot fit in the remaining 2 bytes?

Out of room??

File Name              | Start address | End Address
-----------------------|---------------|-------------
AnotherTermPaper.doc   | 7             | 413         
CuteDogPhoto.jpg       | 5632          | 5946        
ExampleFile.txt        | 4             | 6           
IllegalDownload.mp3    | 679           | 4672        
Lesson2.ppt            | 423           | 523         
Etc...                 |               |             

Notice that I cannot simply continue the file into address 7. As you see in red, that address any many more after it are already used by another file.

Where does it go??

File Name              | Start address | End Address
-----------------------|---------------|-------------
AnotherTermPaper.doc   | 7             | 413         
CuteDogPhoto.jpg       | 5632          | 5946        
ExampleFile.txt        | 4             | 6           
IllegalDownload.mp3    | 679           | 4672        
Lesson2.ppt            | 423           | 523         
Etc...                 |               |             

Since I cannot use addresses 7–413, I will have to put the remainder of the ExampleFile.txt file somewhere else. Where??
The computer will use whatever free space it can find on the disk. ASSUME that addresses 414–422 are not used. So the remainder of the file will start at address 414 and use as much as needed.

Splitting the file into multiple parts is called **fragmentation**. It would be very inefficient to shuffle all the data on the disk around to make all files take up contiguous addresses. So the table of contents will specify which ranges of addresses a single file can be located in.

From Address 4 to Address 6, read in the data. For our case, assume 00 means “nothing” or “unused space.”
After our file was added to...

<table>
<thead>
<tr>
<th>Address</th>
<th>4-Byte chunk of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>ED F8 05 6A</td>
</tr>
<tr>
<td>4</td>
<td>54 68 69 73</td>
</tr>
<tr>
<td>5</td>
<td>20 63 6C 61</td>
</tr>
<tr>
<td>6</td>
<td>73 73 20 69</td>
</tr>
<tr>
<td>7</td>
<td>FF 5B AB 08</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>414</td>
<td>73 20 61 20</td>
</tr>
<tr>
<td>415</td>
<td>62 6F 72 65</td>
</tr>
<tr>
<td>416</td>
<td>21 00 00 00</td>
</tr>
</tbody>
</table>

Notice we only needed three more addresses to store the modified file. So the table of contents gets updated accordingly.

Updated table of contents

<table>
<thead>
<tr>
<th>File Name</th>
<th>Start address</th>
<th>End Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnotherTermPaper.doc</td>
<td>7</td>
<td>413</td>
</tr>
<tr>
<td>CuteDogPhoto.jpg</td>
<td>5632</td>
<td>5946</td>
</tr>
<tr>
<td>ExampleFile.txt (part 1)</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>ExampleFile.txt (part 2)</td>
<td>414</td>
<td>416</td>
</tr>
<tr>
<td>IllegalDownload.mp3</td>
<td>679</td>
<td>4672</td>
</tr>
<tr>
<td>Lesson2.ppt</td>
<td>423</td>
<td>523</td>
</tr>
<tr>
<td>Etc...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice we only needed three more addresses to store the modified file. So the table of contents gets updated accordingly.

What does the file “say” now??

<table>
<thead>
<tr>
<th>Address</th>
<th>4-Byte chunk of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>ED F8 05 6A</td>
</tr>
<tr>
<td>4</td>
<td>54 68 69 73</td>
</tr>
<tr>
<td>5</td>
<td>20 63 6C 61</td>
</tr>
<tr>
<td>6</td>
<td>73 73 20 69</td>
</tr>
<tr>
<td>7</td>
<td>FF 5B AB 08</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>414</td>
<td>73 20 61 20</td>
</tr>
<tr>
<td>415</td>
<td>62 6F 72 65</td>
</tr>
<tr>
<td>416</td>
<td>21 00 00 00</td>
</tr>
</tbody>
</table>

Use the ASCII chart to figure it out.
What does the file “say” now??

<table>
<thead>
<tr>
<th>Address</th>
<th>4-Byte chunk of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ED F8 05 6A</td>
</tr>
<tr>
<td>4</td>
<td>54 (T) 68 (h) 69 (l) 73 (s)</td>
</tr>
<tr>
<td>5</td>
<td>20 (space) 63 (c) 6C (l) 61 (a)</td>
</tr>
<tr>
<td>6</td>
<td>73 (s) 73 (s) 20 (space) 69 (l)</td>
</tr>
<tr>
<td>7</td>
<td>FF 5B AB 08</td>
</tr>
<tr>
<td>414</td>
<td>73 (s) 20 (space) 61 (a) 20 (space)</td>
</tr>
<tr>
<td>415</td>
<td>62 (h) 6F (o) 72 (r) 65 (a)</td>
</tr>
<tr>
<td>416</td>
<td>21 (!) 00 00 00</td>
</tr>
</tbody>
</table>

This class is a bore!

Storage example problems

- Given a table of contents and data in disk addresses, I may ask you to locate a file from the table of contents, then translate it using the ASCII chart. (what we did earlier).
- Or I may give you a sentence/words/etc. and a table of contents, and ask you to find space for the file and place the entries into the table of contents.
- Just know the relation between the table of contents and the storage addresses, and you should be able to work out problems like this.

In real life...

- The Table of Contents on a disk is called the “File Allocation Table” and is much more complex than the illustration I presented.
- Disks are broken into hierarchies of addressable space called tracks, sectors, and clusters. Our examples were not nearly as complex.
- Typical addressable “chunks” are 2MB long – much larger than the tiny 4 byte “chunks” we used in our example.
Storage review

- Know the different types of storage.
- Know what random access means as opposed to sequential access.
- Know how to represent bytes using base-16 notation.
- Know how to use the table of contents to look up a file.
- Know how to place a given file into available addresses and update the table of contents accordingly.

Processing

- So far we have only examined situations where data already exists.
- But a computer would be useless there was a way to process our data. That it, to do something with is (organize it, do calculations on it, use it to solve problems, etc.)
- The next slides will briefly discuss the pieces of electronic machinery used to process data.
- Once again, this will only scratch the surface and will not delve into the real-life complexities. Processing machinery is VERY complicated.

Processing Overview

- On the highest level there are two main types of machinery that are involved with processing data.
- These are
  - The Microprocessor
  - Memory
- There are also other important pieces of the machine that allow for processing to take place.
The Microprocessor

- The Microprocessor (AKA processor) is the “brain” of the computer.
- Some main parts of the processor include:
  - The instruction set
  - The ALU
  - Registers
  - The control unit
  - System Clock
- You may also hear the term “Central Processing Unit” (AKA “CPU”) used to describe the microprocessor. For our purposes, these terms are interchangeable.

The Instruction Set

- These are hard-wired digital circuits that know how to interpret instructions (in binary form of course) that are sent to the processor.
- Different processors use different instruction sets. This is why for example different types of computers are not compatible with each other.
- There are several different classes of instructions. We will not get into these specifics in this class.

The ALU

- The ALU stands for “Arithmetic Logic Unit”.
- This part of the processor contains the circuitry necessary for performing both arithmetic (like adding, subtracting, etc.) and logic (like AND, OR, NOT, etc.) operations.
- Generally speaking, any instruction the computer needs to perform can be broken down into either an arithmetic or logic operation.
- Oh, remember that real-life adder circuit I showed you earlier? Guess what part of the computer it is located in.
Registers

- Registers are **very small** pieces of digital machinery that store **individual** pieces of data.
- What resides in the registers changes very frequently as instructions are being processed.
- Example: two numbers being added – each addend goes into its own register before being fed into the ALU. The sum produced by the ALU will likewise be stored in yet another register.

The Control Unit

- The control unit is a special circuit that ensures all of the processor operations are taking place in the correct order, at the correct time, and tells the other parts of the machine what to do.
- It is what actually fetches an instruction, decodes it, executes it, and stores the result (by sending instructions to the other pieces of equipment).
- How many instructions the control unit can process is dependent on the system clock.

The System Clock

- This is not the "clock" that tells time.
- Rather, think of it as a constantly flipping on/off switch.
- An instruction can only execute during one of these "cycles".
- How many times the system clock cycles per second determines how "fast" the processor is (i.e. If you have a 2GHz processor that means the system clock completes about 2 billion cycles per second).
Simple Microprocessor Example

1. I want to add 2 numbers together. I am assuming there is an ADD instruction in the instruction set.
2. The ADD instruction, along with the addends, is sent to the ALU.
3. The two addends are each placed into their own registers.
4. The ALU gets the addends from the registers and adds them.
5. The ALU then places the sum into yet another register.
   The control unit makes sure all of this takes place correctly.

Memory

- Memory refers to circuitry that can store larger amounts of data/instructions/etc.
- Data exists in memory before it is acted upon by the microprocessor.
- There are two main types of memory:
  - Read-Only Memory
  - Random Access Memory

Read–Only Memory (ROM)

- Read–only memory, as the name suggests, can only be read from, not written to.
- In every computer there is a ROM chip that contains hard–wired instructions that automatically execute when the computer is turned on.
- The most important is the “Bootstrap Loader” which tells the computer how to start itself (this is where the phrase “booting” comes from).
Random Access Memory (RAM)

- RAM machinery consists of a very large number of tiny individual circuits used to temporarily store anything that the computer is “working on.”
- For instance, RAM can store a data file we are working with, or even an entire program that the computer is running.
- Technical note: RAM chips consist of millions of special logic gate circuits which are designed to store a single 0 or 1 at any given time, and maintain that value until changed.

Random Access Memory (RAM)

- Because the logic gate circuits in RAM require power to operate, RAM can only contain data as long as the machine is turned on.
- If the computer is turned off, the circuitry will lose power, and any data in RAM is therefore lost.
- To put it technically, RAM is “volatile.” Compare this to ROM, which is “non-volatile.”

Random Access Memory (RAM)

- Now we already know what Random Access means, from our previous discussion of disk storage.
- Similar to a disk, RAM also stores data in addresses.
- Data can be written to RAM, or read from RAM, in a similar fashion.
What does RAM have to do with processing??

- Remember we described our ADD instruction example?
- How does the processor know where the addends are stored (to bring them into the registers)?
- The instruction must actually contain the RAM addresses where the addends are located.
- Then the processor – in actuality the control unit piece of it – can go out to the proper RAM address, and pull these into the proper registers.

What does RAM have to do with processing??

- Continuing with our ADD example, the sum likewise needs to get from its register back to some address in RAM.
- This is because we cannot do anything with data until it exists in RAM.
- Once what we need is in RAM, we can work further with it.

Example processing problems

- You should know how to locate and retrieve bytes stored in RAM if given an address or range of addresses, and a table that shows the contents of the RAM.
- Then you can work out simple processing problems.
- There may be some problems like this on a take home exam.
An important note

- We see how we have expanded our ADD instruction example to account for the role of RAM.
- This is as far as we are going to take the example.
- But please note that in real life the execution of the instruction is more complex, and involves some other pieces of machinery.
- That is beyond the scope of this class.

What does RAM have to do with storage (part 1)??

- When we open a file from a disk, we are actually reading it into RAM so we can work with it. This applies to both data files as well as computer programs.
- Likewise, when we save a file, we are really reading it from RAM and writing it to the storage medium (disk, etc.).

What does RAM have to do with storage (part 2)??

- There is another important relationship between storage and RAM.
- When working with large files and programs, we can actually run out of RAM addresses.
- When this happens, a part of the disk is set aside and used as an extension of the RAM – it is broken down into extra RAM addresses that can be written/read. This “fools” the computer into thinking there is more RAM available than there really is.
- This is called Virtual Memory.
Consider a moment how fast it will take to read/write data from RAM chips. These are electronic circuits with no moving parts.

Now think about what happens when we run out of real RAM in the chips, and start using Virtual Memory on a disk instead.

A disk has mechanical moving parts. Do you think this is faster or slower than working with RAM in chips?? (Hint: What’s that “chugging” sound and why is my computer running so slowly??)

This concludes our brief survey of computer hardware.

Our objective was to gain a basic understanding of the different pieces and how they work together.

Hopefully you now have a greater understanding of the topic, and can use this as a starting point should you take more technical computer courses.