Introduction to computer: executes instructions
Overview

• Topics discussed in this webnote:
  – Structure and operation of the CPU
  – Program flow
  – Types of instructions that a computer can execute
Structure of a CPU

Computer system

CPU

Instruction register
Program counter
Processor status register

General purpose registers

Arithmetic Logic Unit (ALU)

Addresses

memory (RAM)

One memory cell
Components of a CPU

• **General purpose registers:**
  - Register is a *synonym* for memory in *computer science*
  - A general purpose register is a *memory cell*
  - Each general purpose register has a *unique name*
  - It is used to *store (and recall) intermediate result* of complex computations

• **Arithmetic and Logic Unit (ALU):**
  - Is a *complex electrical circuit* that can perform Mathematical (+, -, ×, /) and logical operations (<, ≤, ≥, and, or)
  - The output (result) of the computation (obtained by the ALU) is often stored in a *general purpose register*
Structure of a CPU (cont.)

- Instruction register (IR):
  - Contains the *current instruction* being executed by the CPU
  - The CPU will perform the operation indicated by the *instruction code* contained in the *instruction register*

- Program counter (PC):
  - The program counter is a *register* (memory cell)!
  - This register contains the *address (location in memory)* of the *next instruction* after the CPU finishes executing the *current instruction* in the instruction register
  - The *value* in the program counter will be *increased* after the CPU finishes executing one instruction
Structure of a CPU (cont.)

• Processor status register (PSR):

  • This register contains the *various information* about the CPU
  • Among the information contained in the PSR is:
    • The *result* of a *comparison operation*
  • When the CPU *compares* 2 numbers \(a\) and \(b\), the *outcome* of the comparison is stored in the PSR. The outcome of a compare operation will allow the CPU to determine the following fact between \(a\) and \(b\):
    • equal
    • not equal
    • less than
    • less than or equal
    • greater than
    • greater than or equal
Operation of a CPU

- The CPU performs the following sequence of operations repeatedly:

1. After executing an instruction, the CPU obtains ("fetches") the (next) instruction at the address (location) given in the program counter.

2. The fetched instruction is stored in the instruction register in the CPU and the program counter is increased to point to the next instruction in the memory.

3. The fetched instruction (stored in the instruction register) is the executed (the CPU will do the operation indicated by the instruction code in the instruction register).
• Example:

• Supposed the CPU has just finished executing the instruction (instruction code 0) in the instruction register:

(Instruction code 0 in the instruction register means "add" and it has just been executed)
The following figures illustrates the CPU's execution cycle (that goes on indefinitely):

1. The CPU first sends a request to retrieve (recall) the data stored at memory location given by the program counter (PC) (in the figure, the value of the PC = 1):

The CPU will cause the memory to retrieve (recall) the value stored at memory address (location) 1
2. In response, the **RAM memory** sends the value stored in memory location 1 (\(=\) the number 4) to the CPU which stores it in the instruction register (IR):

This number (4) will be interpreted as a code for a **computer instruction**.

The **old instruction code** (0) will be **over-written** by the **new instruction code**
3. The program counter is then increased:

Notice that the CPU is now ready to fetch the next instruction when the current instruction (stored in the instruction register) is processed.
Pointers: *pointing* to a location in memory

- Pointer:

  - A **pointer** is "something" that **points to somewhere**
  
Examples:

  - 3 pointers to different directions
In Computer Science, a **pointer** always **points** to a **location in memory**

Examples:

- Pointer to memory location 1

- Pointer to memory location 4
Pointers: *pointing* to a location in memory (cont.)

- The computer does not have "pointing finger" that point to some location in memory

- The *pointer* is represented in the computer by the *value of the address* (location) of the memory where the pointer is *pointing* at
Pointers: *pointing* to a location in memory (cont.)

- **Examples:**

  - Storing the value 1 in some register/memory cell will remember the memory location 1
  - The *value* 1 in the register/memory cell is *effectively* pointing to the memory location 1

  ![Diagram of pointing to memory location 1]

  ![Diagram of pointing to memory location 4]
Example of a pointer: the *program counter* (PC)

- The program counter (PC) is in fact a pointer

Example

When the program counter (PC) contains the value 1:

- The program counter (PC) is in fact pointing to the memory location 1:
Program flow

- Definition: program flow
  
  - Program flow = the sequence of instructions from the program executed by the CPU
  
- Default program flow:
  
  - After executing a non-branching (or non-jumping) instruction at memory location $n$, then next instruction that is executed is the instruction at memory location $n+1$. 
Program flow (cont.)

• Example:

<table>
<thead>
<tr>
<th>Memory</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>22</td>
<td>56</td>
<td>0</td>
</tr>
</tbody>
</table>

- "add"
- "compare"
- "subtract"
- "logical and"
- "negate"
- "add"

- All of the instructions in the figure are non-branching

- Therefore, the sequence of instructions executed by the CPU starting at memory location 0 is:

1. 0 ("add")
2. 4 ("compare")
3. 1 ("subtract")
4. And so on (in sequence)
Branching: changing the program flow

- Branch instruction:

  - A *branch instruction* can *alter (change)* the default program flow

  - When the *CPU* executes a *branch x instruction*, the *next instruction* that will be executed by the *CPU* is the *instruction at memory location x*
Branching: changing the program flow (cont.)

- Example:

<table>
<thead>
<tr>
<th>Memory</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>64</td>
<td>1</td>
<td>22</td>
<td>56</td>
<td>0</td>
</tr>
</tbody>
</table>

- "add"
- "branch to location 4"
- "subtract"
- "logical and"
- "negate"
- "add"

- Notice there is a *branch to location 4* instruction in the example.

- Therefore, the sequence of instructions executed by the CPU starting at memory location 0 is:

1. 0 ("add")
2. 64 ("branch to location 4")
3. 56 ("negate" --- this is the instruction at memory location 4)
4. 0 ("add")
5. And so on (in sequence again)
The *types* of instructions that a computer can execute

- The **computer** is a *computing* (reckoning) *machine*
- All existing computers (actually the CPU) execute the following 3 *types* of instructions:

<table>
<thead>
<tr>
<th>Arithmetic and logic operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
</tr>
<tr>
<td>−</td>
</tr>
<tr>
<td>×</td>
</tr>
<tr>
<td>/</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>OR</td>
</tr>
<tr>
<td>NOT</td>
</tr>
</tbody>
</table>

The result of an arithmetic and logic operation is often stored in a **general purpose register**
The *types* of instructions that a computer can execute (cont.)

- Memory transfer operations

- Transfer the *content from some specific memory location* *to* a specific register (memory cell) in the CPU.

- and vice versa (transfer the *content from some specific register (memory cell)* in the CPU *to some specific memory location*).
The *types* of instructions that a computer can execute (cont.)

- **Branch operations**

  - A *branch instruction* will cause the CPU to *branch (jump)* to the specified location in memory.
  
  - After the *jump* has occurred, the CPU will continue to execute the instructions *in sequence*, until another branch/jump instruction is encountered.

- There are 2 kinds of *branch operations*:

  1. A *unconditional branch* instruction will *always* cause the CPU to *jump* to the target location.
  
  2. A *conditional branch* instruction will *only* cause the CPU to *jump* to the target location when the *specified condition* is met.
Computer Algorithms
Fact: computers are *dumb* machines

- Basic property of a computer (a machine):

  - Computers do *what we tell* them to do
  - Unfortunately, computer do not *necessarily* do what we *want* them to do....

  (Because we can make mistake in *telling the computer* what we *want to do... These mistakes are called "bugs")
Algorithm

• Definition: **algorithm**

**Dictionary definition:**

- *Algorithm* = a step-by-step procedure for solving a problem or accomplishing some task, especially by means of a **computer**
Computer Algorithms

- Computer Algorithm:

  is an algorithm that can be executed by a computer
Properties of computer algorithms:

- The steps in an algorithm must be consists of operations that can be executed by a computer.
- The step in an algorithm must be unambiguous. (Remember that a dumb machine like a computer will do what it is told to do.)
- Computers cannot think.

Resolving ambiguity requires some thinking (intelligence) which computers cannot do!
Algorithm development

• We will now illustrate the process of developing an algorithm.
• An algorithm always accomplishes some well-defined task or solves some well-defined problem.
• The task/problem that we will use to illustrate the process of developing an algorithm is:

  • Replacing a burned out light bulb
Instruction for *humans* on replacing a burned out light bulb

- Typical instructions given to *humans* on how to replace a light bulb:
  - Remove the burned-out bulb
  - Insert a new bulb

These *brief instructions* assume *a lot of common sense knowledge* that a *machine* does not have !!!
Instruction for *humans* on replacing a burned out light bulb (cont.)

- What can go *wrong* if a *machine* uses these instructions:

  - A *machine* does not know **how** to remove a bulb
    
    It could **yank** the bulk out of its socket and damage the fixture in the process.
  
  - A *machine* does not know **how** to insert a bulb

  - A *machine* can replace the bulb with one that has an **inadequate wattage** (too bright or too dim)
Instructions for computers on replacing a burned out light bulb

- Computers have no common sense knowledge (really dumb)
  Instructions for computers must be given very explicitly (in "baby steps")
Example of how you would instruct a computer to change a light bulb:

[The following instructions will remove the burn-out bulb]
repeat until (bulb comes free of socket)
{
  turn bulb in counter-clockwise direction
}

[The following instructions will find a suitable bulb]
select a new bulb
repeat until (wattage of bulb selected = wattage of old bulb)
{
  discard the selected bulb select another bulb
}

[The following instructions will insert the new bulb]
repeat until (bulb is secure in socket)
{
  turn bulb in clockwise direction
}
A real life algorithm

Here is a famous algorithm that you can try out:

- Take 2 pieces of paper and write on one piece "A" and on the other piece "B":

- Write a positive integer on each piece of paper.
A real life algorithm (cont.)

- Euclid's Algorithm:

As long as one of the number is not zero (0) do
{
  if ( number on A ≥ number on B )
    replace the number on A by the value (A - B)
  otherwise
    replace the number on B by the value (B - A)
}

The Greatest Common Divisor (GCD) of the numbers A and B is the non-zero number on one of the papers.
A real life algorithm (cont.)

- Example of the Euclid Algorithm:

  - We start with the following 2 numbers:

    - Since number on A < number on B, we replace the number on B (= 36) by the value \((B - A) = 8\):
Now number on A ≥ number on B, so we replace the number on A (= 28) by the value \((A - B) = 20\):

![Diagram showing A = 20 and B = 8]

Again, number on A ≥ number on B, so we replace the number on A (= 20) by the value \((A - B) = 12\):

![Diagram showing A = 12 and B = 8]
A *real life* algorithm (cont.)

- Yet again, *number on A ≥ number on B*, so we *replace* the number on A (= 12) by the value \((A - B) = 4\):

```
A  B
4  8
```

- This time, *number on B > number on A*, so we *replace* the number on B (= 8) by the value \((B - A) = 4\):

```
A  B
4  4
```
A real life algorithm (cont.)

- Now, number on A ≥ number on B, so we replace the number on A (= 4) by the value (B − A) = 0:

  ![Diagram](image)

- One of the number on the papers is now zero

According to the Euclid's Algorithm, the other number (= 4) is equal to the GCD of (28, 36)
Programming a computer
What does *programming a computer* mean?

- Programming a computer:
  - Programming a computer = instruct a computer to perform a task/solve a problem

- Since a computer can only execute *machine instructions* (encoded in binary numbers), this means:
  - Write the steps of an *algorithm* using machine instructions !!!
What does *programming a computer* mean? (cont.)

This is a *painfully complicated* process

But *fortunately*, we have developed *tools* that help make this process *less painful*
Instructing a computer to execute an algorithm

- A computer is the **ideal machine** to execute **computational algorithms** because:

  - The computer can perform **arithmetic operations**
  - It can also perform an operation **only when some condition** is satisfied (using the **conditional branch instruction**)

- Problem

  - **We (a human) has to tell** (*communicate*) **the computer** what steps to do in **machine language**

Humans are **not** good in **machine languages**
Instructing a computer to execute an algorithm (cont.)

- We have developed specialized languages (based on the English language) to instruct/command a computer.

These specialized languages are called:

- *High level programming languages*
Types of languages used in computer programming

• Machine language

  • The *machine language* (or *instruction code*) consists of (binary) numbers that encode instructions for the computer
  
  • Every computer (CPU) has its own *machine language* (I.e., the instruction code 1 can encode a different instruction for different CPUs)
  
  • Instruction encoding was discussed in this webnote: click here
Types of languages used in computer programming (cont.)

- Assembler language or low level programming language

- An assembler language consists of (English like) mnemonics

- There is one mnemonic for each machine instruction of the computer
Types of languages used in computer programming (cont.)

- What an assembler program look like:

```
start
add x, y <-- one assembler instruction
sub x, y <-- one assembler instruction
...
end
```

Each assembler instruction is a mnemonic that corresponds to a unique machine instruction.
A *high level programming language* consists of *(English-like) "people" language* to *simplify* the writing computer algorithms.

A *high level programming language* allows the programmer to write *sentences* in this language which can be *easily* translated into *machine instructions*.

The *sentences* written in a high level programming language are called:

- *statements*
High level programming language (cont.)

- What an assembler program look like:

```c
main( )
{
    if ( x > y ) // One sentence (statement) in a high level prog. lang.
    {
        max = x;
    }
    else
    {
        max = y;
    }
}
.....
```
Some well-known programming languages

Well-known programming languages:

<table>
<thead>
<tr>
<th>Name</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>Scientific application (FORmula TRANslator)</td>
</tr>
<tr>
<td>Cobol</td>
<td>Business application (COmmon Business Oriented Language)</td>
</tr>
<tr>
<td>C</td>
<td>System application (successor of a language called &quot;B&quot;)</td>
</tr>
<tr>
<td>C++</td>
<td>System application (successor of the language &quot;C&quot;)</td>
</tr>
<tr>
<td>Java</td>
<td>General purpose (a simplification of &quot;C++&quot;)</td>
</tr>
<tr>
<td>C#</td>
<td>General purpose (another simplification of &quot;C++&quot;)</td>
</tr>
</tbody>
</table>
A taste of computer algorithm

- Recall the Euclid's Algorithm: (to find the **GCD**)

As long as one of the number is not zero (0) do

{  
  if ( number on A ≥ number on B )
      replace the number on A by the value (A - B)     
  otherwise
      replace the number on B by the value (B - A)  
}

The Greatest Common Divisor (GCD) of the numbers A and B is the non-zero number on one of the papers
Here is the Euclid Algorithm written in the Java programming language:

```java
public class Euclid {
    public static void main (String args[]) {
        int A; // Memory cell named "A"
        int B; // Memory cell named "B"
        // These memory cells are like the 2 pieces of paper
        // we used above. They can store and recall a value
        A = 28; // Write "28" on the piece of paper named "A"
        B = 36; // Write "36" on the piece of paper named "B"
        System.out.println("A = " + A + " B = " + B); // Print A and B
        while ( A != 0 && B != 0 ) {
            if ( A >= B )
                A = A - B; // Replace the number on A by (A-B)
            else
                B = B - A; // Replace the number on B by (B-A)
            System.out.println("A = " + A + " B = " + B); // Print new A and B
        }
        System.out.println("GCD = " + B);
    }
}
```
Representation gap: algorithm in programming and comp instructions

- Notice the following representation gap:

1. An **algorithm** expressed in a **programming language** is "English-like"

   The **algorithm** consists of **instructions** that the computer must perform to **accomplish a given task/solve a given problem**

2. *However*: the **instructions** that a computer executes are **numbers** (that encode the instruction)

   These numbers that encode **computer instructions** are *not* "English-like" !!
Bridging the translation gap: compiler

- We bridge the representation gap using 2 techniques:

  1. We **design** the **programming language** in such a way that the **sentences** in the programming language can be **easily** understood

  2. We write a **(pretty complex) computer program** to do the **translation** from:

     - Algorithm written in a program language ⇒ **same algorithm in machine instruction (numbers)**
Bridging the translation gap: compiler (cont.)

• The *(pretty complex)* computer program that does this translation is called:

- **a compiler**
Interacting with a computer (in general)
Before we can program a computer...

• Before we can learn to instruct a computer, we must first learn to:
  - interact with a computer

• Devices used to interact with a computer:
  • Mouse
  • Touch pad
  • Keyboard
Before we can program a computer...
(cont.)

- For *casual* computer users, the most important interaction device is:
  - mouse (and a little keyboarding for chatting)...

- For *computer programmers*, the most important interaction device is:
  - keyboard !!!

because they must *write programs* in an *English-like programming language*

(I dare you to write a book using a mouse by clicking on a virtual keyboard !)
How a human interact with a computer

• Preliminary step:

  • We must gain authorization to use the computer system
  • This is usually achieve through an authentication process (popularly known as log in process)

• After log in is successful, we interact with a computer by:

  • Running various computer applications
How a human interacts with a computer (cont.)

- The application that you need to run depends on the task that you want to perform

Examples:

<table>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surf the Internet</td>
<td>Web browser (such as FireFox, IE, Chrome, etc)</td>
</tr>
<tr>
<td>Edit a report</td>
<td>Text editor (such as Word, gedit, vi, emacs, etc)</td>
</tr>
<tr>
<td>Email</td>
<td>Email client (such as Thunderbird)</td>
</tr>
<tr>
<td>Ledger</td>
<td>spreadsheet (such as Microsoft Excel)</td>
</tr>
<tr>
<td>Database application</td>
<td>Database server (such as SQL server)</td>
</tr>
</tbody>
</table>
Computer Operating Systems

- Every computer is controlled (= "managed") by a very complex computer program called an Operating System (OS)
- The Operating System (OS) controls:
  - The mouse
  - The keyboard
  - The monitor
  - The disks
  - ...
  - Every component of the computer !!!
Computer Operating Systems (cont.)

- The Operating System (OS) can also:
  - Detect mouse clicks
  - Detect key strokes
  - Load a computer program from hard disk into RAM memory and execute it
Today's most popular Operating Systems

• Microsoft Windows

  • First developed in 1980 for the IBM-PC by Microsoft
  • Mostly used in PCs
Today's most popular Operating Systems (cont.)

• UNIX

  • First developed in the early 1970's for “mini”-computers by AT&T

  (A mini computer is a computer that is less power than a main-frame computer in the 1970's)

  • Used in PCs, and servers

  • A history of UNIX:
     http://www.unix.org/what_is_unix/history_timeline.html
Today's most popular Operating Systems (cont.)

- **MAC OS**
  - First developed in 1984 for the Macintosh PC by Apple
  - Used in Apple's computers
  - A history of MAC OS:
Organizing your files

- When you have a small number (like 10 or so) of files, you can put the files in the same location (e.g., the desk top) and you can find them back easily.

- However, if you have a large number (like 1000 or more) of files, you will have a hard time locating a file if they are put in one location.
Encoding

• A computer file is a sequent of bytes and it can be readily placed into RAM.
• **Most files** use the **ASCII encoding** method.
  – For details on ASCII encoding, see AsciiTable.com
  – However, files containing **non-English** characters (e.g. Greek letters, Chinese characters, etc.), use **Unicode encoding** method.
  – for details on Unicode encoding, see Unicode.org
## ASCII encoding

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32</td>
<td>20 040</td>
<td>$#32$</td>
<td>Space</td>
<td>64</td>
<td>40 100</td>
<td>$#64$</td>
<td>@</td>
<td>96</td>
<td>60 140</td>
<td>$#96$</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21 041</td>
<td>$#33$</td>
<td>!</td>
<td>65</td>
<td>41 101</td>
<td>$#65$</td>
<td>A</td>
<td>97</td>
<td>61 141</td>
<td>$#97$</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22 042</td>
<td>$#34$</td>
<td>&quot;</td>
<td>66</td>
<td>42 102</td>
<td>$#66$</td>
<td>B</td>
<td>98</td>
<td>62 142</td>
<td>$#98$</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23 043</td>
<td>$#35$</td>
<td>#</td>
<td>67</td>
<td>43 103</td>
<td>$#67$</td>
<td>C</td>
<td>99</td>
<td>63 143</td>
<td>$#99$</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24 044</td>
<td>$#36$</td>
<td>$</td>
<td>68</td>
<td>44 104</td>
<td>$#68$</td>
<td>D</td>
<td>100</td>
<td>64 144</td>
<td>$#100$</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25 045</td>
<td>$#37$</td>
<td>%</td>
<td>69</td>
<td>45 105</td>
<td>$#69$</td>
<td>E</td>
<td>101</td>
<td>65 145</td>
<td>$#101$</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26 046</td>
<td>$#38$</td>
<td>&amp;</td>
<td>70</td>
<td>46 106</td>
<td>$#70$</td>
<td>F</td>
<td>102</td>
<td>66 146</td>
<td>$#102$</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27 047</td>
<td>$#39$</td>
<td>'</td>
<td>71</td>
<td>47 107</td>
<td>$#71$</td>
<td>G</td>
<td>103</td>
<td>67 147</td>
<td>$#103$</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28 050</td>
<td>$#40$</td>
<td>(</td>
<td>72</td>
<td>48 110</td>
<td>$#72$</td>
<td>H</td>
<td>104</td>
<td>68 150</td>
<td>$#104$</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>HT (horizontal tab)</td>
<td>41</td>
<td>29 051</td>
<td>$#41$</td>
<td>)</td>
<td>73</td>
<td>49 111</td>
<td>$#73$</td>
<td>I</td>
<td>105</td>
<td>69 151</td>
<td>$#105$</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>012</td>
<td>LF (line feed, new line)</td>
<td>42</td>
<td>2A 052</td>
<td>$#42$</td>
<td>*</td>
<td>74</td>
<td>4A 112</td>
<td>$#74$</td>
<td>J</td>
<td>106</td>
<td>6A 152</td>
<td>$#106$</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B 053</td>
<td>$#43$</td>
<td>+</td>
<td>75</td>
<td>4B 113</td>
<td>$#75$</td>
<td>K</td>
<td>107</td>
<td>6B 153</td>
<td>$#107$</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>014</td>
<td>FF (form feed, new page)</td>
<td>44</td>
<td>2C 054</td>
<td>$#44$</td>
<td>,</td>
<td>76</td>
<td>4C 114</td>
<td>$#76$</td>
<td>L</td>
<td>108</td>
<td>6C 154</td>
<td>$#108$</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D 055</td>
<td>$#45$</td>
<td>-</td>
<td>77</td>
<td>4D 115</td>
<td>$#77$</td>
<td>M</td>
<td>109</td>
<td>6D 155</td>
<td>$#109$</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>016</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E 056</td>
<td>$#46$</td>
<td>.</td>
<td>78</td>
<td>4E 116</td>
<td>$#78$</td>
<td>N</td>
<td>110</td>
<td>6E 156</td>
<td>$#110$</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>017</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F 057</td>
<td>$#47$</td>
<td>/</td>
<td>79</td>
<td>4F 117</td>
<td>$#79$</td>
<td>O</td>
<td>111</td>
<td>6F 157</td>
<td>$#111$</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30 060</td>
<td>$#48$</td>
<td>0</td>
<td>80</td>
<td>50 120</td>
<td>$#80$</td>
<td>P</td>
<td>112</td>
<td>70 160</td>
<td>$#112$</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31 061</td>
<td>$#49$</td>
<td>1</td>
<td>81</td>
<td>51 121</td>
<td>$#81$</td>
<td>Q</td>
<td>113</td>
<td>71 161</td>
<td>$#113$</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32 062</td>
<td>$#50$</td>
<td>2</td>
<td>82</td>
<td>52 122</td>
<td>$#82$</td>
<td>R</td>
<td>114</td>
<td>72 162</td>
<td>$#114$</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33 063</td>
<td>$#51$</td>
<td>3</td>
<td>83</td>
<td>53 123</td>
<td>$#83$</td>
<td>S</td>
<td>115</td>
<td>73 163</td>
<td>$#115$</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34 064</td>
<td>$#52$</td>
<td>4</td>
<td>84</td>
<td>54 124</td>
<td>$#84$</td>
<td>T</td>
<td>116</td>
<td>74 164</td>
<td>$#116$</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35 065</td>
<td>$#53$</td>
<td>5</td>
<td>85</td>
<td>55 125</td>
<td>$#85$</td>
<td>U</td>
<td>117</td>
<td>75 165</td>
<td>$#117$</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36 066</td>
<td>$#54$</td>
<td>6</td>
<td>86</td>
<td>56 126</td>
<td>$#86$</td>
<td>V</td>
<td>118</td>
<td>76 166</td>
<td>$#118$</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37 067</td>
<td>$#55$</td>
<td>7</td>
<td>87</td>
<td>57 127</td>
<td>$#87$</td>
<td>W</td>
<td>119</td>
<td>77 167</td>
<td>$#119$</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38 070</td>
<td>$#56$</td>
<td>8</td>
<td>88</td>
<td>58 130</td>
<td>$#88$</td>
<td>X</td>
<td>120</td>
<td>78 170</td>
<td>$#120$</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39 071</td>
<td>$#57$</td>
<td>9</td>
<td>89</td>
<td>59 131</td>
<td>$#89$</td>
<td>Y</td>
<td>121</td>
<td>79 171</td>
<td>$#121$</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A 072</td>
<td>$#58$</td>
<td>:</td>
<td>90</td>
<td>5A 132</td>
<td>$#90$</td>
<td>Z</td>
<td>122</td>
<td>7A 172</td>
<td>$#122$</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B 073</td>
<td>$#59$</td>
<td>;</td>
<td>91</td>
<td>5B 133</td>
<td>$#91$</td>
<td>[</td>
<td>123</td>
<td>7B 173</td>
<td>$#123$</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C 074</td>
<td>$#60$</td>
<td>&lt;</td>
<td>92</td>
<td>5C 134</td>
<td>$#92$</td>
<td>\</td>
<td>124</td>
<td>7C 174</td>
<td>$#124$</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D 075</td>
<td>$#61$</td>
<td>=</td>
<td>93</td>
<td>5D 135</td>
<td>$#93$</td>
<td>]</td>
<td>125</td>
<td>7D 175</td>
<td>$#125$</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E 076</td>
<td>$#62$</td>
<td>&gt;</td>
<td>94</td>
<td>5E 136</td>
<td>$#94$</td>
<td>^</td>
<td>126</td>
<td>7E 176</td>
<td>$#126$</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F 077</td>
<td>$#63$</td>
<td>?</td>
<td>95</td>
<td>5F 137</td>
<td>$#95$</td>
<td>_</td>
<td>127</td>
<td>7F 177</td>
<td>$#127$</td>
<td>DEL</td>
</tr>
</tbody>
</table>
Organizing your files (cont.)

- Directories:

  - A directory is a special file (yep, a directory is a file) that can contain:
    - Information of the locations of files
    - Information of the locations of other directories
Organizing your files (cont.)

• Organizing files and directories:

  • File and directories are organized as a file system
  • Every modern file system is organized as a logical tree structure
The (logical) tree structure (cont.)

- What a tree structure in Computer Science look like:
The (logical) tree structure (cont.)

• Terminology:

  • The circles in the tree structure are called nodes
  • The top most node in the tree structure is called the root node
  • The lines in the tree structure are called branches or links