Lesson #1

Computer Systems and Program Development

Computer Systems

- Computers are electronic systems that can transmit, store, and manipulate information (data).
- Data can be numeric, character, graphic, and sound.
- For beginner programmers, the two most important are character and numeric.
- To manipulate data, a computer needs a set of instructions called a program.
- To write such programs is the object of this course.

Algorithms

- An algorithm is a series of instructions on how to solve the problem. We need algorithms before we can write programs. Algorithms have the following characteristics:
  - The order of execution of operations must be correct.
  - The operations must be clear and unambiguous.
  - The operations must be things that are feasible.
  - They must produce a result.
  - They must stop in a finite amount of time.

A Simple Algorithm

Baking Bread:

1. Add dry ingredients.
2. Mix.
3. Add water.
4. Knead.
5. Let rise.
6. Bake.

A More Complex Algorithm

Washing dishes:

1. Stack dishes by sink.
2. Fill sink with hot soapy water.
3. While there are more dishes:
   3.1 Get dish from counter,
   3.2 Wash dish,
   3.3 Put dish in drain rack.
4. Wipe off counter.
5. Rinse out sink.

Another Complex Algorithm

Sorting mail:

1. Get piece of mail from mail box.
2. If piece is personal,
    2.1 Read it.

else
    if piece is a magazine,
        2.1.1 Put in magazine rack.
    else
        if piece is a bill,
            2.1.1.1 Pay it.
        else
        if piece is junk mail:
                2.1.1.1.1 Throw it away.

Another Algorithm

Directions from Toronto to Port Dover:

1. Take the Gardiner Expressway West.
2. Continue onto Queen Elizabeth Way West.
3. Continue onto Highway 403 West.
4. Exit onto Highway 6 South toward Caledonia/Port Dover.
5. Turn right to stay on Highway 6 South (signs for Caledonia/Port Dover).
6. Turn right at Greens Rd / Highway 6 South.
7. Turn left at Caledonia Bypass / Highway 6 South.
8. Continue to follow Highway 6 South.

About Algorithms

* Algorithms are a set of instructions for solving a problem.

* Once you have created an algorithm, you no longer need to understand the principles on which the algorithm is based.

* For example, once you have the directions to Port Dover, you don't need a map any more. The information needed to find the correct route is contained in the algorithm. All you have to do is follow the directions.

About Algorithms

* Algorithms are a way of sharing expertise with others.

* Once you have encoded the necessary intelligence to solve a problem in an algorithm, many people can use your algorithm without needing to become experts in a particular field.

* Algorithms are especially important to computers because computers are really machines for solving problems.

* For a computer to be useful, we must give it a problem to solve and a technique for solving the problem.

About Algorithms

* You can make computers "intelligent" by programming them with various algorithms to solve problems.

* Because of their speed and accuracy, computers are well suited for solving tedious problems such as searching for a name in a large telephone directory.

* Not all problems can be solved with computers because the solutions to some problems can't be stated in an algorithm.

Writing an Algorithm

* We will now learn how to write an algorithm to solve a simple problem:

* Sort the following numbers in ascending order:

7 2 8 3 5

* Think about how you would solve this problem for a moment.

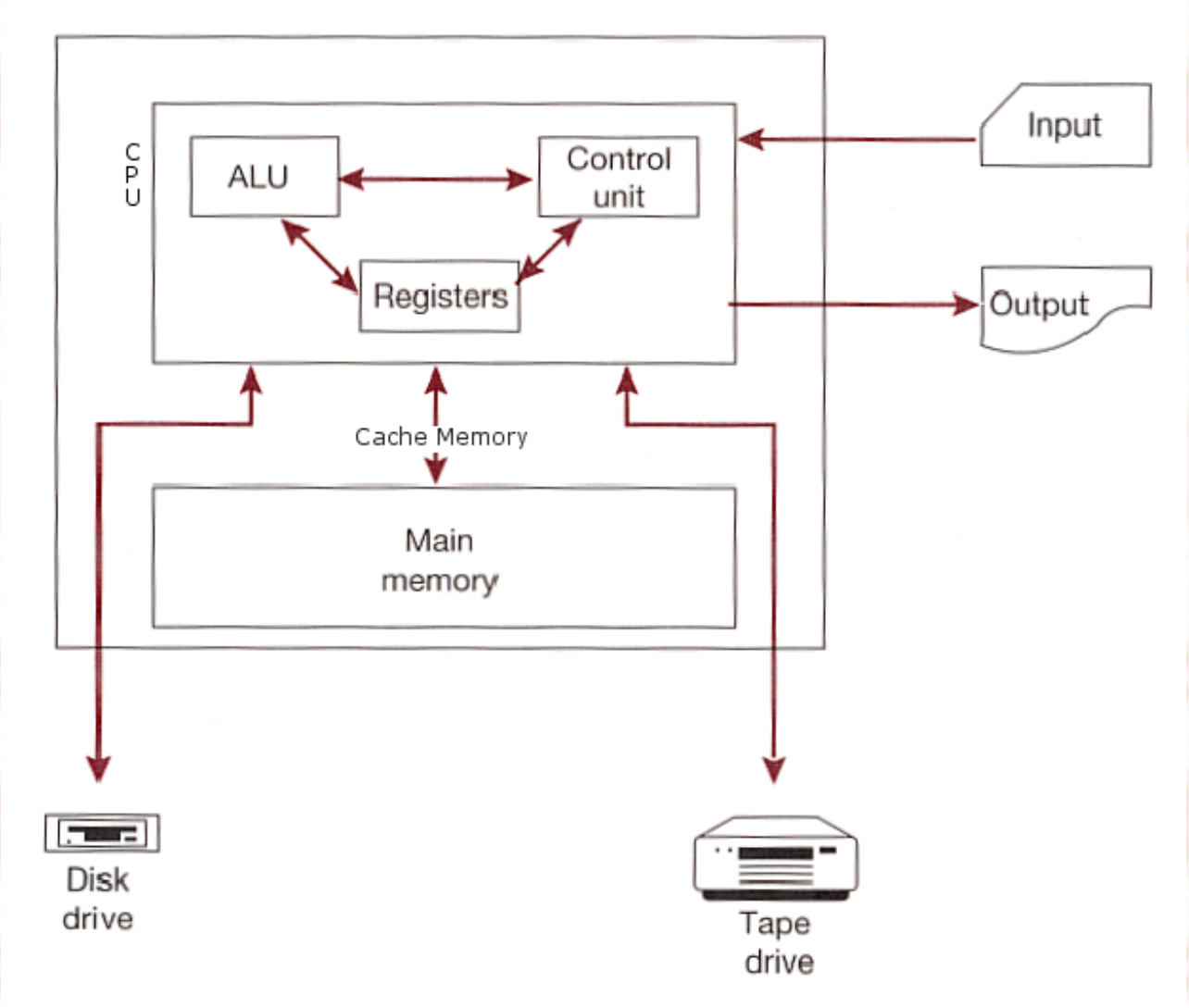
* Sorting is a common problem handled by computers.

Computer Systems

Types of computer systems:

- Mainframes and minicomputers
- Workstations
- Desktop computers
- Laptop or notebook computers
- Tablets
- Palmtop Computers or Personal Digital Assistants (PDA)

Components of a Computer



Storage Types

- CPU registers: Only a few cells on CPU.
- Main memory (RAM): Billions of cells on circuits separate from CPU.
- Secondary storage: Hundreds and thousands of billions of cells on disks or tapes.
- Secondary storage is not volatile (data is kept even when power is off).

Internal Representations

Bit: Binary digit (0 or 1).

Byte: A group of 8 bits. One character.

Word: The width of a memory cell.

Each byte of main memory has an address.

All numbers are represented in binary code.

Binary System

- To count in binary is similar to the usual count in decimal: You start at 0 and at 9 you run out of digits so you use two instead like 10. In binary we run out of digits at 1 instead of 9. So we count in binary from 0 to 10 decimal with 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, and 1010.

- Like in decimal, we can pad with zeros to the left without changing the value: 00001001 is still a binary equivalent of 9.

Integer Numbers (Unsigned)

All numbers are converted in binary:

ex: 9 = 1001

- So 9, the integer in a 32-bit system, would look like:
  (the spaces between the groups of 4 bits are only there for clarity)
  0000 0000 0000 0000 0000 0000 0000 1001
- (Always place the number on the right and pad the unused bits with zeros)
- Can you find the decimal value of this number?
  0000 0000 0000 0000 0000 0000 0010 1101

Integer Numbers (Signed)

- There are many ways to represent negative integers. The simplest method is called **sign-and-magnitude**. It means using the leftmost bit as a sign bit (1 for negative, 0 for positive). In a 32-bit system that means that the first bit represents the sign, and the other 31 the absolute value of the number.
- For example –9 will be:
  1000 0000 0000 0000 0000 0000 0000 1001
- This method has one major drawback, it allows for two zero values –0, and +0.

```
+0: 0000 0000 0000 0000 0000 0000 0000 0000
–0: 1000 0000 0000 0000 0000 0000 0000 0000
```

Integer Numbers (Signed)

- Without a doubt, your personal computer uses a method known as **two's complement**. A little more complicated than sign-and-magnitude but not that much and it has only one zero value.

- To have –9 in two's complement, you invert all the bits in the representation of 9 and add 1.

- You can achieve the same result by starting from the right; find the first 1 and invert all the bits to the left of that one.

```
+9 = 0000 0000 0000 0000 0000 0000 0000 1001
-9 = 1111 1111 1111 1111 1111 1111 1111 0111
```

Hexadecimal Numbers (HEX)

- In computer science, the base-16 numeral system is often used to represent groups of 4 bits. It uses 16 distinct symbols, most often the symbols 0–9 to represent values zero to nine, and A, B, C, D, E, F to represent values ten to fifteen.

Example:

+9 = 0000 0000 0000 0000 0000 0000 0000 1001

0 0 0 0 0 0 0 9

-9 = 1111 1111 1111 1111 1111 1111 1111 0111

F F F F F F F 7

Real (Floating Point) Numbers

- Real numbers in binary are expressed with a **sign**, a **mantissa** (or fraction or significand), and an **exponent**.
- The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely used standard for floating-point computation
- The **exponent** is biased depending on the format (127 for single precision, 1023 for double).

Real (Floating Point) Numbers

Suppose +9.0 in binary:

- The sign is positive. It will be represented by a sign-and magnitude convention (**0**) at the leftmost bit.
- The exponent will be 3 (the closest power of 2 less or equal than 9 is 8, which is 2 to the power of 3), then biased (3+127=**130** for single, 3+1023=**1026** for double). That number is then converted to an unsigned binary integer to fit into the 8-bit exponent zone (11-bit zone for double precision).
- To compute the mantissa you take the number (9.0) and divide it by 2 to the power of the exponent (in this case 8). You get 1.125. The mantissa is then filled from left to right with 1s and 0s. The first bit represents 0.5 (1 is assumed but not represented), the second bit 0.25, ... For 1.125 we need 1+0.125 which is the third bit at 1, all the others remaining at 0.

Real (Floating Point) Numbers

9.0 in single-precision binary is:

```
0100 0001 0001 0000 0000 0000 0000 0000
```

A double number is expressed in 64 bits:
52 bits for the mantissa, 11 bits for the exponent, and 1 sign bit.

So 9.0 double-precision binary is:

```
0100 0000 0010 0010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

See a IEEE-754 Floating-Point Conversion calculator at
http://babbage.cs.qc.edu/IEEE-754/Decimal.html

Real (Floating Point) Numbers

- Now let's do the reverse. Let's convert a binary IEEE-754 floating-point number into a decimal.

1100 0011 0**110** 0000 0000 0000 0000 0000

1. Determine the sign: That one is easy. 1 is negative, 0 is positive. We have **negative** number.

2. Determine the exponent: 10000110 is 128+4+2=134. That our biased exponent. Subtract the bias (127) to find the true exponent: 7. Finally, let's calculate 2 to the power of 7, we get **128**. This means our number is between 128 and 255.

3. Determine the mantissa: The first bit represents 0.5, the second 0.25, the third 0.125.... Here we have the two first bits at one: 0.5+0.25 = 0.75. Since 1 is always assumed, our mantissa value is **1.75**.

4. Do the final computation: Multiply the final value of step #2 by the final value of step #3. 128 times 1.75 is 224. So our number is

–224.0

Characters

Characters are expressed using the ASCII code:

'A' = 65 = 01000001
'g' = 103 = 01100111
'$' = 36 = 00100100
'+' = 43 = 00101011

Note that digits expressed in ASCII are different than integers or doubles.

'9' = 57 = 00111001

See ASCII code at **www.asciitable.com**

Programming Languages

Many languages exist to program instructions to the computer. C is just one of them.

Generation 1: Machine languages (pure binary)
10101110101010101011101010101011

Generation 2: Assembly languages (mnemonic codes)
MV R1,R3

Generation 3: High-level languages (C, Fortran, Java)

Solving Problems

- Here are the steps required to solve a problem with a computer program:

1. Define the problem
2. Analyze the problem.
3. Design a solution.
4. Implement the solution.
5. Test the program.
6. Update and maintain the program.

Implementation

- Here is a detail of step #4, implementation (actual programming):

4.1 Write the program source.

4.2 Compile the source code and check for errors.

4.3 Build the program (links your program to the libraries and creates the executable file).

4.4 Run the program.

Why C?

1. It is portable.
2. It is efficient.
3. It is easy to learn.
4. It is modular.
5. It is widespread.

End of Lesson