Basic Computer Structure

A computer is composed of wires and switches.

Data is transmitted down wires and each wire can have two different values: on (has current) or off (no current).

So a wire transmits two different values: 1 (on) or 0 (off).

- If we put 2 wires together, we get 4 different values: both off (00), both on (11), the first on and the second off (10), the first off and the second on (01).
- If we put 3 wires together, we get 8 different values: 000, 001, 010, 011, 100, 101, 110, 111. (Note that 8 = 2^3.)
- If we put 8 wires together, we get 2^8 = 256 different values.

Definition: A bit is the smallest unit of information. A bit is two values, usually denoted as 0 and 1.

A byte is 8 bits. A byte has 2^8 = 256 different values.

The main processing in a computer is done in the ALU (Arithmetic-Logic Unit) that is a part of the CPU (Central Processing Unit).

The ALU is made up of separate circuits to perform different tasks like addition, multiplication, etc.

The ALU has three sets of input wires and one set of output wires.

- Two of the sets of input wires are for the data to the ALU (for example, if you are going to add two numbers, one number arrives in the first set of wires, and the other number on the second set).
- The third input set of wires controls which operation the ALU should perform.

x-bit Machines

We sometimes refer to machines as x-bit machines.

For example, your computer was probably advertised as a 64-bit machine (newer ones) or a 32-bit machine (older ones).

The number refers to how many wires go into the computer circuits, and that means how much data the computer can process in one step.

For example, in a 32-bit machine's each set of input wires to the ALU will consist of 32 wires and the output will also consist of 32 wires.

32 wires together is 32 bits of information, and that means each input can be one of 2^32 different values.

As a result, there is a limit to how many different values a computer can add (or subtract or multiply, etc) directly on the chip.

If we need to do arithmetic on larger numbers, we need to write programs for it.

With a 64-bit machine, we get a lot more values (2^64 different values), but the same idea holds. There are a limited number of values we can manipulate directly on the chip. Anything more requires a program.

How a computer does a single operation:

The input wires for the ALU are turned on/off depending on the desired input values.

The control wires for the ALU are turned on/off depending on the operation being done (addition, multiplication, etc.)

As the electricity from these wires passes through the ALU circuits, different switches are activated to perform the calculation, and the resulting output appears as the output wires turn on/off appropriately.

The control unit has a crystal based clock (similar to what runs a wrist watch), and when enough time passes for the calculation to complete, the control unit takes the output values and stores them into an appropriate location of memory.

The speed of calculation is determined by how long it takes the electric current of the input/control wires to traverse the ALU and reach the output. Modern computers are so fast because these circuits are incredibly tiny.

(Just for fun, your computer can probably add two integers in less than 1 billionth of a second. How far does light travel in 1 billionth of a second?)

Key idea to remember:

All data on a computer is transferred by a group of wires. Therefore all data (whether an integer, a fraction, a music file, etc.) is just a collection of 0's and 1's (is the wire off or on?)

Even the signals that control the ALU are sent on groups of wires, and so those signals are 0's and 1's.

We can think of the 0's and 1's as a number (in fact, as an integer) represented in base 2.

Everything in a digital computer: the data, the commands that control the computer, the program, everything is just numbers.

The computer does not know about music, pictures, games. It only sees numbers represented in binary and all it can do is simple math operations like addition, multiplication, etc.

It is up to us, the human programmers, to give meaning to these numbers.

Data Types:

The type of a piece of data is what the piece of data represents. It is specified by the programmer and by rules of the programming language.

Java's Strict Typing:

Java is a "strongly typed" language. That means that every piece of data will have a well defined type and unambiguous type and the compiler will verify that every type is used correctly.

The programmer must set the type of every expression (either explicitly or implicitly), and the Java compiler will verify that each type is used correctly before you can run your program. Java is very strict. If a type is used incorrectly, an error is given.

For example, if you specify that a piece of data is English text, then Java will only allow you to do actions appropriate for text on it.

(Java was built to replace C/C++ which has types, but does not have strict safety restrictions.)

There are two kinds of types in Java:

1. **primitive types** : there are 8 primitive types, and they are all pre-defined in the Java language

2. **compound types (called reference types in your book) :** there is a potentially infinite number of such types; many are pre-defined in Java, but programmers can make new ones as needed

Primitive Types in Java: The following are the eight primitive types:

int: 32-bits of data. Represents an integer between -2^31 and +2^31 – 1

All whole numbers in your program default to the type int. **example:** 5, -30, 0 are all ints

double: 64-bits of data. Represents a "floating point" number (a number with a decimal point) in scientific notation. Roughly 52 bits for the mantissa, 11 bits for the exponent, 1 sign bit. All numbers with decimal points or in scientific notation in your program default to the type double. The largest/smallest values that can be represented are +/- 4.9 x 10^-324 and +/- 1.7 x 10^308.

example: 3.1415, -0.3, 10., 3e12, 23E-104 are all doubles

(The e/E is for scientific notation. The numbers above are 3x10^(12) and 23x10^(-104). It does not matter if you use e or E.)

char: 16-bits of data. Represents a character.

char values are single characters inside single quotes. **example:** 'a', 'x', ' ', '?', '3' are all char values.

(Note that '3' and 3 are not the same thing! The first is a binary value that Java interprets as the character 3, and the second is a (different) binary value that Java interprets as the whole number 3.)

boolean: 1-bit of data (really stored as 8-bits). Represents either true or false.

We will use int, double, char, and boolean in projects and tests. Here are the other 4 primitive types for your reference (they could show up in quizzes or labs):

short: 16-bits of data. Represents an integer between -2^15 and +2^15 - 1.

byte: 8-bits of data. Represents an integer between -2^7 and +2^7 - 1.

long: 64-bits of data. Represents an integer between -2^63 and +2^63 - 1.

To enter a long value, you use the letter L (upper or lower case) **example:** 10L, 300000L are all long values

float: 32-bits of data. Binary, scientific notation. Roughly 23 bits for the mantissa, 8 bits for the exponent, 1 sign bit. The largest/smallest values that can be represented by floats are +/- 1.4 x 10^-45 and +/- 3.4 x 10^38

To enter a float, you use F (upper or lower case) **example:** 3.14F is a float value

Please note that the number of bits of the type determines how many different values it can store. Even though a long stores numbers between -2^63 and +2^63, and double stores values between +/- 4.9 x 10^-324 and +/- 1.7 x 10^308, the double stores the same number of different values as the long does. The long can store every integer in that range, the double stores only some of the possible floating point numbers in that range. Thus double cannot store all the integer values long can.

Typecasts (type conversions)

Java allows the programmer to convert a value of one type into a value of a different type.

To do this, you place the desired type in parentheses and immediately to the left of the value.

(desired-type) value

For example:

(double)3 converts the int value 3 to the double value that is the closest to 3. In this case, the value is 3.0.

(short)100 converts the int value 100 to the short value that is the closts to 100. In this case, the value is still 100 (but stored in 16 bits instead of 32 bits).

(int)3.8 coverts the double value 3.8 to the int value 3

The typecast must "make sense". In Java, you can convert between all the numeric primitive types, but not between boolean and a numeric primitive so:

(int)false <= error, you can't convert a boolean to a numbers

(int)'A' <= this is legal. Java considers a char to also be a number

Introduction to Variables

A variable is the name given to a location in memory. In Java, variable has a type associated with it. A variable is used to store data. In Java, you can only store values with the appropriate type into the variable.

Creating Variables: We call creating a variable "declaring the variable"

A declaration has the form: type variable-name

Two examples:

int x

double temperature

Java now sets aside a chunk of memory for each varialble. The first is a 32-bit chunk that is associated with the name "x". Java will make sure that only values of type int can be stored in here. (Remember that the type is what WE decide the value represents. The computer does not care. It is happy storing any 32-bit values there.)

The second is a 64-bit chunk of memory that is associated with the name " temperature ", and we can store values of type long in it. Storing Values in Variables. We call storing a value "**assigning a variable**".

To store a value in the variable, you use the = operator.

variable = value

For example:

x = 161

temperature = 40.75

This is a source of confusion because the assignment operator looks like math's equality, but it is not an equality test. Instead, we are storing into the memory location named "x" the data that represents the int 161, and we are storing into the memory that is named "temperature" the data that represents the double value 40.75.

Remember that Java is a strong typed language so

x = 3.1415 <= error, you promised x would store int, and you are giving it a double

x = (int)3.1415 <= this is now legal, and it stores value 3 into the location named "x"

To read the value stored in memory at a location, just use the name of the location:

x <= this evaluates to whatever value is stored in the memory location in "x", interpreted as data type int

int y

y = x + 50 <= this gets what is stored in x, adds 50 to it, and stores the result in the variable y

Automatic Typecasts (called "type coercions")

Values are automatically converted "narrower" to "wider" types. A conversion from a "wider" to a "narrower" type requires an explicit typecast. Generally, when converting to a wider type, Java converts the value to be as close as possible to the original value. When converting to a narrower type, Java generally truncates the value.

Widest: double, float, long, int, short/char, byte : **Narrowest**

Note that short and char are at the "same" level, and so you must explicitly typecast between these two types. Also note that boolean is not in this list.

You can not convert a value of type boolean to any other primitive.

You can not convert another primitive value to boolean.

int y

y = 3.0 <- This is illegal because 3.0 is type double, double is wider than int, and we need an explicity typecast

y = (int)3.0 <- This is legal because the typecast is explicit.

y = 'A' <- This is legal because 'A' is type char, char is narrower than int, so the 'A' will be automatically converted to type int