Programming Essentials

Topics

1. What is an Algorithm?
2. Interpreters vs Compilers
3. Hardware Operations
4. Computing Power
5. Time and Space Limits for Programs
6. Process Memory Layout
7. Bits/Bytes
8. 32bit vs 64bit architecture
9. One's Complement and Two's Complement
10. Important References

What is an algorithm?

- "An algorithm is the step-by-step instructions to solve a problem".
- "An algorithm is a well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transforms the input into the output".

Example: We might want to sort a sequence of numbers in a non-decreasing order.
Input: Sequence of *n* numbers $(a_1, a_2, a_3, \ldots, a_n)$
Output: A permutation of the input sequence such that $b_1 \leq b_2 \leq b_3 \leq \ldots \leq b_n$

Interpreters vs Compilers

We generally write a computer program using a high-level language. A high-level language is one which is understandable by us humans. It contains words and phrases from English (or other) language. But a computer does not understand high-level language. It only understands programs written using binary code (0's & 1's), aka the *machine code*. A program written in high-level language is called a *source code*. We need to convert the source code into machine code and this is accomplished my **compilers** and **interpreters**. Hence, a compiler or an interpreter is a program that converts program written in high-level language into machine code, which can be understood by the computer.

The difference between an interpreter and a compiler is given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence, Interpreters are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Interprets the same program statements each time it meets them eg: instructions within a loop.	The generation of machine code happens only once.
Example: Programming language like Python, Ruby uses interpreters.	Example: Programming language like C, C++ uses compilers.

Quora answer on Assembler, Interpreter, & Compiler

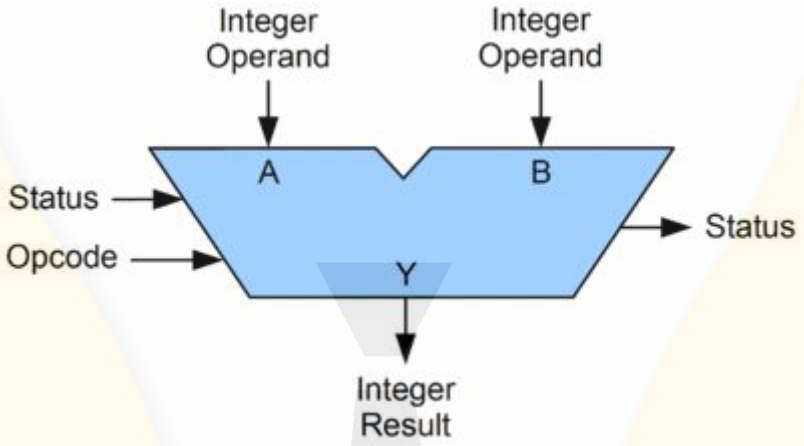
Java's Execution Environment:

1. **Is the JVM a compiler or an interpreter?**
2. **Why do we say that Java is both compiled and an interpreted language?**

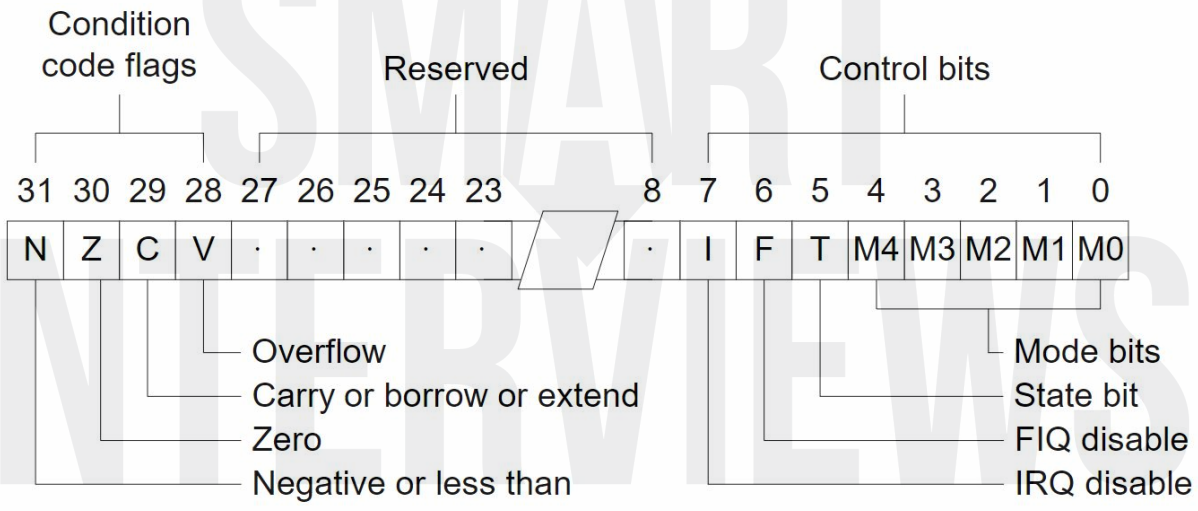
Hardware Operations

An arithmetic-logic unit (ALU) is the part of a computer processor (CPU) that carries out arithmetic and logic operations on the operands in computer **instruction** words. In some processors, the ALU is divided into two units, an arithmetic unit (AU) and a logic unit (LU). Some processors contain more than one AU - for example, one for fixed-point operations and another for floating-point operations.

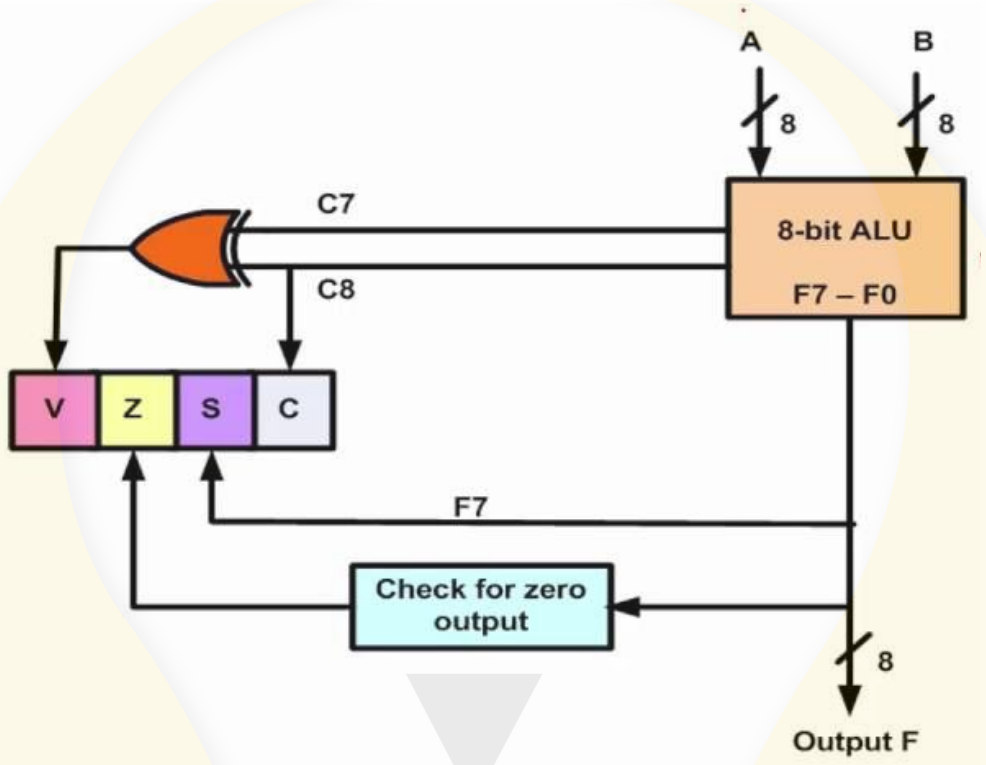
An ALU is a fundamental building block of many types of computing circuits, including the **Central Processing Unit** (CPU) of computers, **Floating Point Unit** (FPUs), and **Graphics Processing Units** (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.



Opcode - The opcode input is a parallel bus that conveys to the ALU an operation selection code, which is an enumerated value that specifies the desired arithmetic or logic operation to be performed by the ALU.
Status - The status outputs are various individual signals that convey supplemental information about the result of an ALU operation. Ex: Carry-out (C), Zero (Z), Negative/Sign (N/S), Overflow (V), etc. It is sometimes known as Program Status Word [PSW].



PSW bits example in a 32-bit system. We can see that the 4 MSBs are referring to the status of the Instruction that ran through ALU.



Circuit Diagram to generate the PSW bits

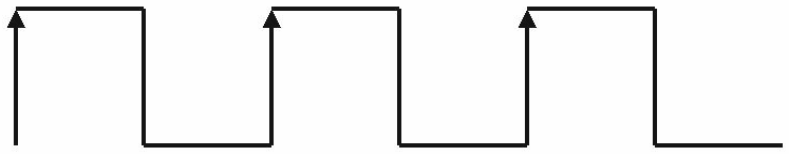
More Details:

1. **What is an Instruction?**
2. **More on Instruction**
3. **Definition of ALU**
4. **A short crash course on ALU**
5. **NPTEL's Computer Architecture Lecture on ALU**
6. **More on ALU**

Also, most processors do not have a hardware circuit especially made for modulo operation, many even do not have a hardware circuit for division operation. Now, if we have burned the hardware circuit to apply division operation, we might be willing to go that extra mile to add the hardware circuit for modulo operation as well. Most processors take our question to the next level, why would we implement a division operation using a separate hardware circuit when it can be done using clever software implementation. The answer to our question is, most processor families do not have a hardware circuit for both division and modulo operation, because it is not worth the chip's real estate, power consumed, etc compared to the software solution. The software solution is less painful/costly/risky.

Computing Power

Clock is a signal used to sync things inside the computer. Take a look at the figure below, where we show a typical clock signal: it is a square wave changing from "0" to "1" at a fixed rate. In this figure we can see three full clock cycles ("ticks"). The beginning of each cycle is when the clock signal goes from "0" to "1"; we marked this with an arrow. The clock signal is measured in a unit called Hertz (Hz), which is the number of clock cycles per second. A clock of 100 MHz means that in one second there are 100 million clock cycles or 10^8 clock cycles.



Example of a Clock Pulse

Another way to define it is to say that clock cycle is the amount of time between two pulses of an oscillator. Generally speaking, the higher the number of pulses per second, the faster the computer processor will be able to process information. CPU's clock speed, or clock rate, is measured in Hertz — generally in Gigahertz, or GHz. A CPU's clock speed rate is a measure of how many clock cycles a CPU can perform per second. For example, a CPU with a clock rate of 1.8 GHz can perform 1,800,000,000 clock cycles per second.

This seems simple on its face. The more clock cycles a CPU can perform the more things it can get done, right? Well, yes and no.

This may not seem obvious at first, but it's actually for a very simple reason. It's because the speed of a computer is also influenced by other factors, such as the efficiency of the processor, the bus architecture, the amount of memory available, and the software that is running on the computer, the instruction set being used by the computer, etc. Some processors can perform more instructions per clock cycle than other processors, making them more efficient than other processors with higher clock speeds. All other things being equal, fewer clock cycles mean the CPU requires less power and produces less heat.

In addition, modern processors also have other improvements that allow them to perform faster. This includes additional CPU cores and larger amounts of CPU cache memory that the CPU can work with. For example, if the cache is not sufficient the computer will wait until the instruction or the data will be retrieved from the RAM, and since the RAM is a slower device compared to cache memory, the data fetch takes more time, and therefore, overall processing time increases due to this.

The Random Access Memory (RAM) module inside a computer, stores temporary data for the Central Processing Unit. Data that cannot be stored inside the RAM must be accessed from the hard drive, thereby, slowing down the processing of the machine. More RAM implies more programs and larger files can be used simultaneously without any impact on the overall performance of the system. RAM is important because it eliminates the need to "swap" programs in and out of the memory, thereby not leading to **thrashing**.

More Details:

1. **Video on Triggering (Edge Triggers & Level Triggers) in a Clock Pulse**
2. **Video on Clock Triggering**
3. **What does Edge Triggered & Level Triggered mean?**

Time and Space Limits for Programs

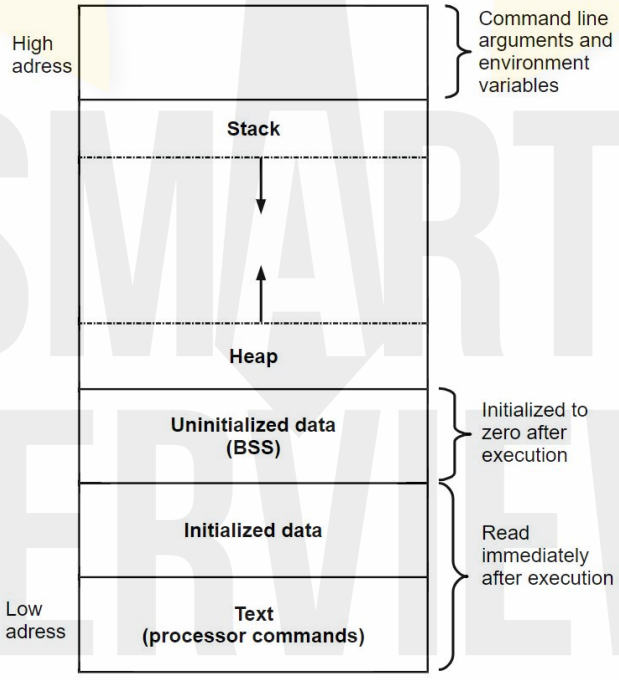
Different algorithms have different constant factors. Some operations are slow - modulo (slows down the program significantly), if-else conditions, standard minimum/maximum functions, etc., while others, like bitwise operations, are really fast and a good way to optimize a program (they can be used for faster min/max functions).

Then, there are small tricks like adding a condition which for most test data makes the code skip some redundant operations, or noticing that it's hard to make test data for which some assumption doesn't hold. But that's another topic! The point is that, sometimes, these tricks (they can be in our code even if we don't know about it) can improve our constant factor, or even complexity, noticeably.

In short, it all depends on experience. When we code more algorithms and note how fast they ran, we'll be able to estimate better what could pass later, even without coding it.

- Approximately we can perform about **10^8 operations in one second**. So with that we can estimate whether our algorithm will run in time or not.
- Approximately our program can consume 10^7 bytes of memory (≈ 10MB). From there on it goes on to allocate memory on our virtual memory.

Process Memory Layout



A typical memory layout of a running process

1. **Text Segment:**
   - Sections of a program in memory which contains Executable Instructions.
   - As a memory region, a text segment may be placed below the heap or stack in order to prevent heap and stack overflows from overwriting it.

2. **Initialized Data Segment:**
   - Contains the global variables and static variables that are initialized by the programmer.
   - This segment can be further classified into initialized read-only area and initialized read-write area.

3. **Uninitialized Data Segment:**
   - Often called the "BSS" segment, named after an ancient assembler operator that stood for "Block Started by Symbol."
   - Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing.
   - Contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

4. **Stack:**
   - The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory.
   - A "**stack pointer**" [SP] register tracks the top of the stack; it is adjusted each time a value is "pushed" onto the stack area.
   - The set of values pushed for one **function call** is termed as a "**stack frame**". A stack frame consists at minimum, a return address.
   - Each time a function is called, the address of where to return to and certain information about the caller's environment, such as some of the machine registers, are saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables. This is how function calls or recursive functions in any language work.

5. **Heap:**
   - Heap is the segment where dynamic memory allocation usually takes place.
   - The heap area begins at the end of the BSS segment and grows to larger addresses.
   - The Heap area is managed by `malloc()`, `realloc()`, and `free()`.
     example: `ptr = (int*) malloc(100 * sizeof(int));`
   - The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

Bits/Bytes

- A **BIT** (short for **BI**nary digi**T**) is the smallest unit of data in a computer. A bit can have a binary value of either 0 or 1. Binary means that there are only two logical (*i.e.* on/off, true/false) choices. Until the value is actually determined, it can have two states.

- A Byte is composed of 8 bits. Half a Byte (4 bits) is called a **Nibble**.

8 bits	=	1 Byte
1,024 Bytes	=	1 KiloByte
1,024 KB	=	1 MegaByte
1,024 MB	=	1 GigaByte
1,024 GB	=	1 TeraByte
1,024 TB	=	1 PetaByte
1,024 PB	=	1 ExaByte
1,024 EB	=	1 ZetaByte
1,024 ZB	=	1 YottaByte

- Note: We see that internet speeds are often advertised with the usage of 15 Mbps, 25 Mbps, 100 Mbps, etc. Mbps is "Megabits per second", and not "Megabytes per second". In terms of acronyms, Mb and MB have two different meanings, *i.e.*, Mb is Megabit, which is, 10^6 bits = 125 KB, and MB is 10^6 Bytes = 1 MB (or) 1024 KB.

32bit vs 64bit architecture

- The key difference: 32-bit processors are perfectly capable of handling a limited amount of RAM, and 64-bit processors are capable of utilizing much more.
- As a general rule, if we have less than 4 GB of RAM in our computer, we don't need a 64-bit CPU, but if we have more than 4 GB RAM, we do.
- More bits mean that our system can point to or address a larger number of locations in physical memory.
- Although, theoretically, 64-bit CPUs can fit in a RAM of 2^{64} Bytes, which is 16 Exabytes, because using 64-bits, the CPU can uniquely address 2^{64} addresses. But, because the hardware is still unable to catch up, the commercially available RAM that fits into a 64-bit CPU is a maximum of 128 GB. Non-commercially, the highest RAM is 3 TB used by HP DL380 Generation 9.
- A 32-bit CPU has the pointer size as 4 Bytes, and a 64-bit CPU has the pointer size as 8 Bytes. This is because, a pointer basically holds the address of a variable, and the variable has an address in the memory which is uniquely identifiable. In a 32-bit system, each instruction/address is 32 bits, which is 4 Bytes, and in a 64-bit system, each instruction/address is 64 bits, which is 8 Bytes, therefore, the pointer size in a 32-bit & 64-bit system are justifiable as 4B & 8B respectively.
- Although, theoretically, a 32-bit system can address only 2^{32} unique addresses, which means that the system can only fit upto 4GB of RAM. But practically, we can use 64GB RAM in a 32-bit CPU, using a technique called **Physical Address Extension** [PAE]. Using PAE, theoretically, we can extend the maximum RAM that can be fit into a 32-bit CPU from 4GB to 4PB.

One's Complement and Two's Complement

- If all bits in a byte are inverted, by changing each 1 to 0 and each 0 to 1, we have formed the one's complement of the number. To that one's complement number, if we do a binary addition with $(1)_2$, then, the resulting binary number is known as the two's complement of the original number.
- Two's Complement is used in place of one's complement, because of a primary reason, which is, in one's complement, there are two representations of 0, i.e., there's a binary representation for -ve 0, and a +ve 0, therefore, there are two binary representations for 0 in one's complement. Another reason is, in one's complement, the addition of 2 one's complement numbers is not intuitive and is a difficult task.
- Two's complement is a more intuitive way of representing negative numbers.

  Example:

  $(65)_{10}$ -> $(01000001)_2$

  $(-65)_{10}$ -> $(10111111)_2$ [2's complement of 65]
- With a system like two's complement, the circuitry for addition and subtraction can be unified, whereas otherwise they would have to be treated as separate operations.

Tips:

Powers of 2 should be at the tip of our fingers:

$2^0 = 1$;	$2^1 = 2$;	$2^2 = 4$;	$2^3 = 8$;	$2^4 = 16$;	$2^5 = 32$;
$2^6 = 64$;	$2^7 = 128$;	$2^8 = 256$;	$2^9 = 512$;	$2^{10} = 1024$;	
$2^{10} \approx 10^3$;	$2^{20} \approx 10^6$;	$2^{30} \approx 10^9$;	$2^{40} \approx 10^{12}$;	$2^{50} \approx 10^{15}$;	$2^{60} \approx 10^{18}$;

Important References

1. Notes on The **Von Neumann Computer Model**.
2. Tutorial on **Data Representation**: Integers, Floating Point Numbers, & Characters.
3. What is **Booting**?
4. How can a CPU deliver **more than one instruction per cycle**?
5. **Memory Layout** of a C Program.
6. **Storage classes, scope & memory allocation** in C.
7. Cornell University Notes on **Two's Complement**.
8. What do **x86_64, i386, ia64** and other such jargons stand for?
9. **x86 vs x64** - Why is 32-bit called x86?
10. **32-bit vs 64-bit**.
11. What is **x86-64**?
12. What is the **difference between 32-bit and 64-bit CPUs**? How does the **performance increase for 64-bit CPUs**?
13. **long long int on 32-bit CPUs**.
14. What is **Physical Address Extension**?
15. **Superscalar Processors**.
16. What is an **Instruction Pipeline**?
17. What is **Hyper-threading**?
18. What is **difference between Multiprogramming, Multitasking, Multithreading and Multiprocessing**?
19. What is the **difference between a multiprocessor system and a multicore processor system**?
20. **Computer Organisation & Architecture** [Simplified Video Lectures] [Must Watch]