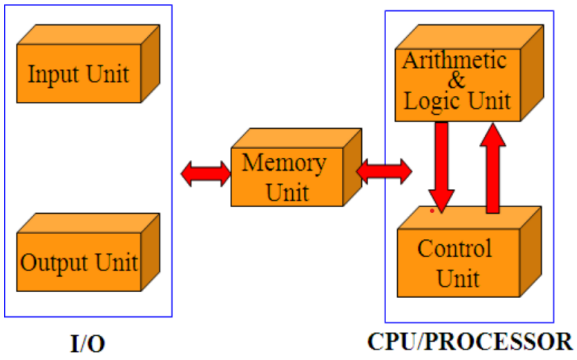
Structure of computer and its functional units

A computer in its simplest form comprises five functional units namely input unit, output unit memory unit, arithmetic & logic unit and control unit. Figure 2 depicts the functional units of a computer system.



1. Input Unit: Computer accepts encoded information through input unit. The standard input device is a keyboard. Whenever a key is pressed, keyboard controller sends the code to CPU/Memory.

Examples include Mouse, Joystick, Tracker ball, Light pen, Digitizer, Scanner etc.

2. Memory Unit: Memory unit stores the program instructions (Code), data and results of computations etc. Memory unit is classified as:

i) Primary /Main Memory

ii) Secondary /Auxiliary Memory

Primary memory is a semiconductor memory that provides access at high speed. Run time program instructions and operands are stored in the main memory. Main memory is classified again as ROM and RAM. ROM holds system programs and firmware routines such as BIOS, POST, I/O Drivers that are essential to manage the hardware of a computer. RAM is termed as Read/Write memory or user memory that holds run time program instruction and data. While primary storage is essential, it is volatile in nature and expensive. Additional requirement of memory could be supplied as auxiliary memory at cheaper cost. Secondary memories are non-volatile in nature.

3. Arithmetic and logic unit: ALU consist of necessary logic circuits like adder, comparator etc., to perform operations of addition, multiplication, comparison of two numbers etc.

4. Output Unit: Computer after computation returns the computed results, error messages, etc. via output unit. The standard output device is a video monitor, LCD/TFT monitor. Other output devices are printers, plotters etc.

5. Control Unit: Control unit co-ordinates activities of all units by issuing control signals. Control signals issued by control unit govern the data transfers and then appropriate operations take place. Control unit interprets or decides the operation/action to be performed

The operations of a computer can be summarized as follows:

1. A set of instructions called a program reside in the main memory of computer.

2. The CPU fetches those instructions sequentially one-by-one from the main memory, decodes them and performs the specified operation on associated data operands in ALU.

3. Processed data and results will be displayed on an output unit.

4. All activities pertaining to processing and data movement inside the computer machine are governed by control unit.

What do you mean by program control instructions? With a neat diagram, explain how the status register containing overflow, zero, sign and carry flags works with the status of the accumulator content obtained from CPU.

Program control type instructions, when executed by the processor, may change the address value of the Program Counter and cause type flow of control to be altered. Program control instruction specifies conditions for altering the content of Program Counter. This causes break in the sequence of instruction execution. This Instruction also gives the capability for branching to different Program segments. Examples - Branch., Jump, Skip, Call, Return etc. Status bits are set or reset depending on the result of a logical or arithmetic manipulation of accumulator data. So status bits are called condition - code bits or flag bits. These status bits constitute status register.

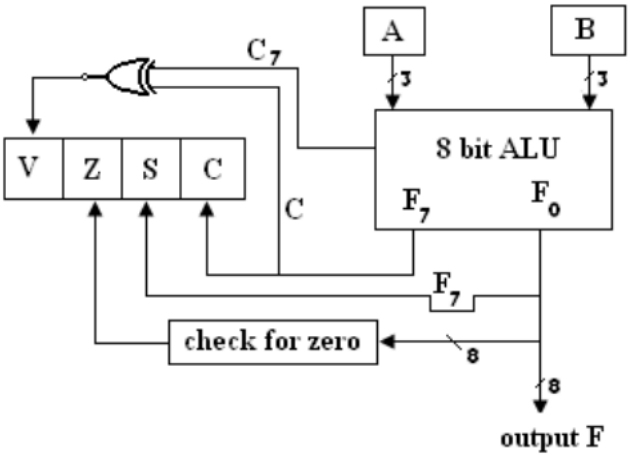
The hardware realization of status register containing overflow, Zero, Sign, Carry flag is shown below

(i) Bit 'c' (carry) is set to 1. if the end carry c8 is 1. It is cleared to zero if the carry is 0

(ii) Bit's' (sign) is set to one if' the highest order bit F7 is 1. It is set zero, if the bitF7=0

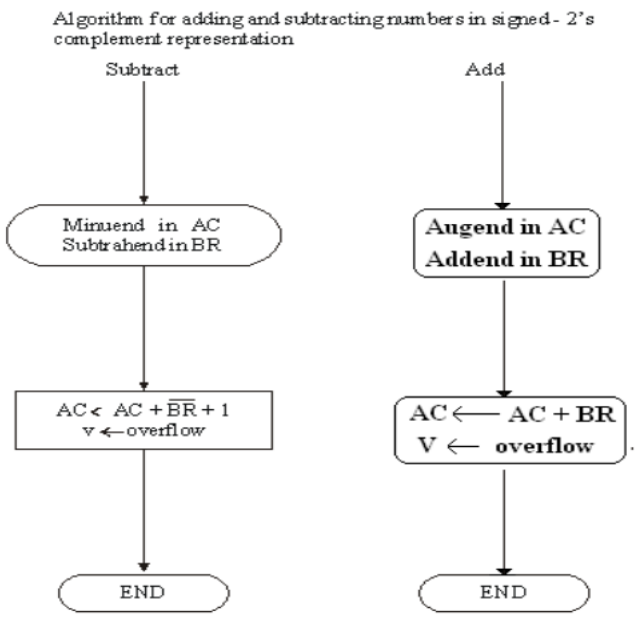
(iii) Bit ‘z’(zero) is set to 1. if the output of ‘ALU’ contains all zeros. Otherwise, it is set to zero.

(iv) Bit ‘v’ (overflow) is set to 1, if the Ex – OR of the last two carries i.e. c7 & c8 is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement form.



Adder-Subtractor with the help of 2's complement.

Addition of two numbers in signed 2's complement form consists of adding the numbers with the sign bits treated the same as the other bits of the number. A carry-out of the sign-bits position is discarded. The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend. When two numbers of a

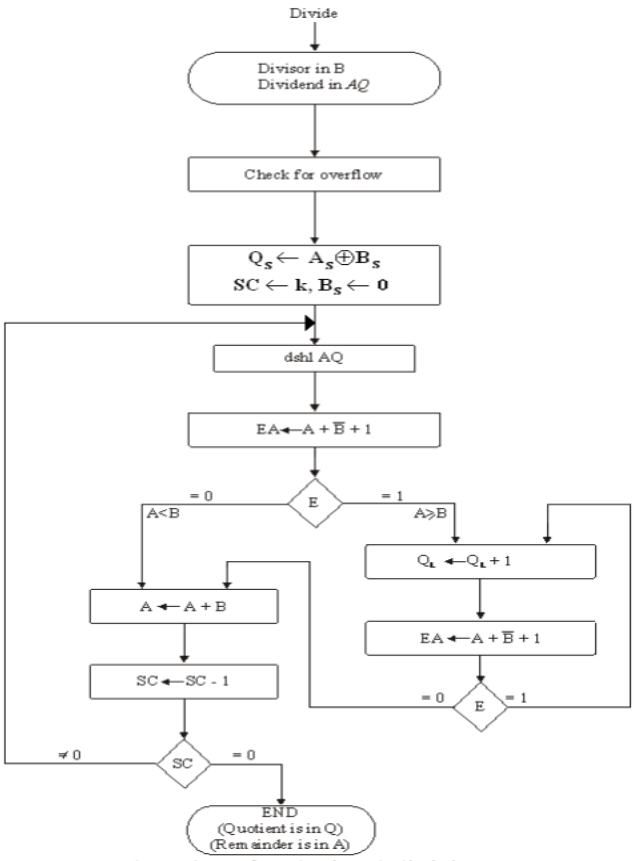


n digits each are added and the sum occupies n. + 1 digits then an overflow occurred. An overflow can be detected by inspecting the last two carries out of the addition. When the two carries are applied to an exclusive OR gate, the overflow is detected when, the output of the gate is equal to 1. The sum is obtained by adding the contents of AC and BR (including their sign bits). The overflow bit V is set to 1 if the exclusive-OR of the last two carries is 1, and it is cleared to 0

otherwise. The subtraction operation is accomplished by adding the content of AC to the 2's complement of BR. Taking the 2's complement of BR has the effect of changing a positive number to negative, and vice versa. An overflow must be checked during this operation because the two numbers added could have the same sign. The programmer must realize that if an overflow occurs, there with be an erroneous result in the AC resister.

With neat flow chart, explain the procedure for division of floating-point numbers carried out in a computer.

Decimal division is similar to binary division except of course that the quotient digits may have any of the 10 values from 0 to 9. In the restoring division method, the divisor is subtracted from the dividend or partial remainder as many times as necessary until a negative reminder results. The correct remainder is then restored by adding the divisor. The digit in the quotient reflects the number of subtractions up to but excluding the one that caused the negative difference. The decimal division algorithm is shown. It is similar to the algorithm with binary data except for the way the quotient bits are formed. The dividend (or partial remainder) is shifted to the left, with its most significant digit placed in Ae. The divisor is then subtracted by adding its 10's complement value. Since Be is initially cleared, its complement value is 9 as required. The carry in E determines the relative magnitude of A and B, If E=0, it signifies that A<B. In this case the divisor is added to restore the partial remainder and QL stays at 0 (inserted there during the shift). If E=1, it signifies that A>B. The quotient digit in QL is incremented once and the divisor
subtracted again. This process is repeated until the subtraction results in a negative difference which is recognized by E being 0. When this occurs, the quotient digit is not incremented but the divisor is added to restore the positive remainder. In this way, the quotient digit is made equal to the number of times that the partial remainder "goes" into the divisor. The partial remainder and the quotient bits are shifted once to the left and the process is repeated k times to form k quotient digits. The remainder is then found in register A and the quotient is to register Q. The value of E is neglected.



Flowchart for decimal division

RISC architecture

The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer. The major characteristics of a RISC processor are:

1. Relatively few instructions.
2. Relatively few addressing modes.
3. Memory access limited to load and store instructions.
4. All operations done within the registers of the CPU.
5. Fixed-length, easily decoded instruction format.
6. Single-cycle instruction execution.
7. Hardwired rather than microprogrammed control.

8. Faster execution.

Other characteristics attributed to RISC architecture are:

1. A relatively large number of registers in the processor unit.

2. Use of overlapped register windows to speed-up procedure call and return.

3. Efficient instruction pipeline.

4. Compiler support for efficient translation of high-level language programs into machine language programs.

A large number of registers is useful for storing intermediate results and for optimizing operand references. The advantage of register storage as opposed to memory storage is that registers can transfer information to other registers much faster than the transfer of information to and from memory. Thus register-to memory operations can be minimized by keeping the most frequent accessed operands in registers. Studies that show improved performance for RISC architecture do not differentiate between the effects of the reduced instruction set and the effects of a large register file.