A floating-point (FP) number is a kind of fraction where the radix point is allowed to move. If the radix point is fixed, then those fractional numbers are called fixed-point numbers. The best example of fixed-point numbers are those represented in commerce, finance while that of floating-point is the scientific constants and values.

Ex:

INR 58698.75, 43.3B$, 5.4 meters - Fixed point fractions

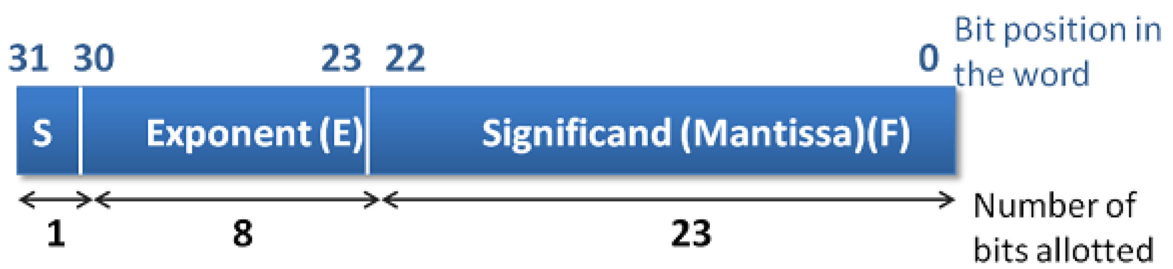
1.37×10^{23}, 0.137×10^{24}, 13.7×10^{22} - Floating point fractions

The disadvantage of fixed-point is that not all numbers are easily representable. For example, continuous fractions are difficult to be represented in fixed-point form. Also, very small and very large fractions are almost impossible to be fitted with efficiency.

A floating-point number representation is standardized by IEEE with three components namely the sign bit, Mantissa and the exponent. The number is derived as:

$$F = (-1)^s M \times 2^E$$

IEEE-754 standard prescribes single precision and double precision representation as in figure 10.1.



a. Single Precision

Floating point normalization

The mantissa part is adjusted in such a way that the value always starts with a leading binary '1' i.e. it starts with a non zero number. This form is called a normalized form. In doing so, the '1' is assumed to be the default and not stored and hence the mantissa 23 or 52 bits get extra space for representing the value. However, during calculations, the '1' is brought in by the hardware.

Floating point operations and hardware

Programming languages allow data type declaration called real to represent FP numbers. Thus it is a conscious choice by the programmer to use FP. When declared real the computations associated with such variables utilize FP hardware with FP instructions.

Add Float, Sub Float, Multiply Float and Divide Float is the likely FP instructions that are associated and used by the compiler. FP arithmetic operations are not only more complicated than the fixed-point operations but also require special hardware and take more execution time. For this reason, the programmer is advised to use real declaration judiciously. Table 10.1 suggests how the FP arithmetic is done.

Addition	X+Y	(adjusted $X_m + Y_m$) 2^{Y_e} where $X_e \le Y_e$
Subtraction	X-Y	(adjusted $X_m - Y_m$) 2^{Y_e} where $X_e \le Y_e$

Multiplication	X x Y	(adjusted X_m x Y_m) 2^{Xe+Ye}
Division	X/Y	(adjusted X_m / Y_m) 2^{Xe-Ye}

Addition and Subtraction

FP addition and subtraction are similar and use the same hardware and hence we discuss them together. Let us say, the X and Y are to be added.

Algorithm for FP Addition/Subtraction

Let X and Y be the FP numbers involved in addition/subtraction, where $Y_e > X_e$.

Basic steps:

1. Compute Ye - Xe, a fixed point subtraction
2. Shift the mantissa of X_m by (Ye - Xe) steps to the right to form $X_m 2^{Ye-Xe}$ if Xe is smaller than Ye else the mantissa of Y_m will have to be adjusted.
3. Compute $X_m 2^{Ye-Xe} \pm Y_m$
4. Determine the sign of the result
5. Normalize the resulting value, if necessary

We explain this with an example and as below:

X = 1.475 x 10^{15} Y = 1.23 x 10^{12}

X mantissa = 1.475,   X exponent = 15 to the base 10

Y mantissa = 1.23,   Y exponent = 12 to the base 10

For addition or subtraction to happen, the exponents need to be the same and then mantissa addition or subtraction is done. This is done by adjusting the smaller exponent and its mantissa. For every shift right of mantissa, the exponent value is increased by 1.

X exponent is > Y exponent.

Y becomes $\rightarrow$ 0.00123 x 10^{15}

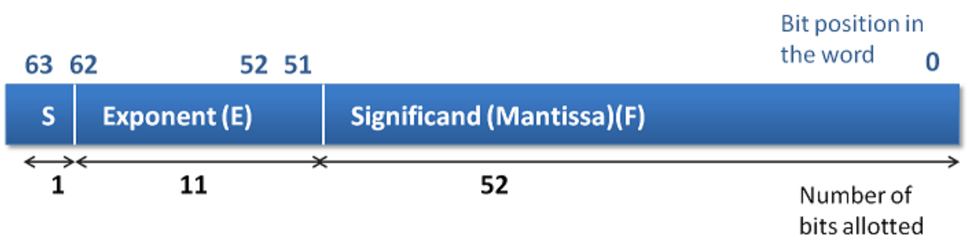
Now add X and Y.

$$\begin{array}{rl} & 1.475 \times 10^{15} \\ + & 0.00123 \times 10^{15} \\ \hline & 1.47623 \times 10^{15} \\ \hline \end{array}$$

(already in normalized form)

Subtraction is done the same way as an addition.

Multiplication and Division



Multiplication and division are simple because the mantissa and exponents can be processed independently. FP multiplication requires fixed point multiplication of mantissa and fixed-point addition of exponents. As discussed in chapter 3 (Data representation) the exponents are stored in the biased form. The bias is +127 for IEEE single precision and +1023 for double precision. During multiplication, when both the exponents are added it results in excess 127. Hence the bias is to be adjusted by subtracting 127 or 1023 from the resulting exponent.

Example:

$X = 1.3255 \times 10^{19}$ $\qquad$ $Y = 1.652 \times 10^{23}$

The process is " multiply the mantissas and add the exponents"

$$\begin{aligned} X \times Y &= (1.3255 \times 10^{19}) \times (1.652 \times 10^{23}) \\ &= (1.3255 \times 1.652) \times 10^{(19+23)} \\ &= 2.189726 \times 10^{42} \end{aligned}$$

Floating Point division requires fixed-point division of mantissa and fixed point subtraction of exponents. The bias adjustment is done by adding +127 to the resulting mantissa. Normalization of the result is necessary in both the cases of multiplication and division. Thus FP division and subtraction are not much complicated to implement.

All the examples are in base10 (decimal) to enhance the understanding. Doing in binary is similar.

Floating Point Hardware

The floating-point arithmetic unit is implemented by two loosely coupled fixed point datapath units, one for the exponent and the other for the mantissa. One such basic implementation is shown in figure 10.2.

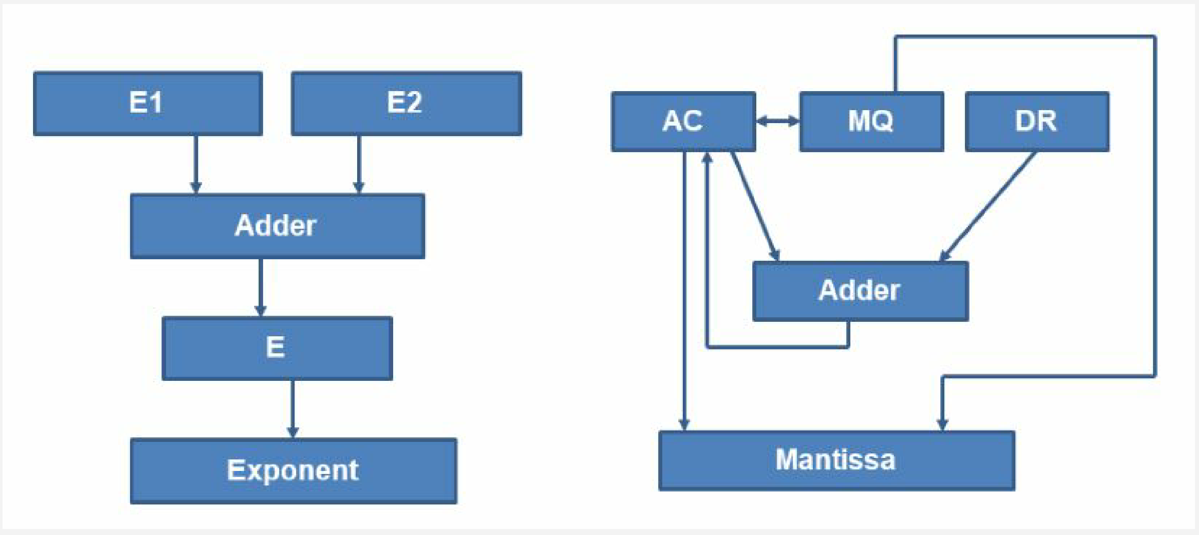


Figure 10.2 Typical Floating Point Hardware

Along with early CPUs, Coprocessors were used for doing FP arithmetic as FP arithmetic takes at least 4 times more time than the fixed point arithmetic operation. These coprocessors are VLSI CPUs and are closely coupled with the main CPU. 8086 processor had 8087 as coprocessor; 80x86 processors had 80x87 as coprocessors and 68xx0 had 68881 as a coprocessor. Pipelined implementation is another method to speed up the FP operations. Pipelining has functional units which can do the part of the execution independently.

Additional issues to be handled in FP arithmetic are:

- FP arithmetic results will have to be produced in normalised form.
- Adjusting the bias of the resulting exponent is required. Biased representation of exponent causes a problem when the exponents are added in a multiplication or subtracted in the case of division, resulting in a double biased or wrongly biased exponent. This must be corrected. This is an extra step to be taken care of by FP arithmetic hardware.
- When the result is zero, the resulting mantissa has an all zero but not the exponent. A special step is needed to make the exponent bits zero.
- Overflow – is to be detected when the result is too large to be represented in the FP format.
- Underflow – is to be detected when the result is too small to be represented in the FP format. Overflow and underflow are automatically detected by hardware, however, sometimes the mantissa in such occurrence may remain in denormalised form.
- Handling the guard bit (which are extra bits) becomes an issue when the result is to be rounded rather than truncated.

The operands of the instructions can be located either in the main memory or in the CPU registers. If the operand is placed in the main memory, then the instruction provides the location address in the operand field. Many methods are followed to specify the operand address. The different methods/modes for specifying the operand address in the instructions are known as addressing modes.

Types of Addressing Modes

There are various types of Addressing Modes which are as follows –

Implied Mode – In this mode, the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied -mode instruction because the operand in the accumulator register is implied in the definition of the instruction. All register reference instructions that use an accumulator are implied-mode instructions.

Instruction format with mode field

Opcode	Mode	Address

Immediate Mode – In this mode, the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field instead of an address field. The operand field includes the actual operand to be used in conjunction with the operation determined in the instruction. Immediate-mode instructions are beneficial for initializing registers to a constant value.

- **Register Mode** – In this mode, the operands are in registers that reside within the CPU. The specific register is selected from a register field in the instruction. A k-bit field can determine any one of the 2^k registers.

- **Register Indirect Mode** – In this mode, the instruction defines a register in the CPU whose contents provide the address of the operand in memory. In other words, the selected register includes the address of the operand rather than the operand itself.

- A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

- **Autoincrement or Autodecrement Mode** &minuend; This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register defines a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be obtained by using the increment or decrement instruction.

- **Direct Address Mode** – In this mode, the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction, the address field specifies the actual branch address.

Indirect Address Mode – In this mode, the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

- **Indexed Addressing Mode** – In this mode, the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory.

Register Transfer Timing

As we have seen, information is processed in the data path by register transfers. In a register transfer operation, information is moved out of a register, along a bus, possibly through combinational logic, and into another register. These actions take time, and in this section we will discuss how much time they take Being able to estimate the timing of such actions is crucial in determining the maximum clock frequency that a circuit will support.

The next figure shows an example of the clocking and timing relationships that exist along the data path. The figure shows the information being propagated from register R1, the transmitter, through tri-state buffer and bus, through a combinational logic block such as an ALU into register R2, the receiver. Propagation time through the wires interconnecting the registers and logic gates is assumed negligible

What is Synchronous Data Transfer

In synchronous data transfer, the transmitter and the receiver are synchronized and uses a common timing signal. It uses timing signals for synchronization. Here, the data flows as a continuous stream one after the other. The transmitter sends data, and the receiver counts the number of bits in the received data. Furthermore, there are no gaps between data. In this method, the timing signals must be accurate to transfer data efficiently. Moreover, this method is faster than asynchronous data transferring.

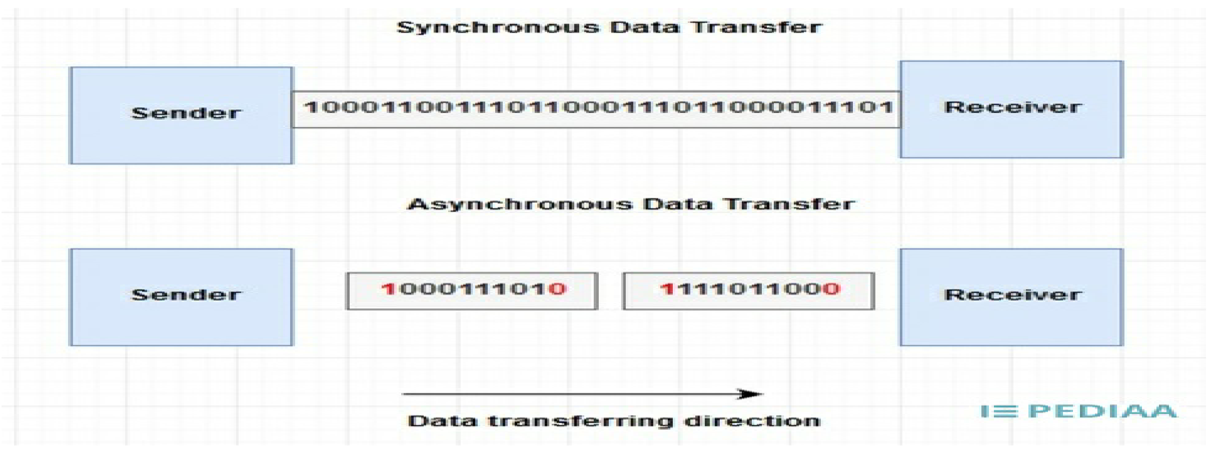


Figure 1: Synchronous and Asynchronous Data Transfer

In a digital system, if the other registers share the same clock with the CPU registers, the data transferring between the CPU and input and output devices is a synchronous data transfer. Both these units obtain the clock pluses from the common pulse generator.

What is Asynchronous Data Transfer

In asynchronous data transfer, the transmitter and receiver operate at different clock frequencies. It uses the start and stop bits to the data. According to the above example (figure 1), each byte of data is embedded in start and stop bits. The ‘0’ indicates the start bit while ‘1’ indicates the end bit. The ‘1’ and ‘0’ highlighted in red are the start and stop bits. Furthermore, timing is not an important factor in asynchronous data transfer.

In a digital system, if the other registers and CPU registers use their own private clocks, they have different timing signals. Therefore, the CPU and input and output devices should coordinate to transfer data. It is called an asynchronous data transfer.

Difference Between Synchronous and Asynchronous Data Transfer

Definition

Synchronous transfer is the data transfer method that sends a continuous stream of data to the receiver using regular timing signals that ensures both transmitter and receiver are synchronized with each other. Conversely, Asynchronous Data Transfer is the data transfer method that sends data from transmitter to receiver with parity bits (start and stop bits) in uneven intervals. Thus, this explains the fundamental difference between synchronous and asynchronous data transfer.

Clocks

In synchronous data transfer, the sender and receiver operate on the same clock frequencies whereas in asynchronous data transfer, the sender and receiver operate on different clock frequencies. Hence, this is the main difference between synchronous and asynchronous data transfer.

Data Transferring Speed

Data transferring speed is another difference between synchronous and asynchronous data transfer. Synchronous transfer is faster than asynchronous transfer.

Start and Stop Bits

There is no overhead of extra start and stop bits in synchronous transfer. On the other hand, asynchronous data transfer uses start and stop bits.

Vector processing is a central processing unit that can perform the complete vector input in individual instruction. It is a complete unit of hardware resources that implements a sequential set of similar data elements in the memory using individual instruction.

The scientific and research computations involve many computations which require extensive and high-power computers. These computations when run in a conventional computer may take days or weeks to complete. The science and engineering problems can be specified in methods of vectors and matrices using vector processing.

Features of Vector Processing

There are various features of Vector Processing which are as follows –

- A vector is a structured set of elements. The elements in a vector are scalar quantities. A vector operand includes an ordered set of n elements, where n is known as the length of the vector.
- Each clock period processes two successive pairs of elements. During one single clock period, the dual vector pipes and the dual sets of vector functional units allow the processing of two pairs of elements.

  As the completion of each pair of operations takes place, the results are delivered to appropriate elements of the result register. The operation continues just before the various elements processed are similar to the count particularized by the vector length register.
- In parallel vector processing, more than two results are generated per clock cycle. The parallel vector operations are automatically started under the following two circumstances –
  - When successive vector instructions facilitate different functional units and multiple vector registers.
  - When successive vector instructions use the resulting flow from one vector register as the operand of another operation utilizing a different functional unit. This phase is known as chaining.
- A vector processor implements better with higher vectors because of the foundation delay in a pipeline.
- Vector processing decrease the overhead related to maintenance of the loop-control variables which creates it more efficient than scalar processing.

Array processors increases the overall instruction processing speed. As most of the Array processors operates asynchronously from the host CPU, hence it improves the overall capacity of the system. Array Processors has its own local memory, hence providing extra memory for systems with low memory.

SIMD Array Processors

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams. A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an ALU the and registers

The master control unit controls all operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed. The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instruction s are send to all PE's simultaneously and results are returned to the memory. The best known SIMD array processor is the the Burroughs corps ILLIAC IV computer developed by . SIMD processors are highly specialized computers. They are only suitable for numer ical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.