3. Computer Arithmetic

1. Data Representation.

) Basic Formats

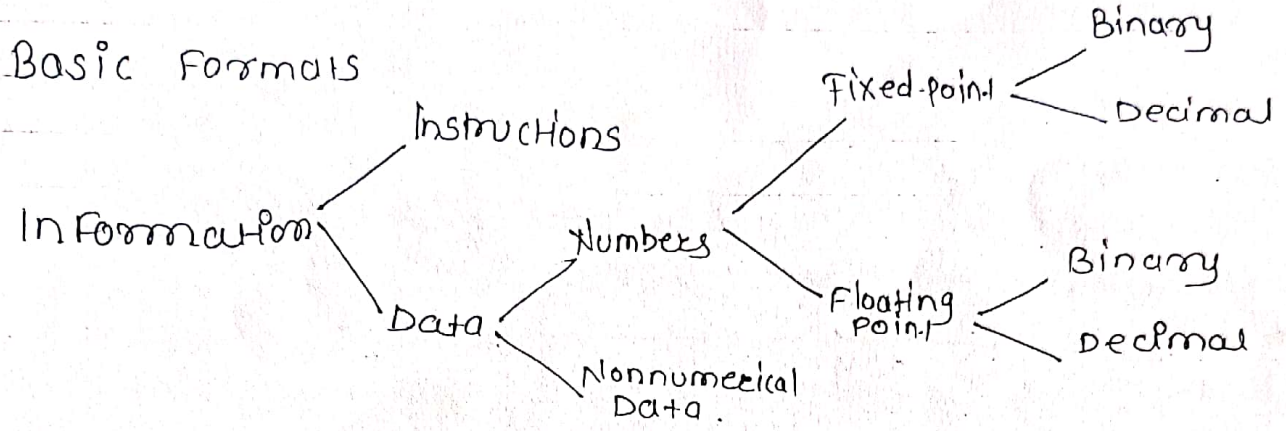


Fig. the Basic information types.

⇒ Word length.

- Information is represented in a digital computer by means of binary words, where word is a unit of information of some fixed length n. An n-bit word allow up to 2^n different items to be represented.

Example, with n=4, we can encode the 10 decimal digits as follows: 0=0000 1=0001 2=0010 3=0011 4=0100 5=0101 6=0110 7=0111 8=1000 9=1001

- To encode alphanumeric symbols/characters, 8-bit words called bytes are commonly used.

The cpu also has a standard word size for the data it processes. Word size is typically a multiple of 8, common cpu word sizes being 8, 16, 32 & 64 bits.

Bits	Name	Illustration	Typical uses
1	Bit		Status flag. Logic variable
8	Byte		smallest addressable memory item. Binary coded decimal digital pair.
16	Halfword		short fixed-point no. short address (offset). short instruction.
32	Word		Fixed / Floating point No. memory Addr.
64	Double word		long instruction. Double-precision Floating-point numbers.

Fig:- Some information formats of the motorola 68xx Microprocessors.

→ Storage Order.

A small but important aspect of data representation is the way in which the bits of a word is indexed

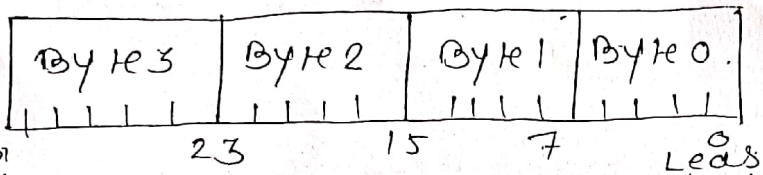


Fig:- Indexing convention for the bits & bytes of a word.

- we will usually follow the convention illustrated in above where the right-most bit is assigned the index 0 & the bits are labeled in increasing order from right to left.
- Advantage of this convention is that when the word is interpreted as an unsigned binary integer, the low-order indexes correspond the numerically less significant bits & the high order indexes correspond to the numerically more significant bits.

2. Fixed point numbers.

~~8~~ 8. In selecting a no. representation to be used in a computer, following factors should be taken into account:

- The no. types to be represented: eg. int / real no.
- The range of values likely to be encountered.
- The precision of the numbers, which refers to the maximum accuracy of the representation.
- The cost of the h/w required to store & process the numbers

→ Binary Numbers:

- The fixed-point format is derived directly from the ordinary representation of a number as a sequence of digits seperated by a decimal point.

The digits to the left of the decimal point represent an integer the digits to the right represent a fraction.
This is positional notation in which each digit has a fixed weight according to its position relative to decimal point.
If i > 0, then ith digit to the left (right) of decimal point has weight (10^{-i}).
Thus five digit decimal number 192.73 is equivalent to
$1 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 + 7 \times 10^{-1} + 3 \times 10^{-2}$.
The most fundamental no. representation used in computer employs a base two positional notation.
A binary word of the form.
$b_N \ldots b_3 b_2 b_1 b_0 . b_{-1} b_{-2} b_{-3} b_{-4} \ldots b_m$
Represents the number

$$\sum_{i=M}^{N} b_i 2^i$$

⇒ Signed Number.

Decimal Representation	Binary code		
	sign magnitude	ones complement	twos complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	1000	1111	0000
−1	1001	1110	1111
−2	1010	1101	1110
−3	1011	1100	1101
−4	1100	1011	1100
−5	1101	1010	1011
−6	1110	1001	1010
−7	1111	1000	1001

Table :- comparison of three 4 bit codes for signed binary number.

Table illustrates how integers are represented using all three codes when n = 4. These codes are all referred to as binary codes to distinguish them from the so called decimal co

→ Decimal Numbers.

To converting an unsigned binary number $X_{n-1} X_{n-2} \ldots X_0$ decimal requires a polynomial of the form.

$$\sum_{i=0}^{n-1} x_i 2^{k+i}$$ to be evaluated.

Eg: consider the addition 5+9 = 14 using Excess-three co

$$\begin{array}{r} 1000 = 5 \\ + \quad 1100 = 9 \\ \hline 0100 \\ + \quad 0011 \\ \hline 0111 = 4 \end{array}$$

carry 1 ←

Binary sum
correction
excess-three sum.

→ Hexadecimal Numbers.

- one/two other numerical codes are encountered in the design or use of computers. of particular importance is hexadecimal (he

The unsigned hexadecimal integer 2FA0C has the interpretation

$$2 \times 16^4 + F \times 16^3 + A \times 16^2 + 0 \times 16^1 + C \times 16^0$$
$$= 2 \times 65,536 + 15 \times 4,096 + 10 \times 256 + 0 \times 16 + 12 \times 1$$
$$= 195,084$$

Hence $2FA0C_{16} = 195,084_{10}$.

- Hexadecimal code is useful for representing long binary numbers. a consequence of the fact that the base 16 is power of

A hexadecimal number is converted to binary simply by replacing each hex digit by the equivalent 4 bit binary form.

Eg: we can convert $2FA0C_{16}$ to Binary by replacing the first digit 2 by 0010 the second digit F by 1111, the third digit A by 1010 and so on.

3. Floating point Number.

The range of numbers that can be represented by a fixed-point number mode is insufficient for many applications, particularly scientific computations where very large & very small numbers are encountered
eg. It is easier to write a quintillion as
1.0×10^{18}.

1) Basic formats.

Three numbers are associated with a floating point Number a mantissa M, an exponent E and a base B.
The mantissa M is also referred to as the significand / fraction in the literature.
These three components together represent the real number $M \times B^E$.

For example in IC implementation the mantissa & exponent are encoded as a fixed-point number with a base r i.e usually 2 / 10.

A floating point numbers is therefore stored as a word (M,E) consisting of a pair of signed-fixed-point numbers a mantissa M, which is usually a fraction / an integer & an exponent E, which is an integer.

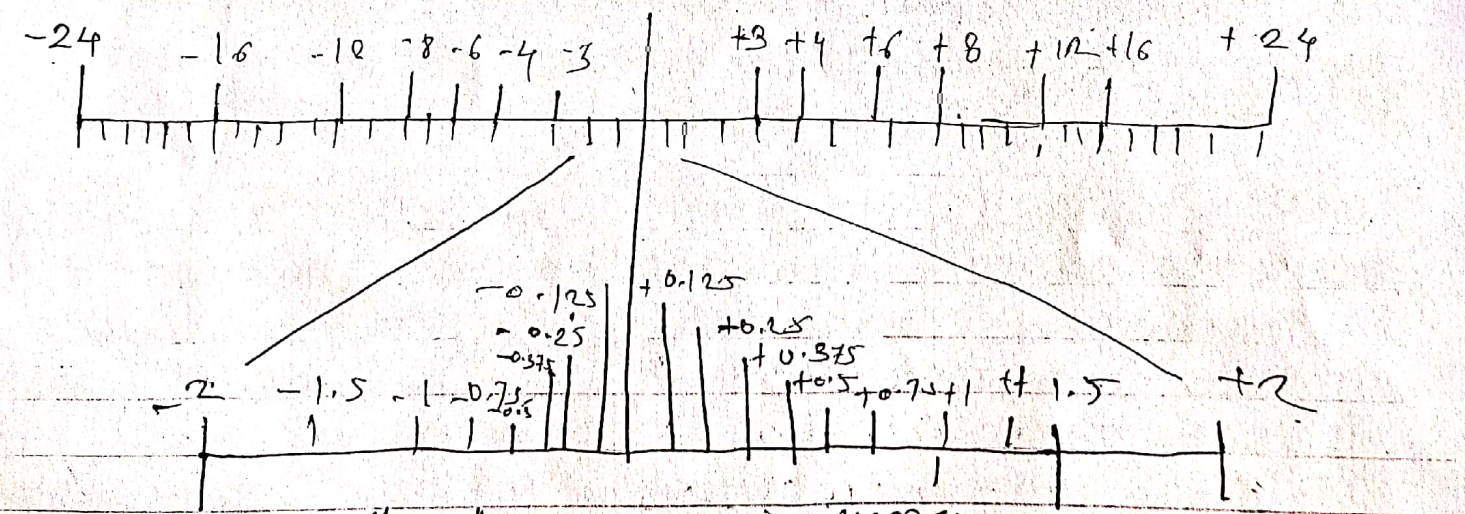


Fig. The real numbers representable by a hypothetical 6-bit floating-point format.

ii) Normalization and Biasing

- Floating point representation is redundant in the sense that the same number can be represented in more than one way.
- It is generally desirable to have a unique / normal form for each representable number in floating point system.
- consider the common case where the mantissa is a signed magnitude fraction & base q or is used. The mantissa is said to be normalized if the digit to the right of the radix point is not zero. i.e no leading zeros appear in the magnitude part of the number.
- Normalization restricts the magnitude of a fractional binary mantissa to the range

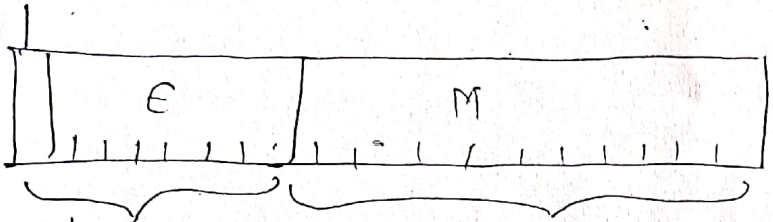
$1/2 \leq |M| < 1$.

- If k bits are allowed for the exponent including its sign, then 2^k exponent bit patterns are available to represent signed integers.
- The quantity K is called bias & an exponent encoded in this way called a biased exponent / characteristic.

iii) IEEE 754.

- Until the 1980's floating point number formats varied from one computer family to the next, making it difficult to transport programs between different computers without encountering small but significant differences in such areas as round-off errors.
- To deal with this problem the IEEE sponsored standard format for 32-bit and larger floating point numbers, known as the IEEE 754 standard, which has been widely adopted by computer manufacturers.

sign



8-bit Exponent (Excess-127 binary integer)

23-bit mantissa (Fraction part of sign magnitude binary significand with hidden bit)

Fig. IEEE 754 Standard 32-bit floating-point number format

the Number N represent by a 32-bit IEEE Standard, floating point number has the following set of interpretations:

If E = 255 & M ≠ 0, then N = NaN.
If E = 255 & M = 0, then N = $(-1)^S \infty$
If 0 < E < 255, then N = $(-1)^S 2^{E-127}(1.M)$.
If E = 0 and M ≠ 0, then N = $(-1)^S 2^{E-126}(0.M)$
If E = 0 & M = 0, then N = $(-1)^S 0$.

Fixed point Arithmetic

Addition and Subtraction.

Add & subtract instructions for fixed-point binary numbers are found in the instruction set of every computer. In smaller machines such as microcontrollers they are the only available arithmetic instructions.

→ Basic Adders.

The sum of Z_i, c_i of 2 1-bit no. x_i & y_i can be expressed by the half-adder logic Equations.

$$Z_i = x_i \oplus y_i$$
$$c_i = x_i y_i$$

Z_i = Sum bit
c_i = carry-out bit

$\oplus$ = exclusive-OR.

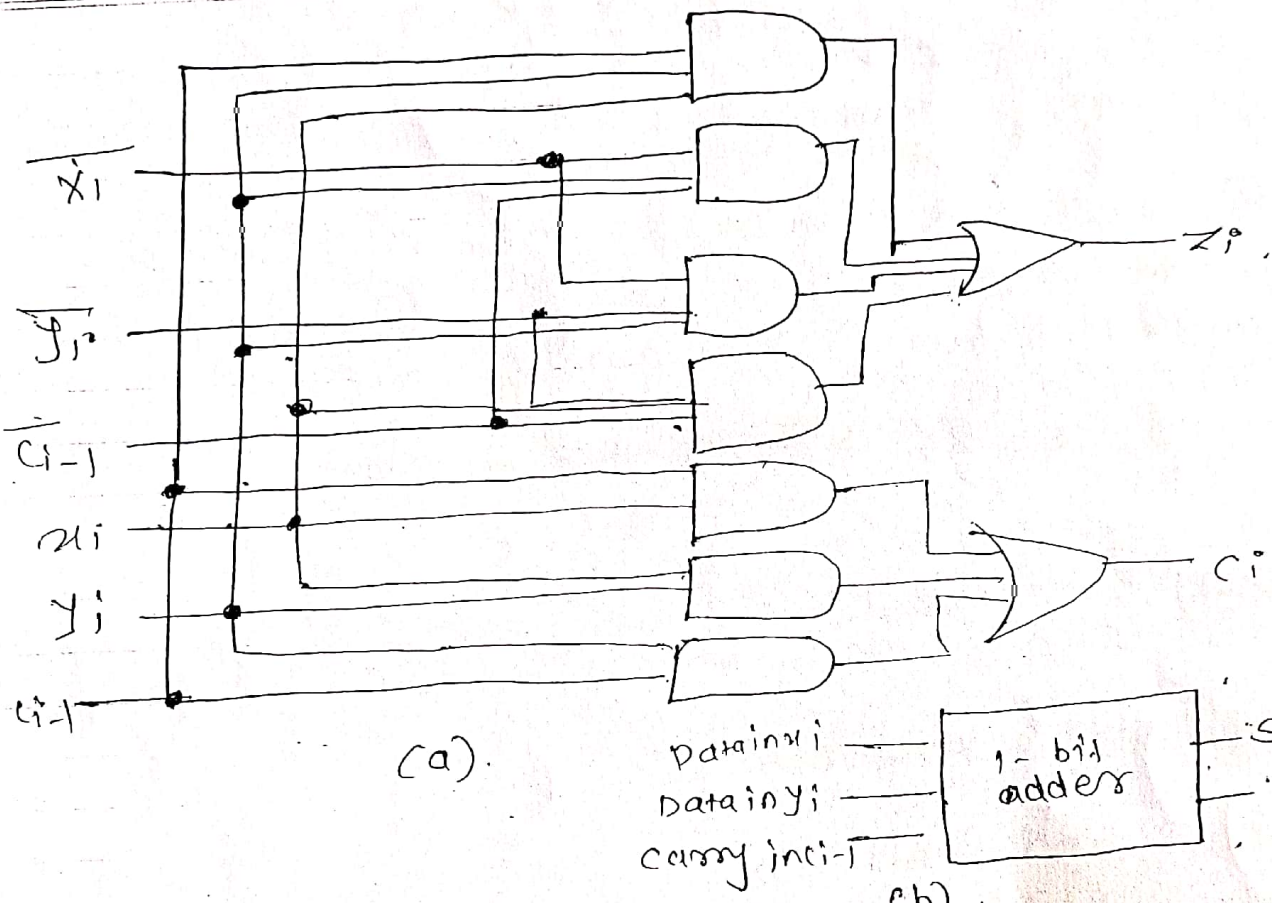


Fig: A 1-bit (full) adder: (a) two-level AND-OR logic ckt & (b) Symbol.

→ Subtractees.

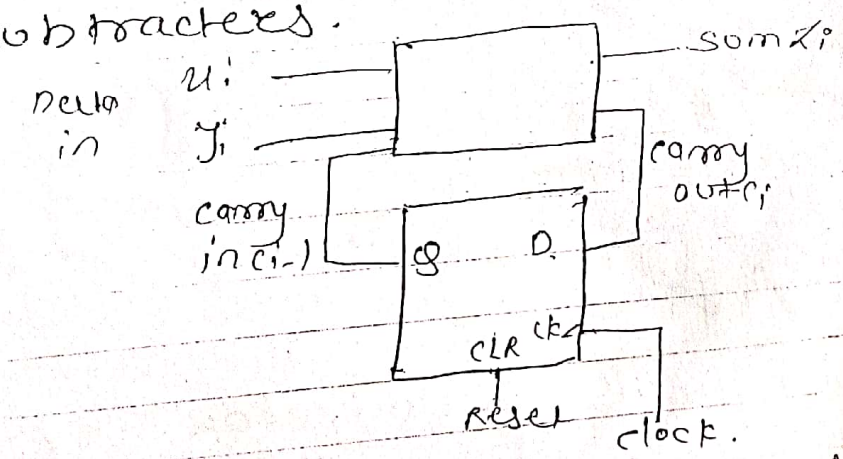


Figure: A serial binary Adder

- Adding -Y to X is equivalent to subtracting Y from X, the ability to add negative numbers implies the

- Subtraction is relatively simple with two's complement code because negation (changing) X to -X) is very easy to implement.

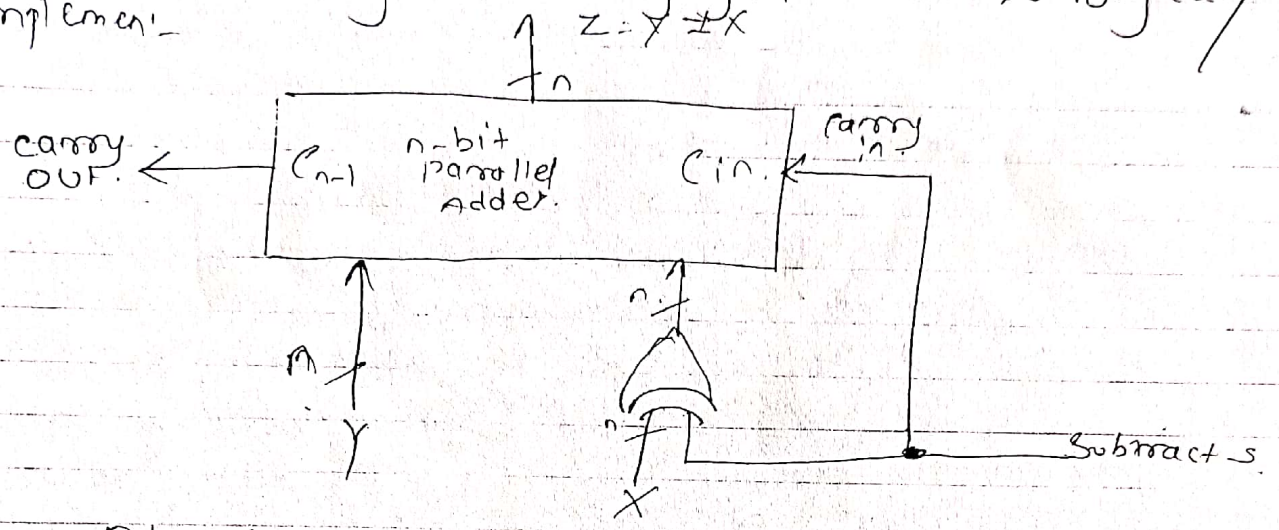


Fig: An n-bit two's-complement adder-subtractor

(iii) Overflow: when the result of an arithmetic operation exceeds the standard word size n, overflow occurs.
- overflow is indicated by a flag bit V in operations involving signed number: this flag is found in cpu status register.
- If we reinterpret the numbers in the preceding example as two's-complement rather than as unsigned then $X = 11101011$ denotes -21_{10}, while $Y = 00101010$ denotes $+42_{10}$.
- The result Z computed in $Z = X + Y = 11101011 + 00101010 =$ now denotes $+21_{10}$ & the fact that $C_{n-1} = 1$ does not indicate overflow.

$$\begin{array}{r} 00101010 \\ \oplus\ 11101011 \\ \hline 00010101 \end{array}$$

In fact, we can never have overflow on adding a positive to a negative number.
Overflow in Modulo-2^n two-complement addition can only result from adding two positive numbers or two negative number.

iii) High speed adders:

- The general strategy for designing fast adders is to reduce the time required to form carry signals.
- One approach is to compute the input carry needed by stage i directly from carrylike signals obtained from all the preceding stages i-1, i-2 ... 0 rather than wait for normal carries to ripple slowly from stage to stage. Adders that use this principle are called "carry lookahead adders".

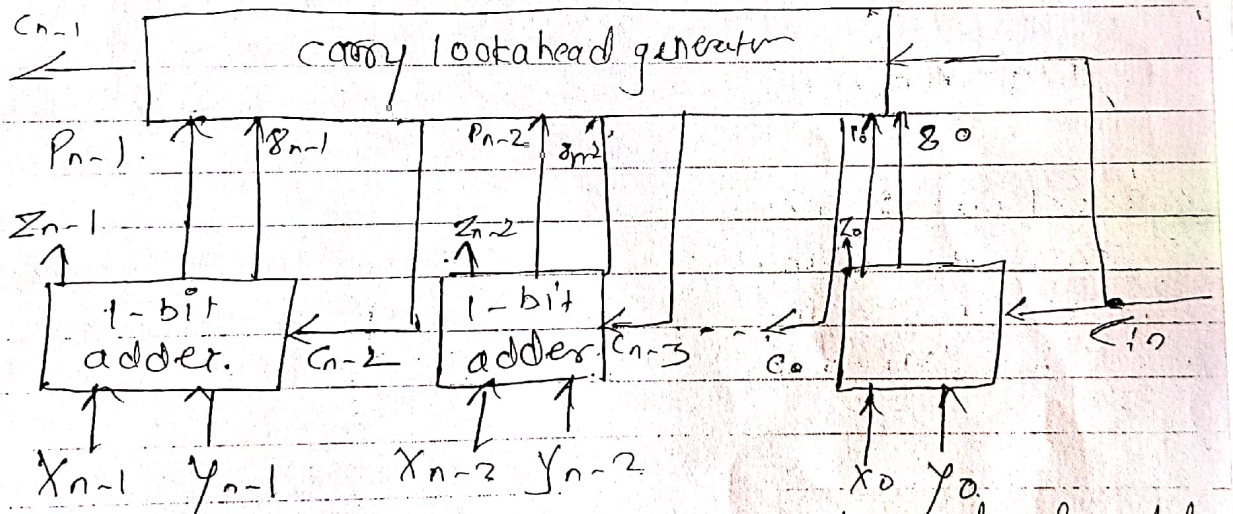


Fig. overall structure of carry lookahead adder

Figure shows the general form of a carry-lookahead adder circuit designed in this way.

We can further simplify the design by noting that the sum equation for stage i

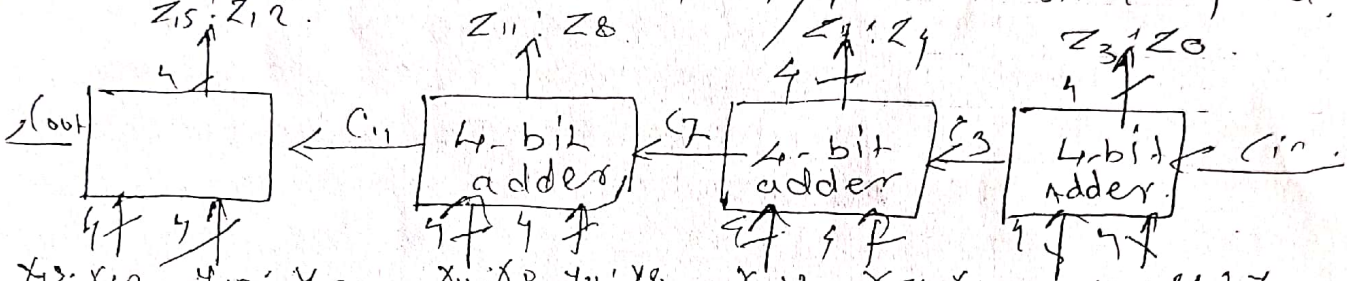
$$Z_i = x_i \oplus y_i \oplus c_{i-1}$$

is equivalent to

$$Z_i = P_i \oplus g_i \oplus c_{i-1}.$$

v) Adder Expansion

The methods of handling carry signals in the two main combinational adder designs considered so far namely ripple carry propagation & carry-lookahead can be extended to large adders of the kind needed to execute add instructions in, say, a 64-bit computer.



5 Fixed point Multiplication

Fixed-point multiplication requires substantially more hardware than fixed-point addition and, as a result, is not included in the instruction set of some smaller processors.

Multiplication is usually implemented by some form of repeated addition.

A simple but slow method to compute $x \times y$ is to add the multiplicand y to itself x times, where x is the multiplier. Often multiplication is implemented by multiplying y to k bits at a time and adding the resulting terms.

```
1010   multiplicand y
1101   multiplier x = x3 x2 x1 x0
-----
1010   x0 y
0000   x1 2y
1010   x2 4y
1010   x3 8y
-----
1000010  product p = Σ (j=0 to 3) xj 2^j y
```

Fig. Typical pencil & paper method for multiplication of unsigned binary numbers.

i°) Twos_complement Multiplier

- The multiplication of twos complement numbers pres
- some difficulties in the case of negative operands.
- A conceptually simple approach to two's-complement multiplication is to negate all negative operands at th beginning, perform unsigned multiplication on the resulting numbers, then negate the result if ne essary. twos_complement negation for an integer $X = X_{n-1} X_{n-2} X_{n-3} \ldots X_1 X_0$ is specified by

$$-X = \overline{X}_{n-1} \overline{X}_{n-2} \overline{X}_{n-3} \ldots \overline{X}_1 \overline{X}_0 + 000 \ldots 01$$

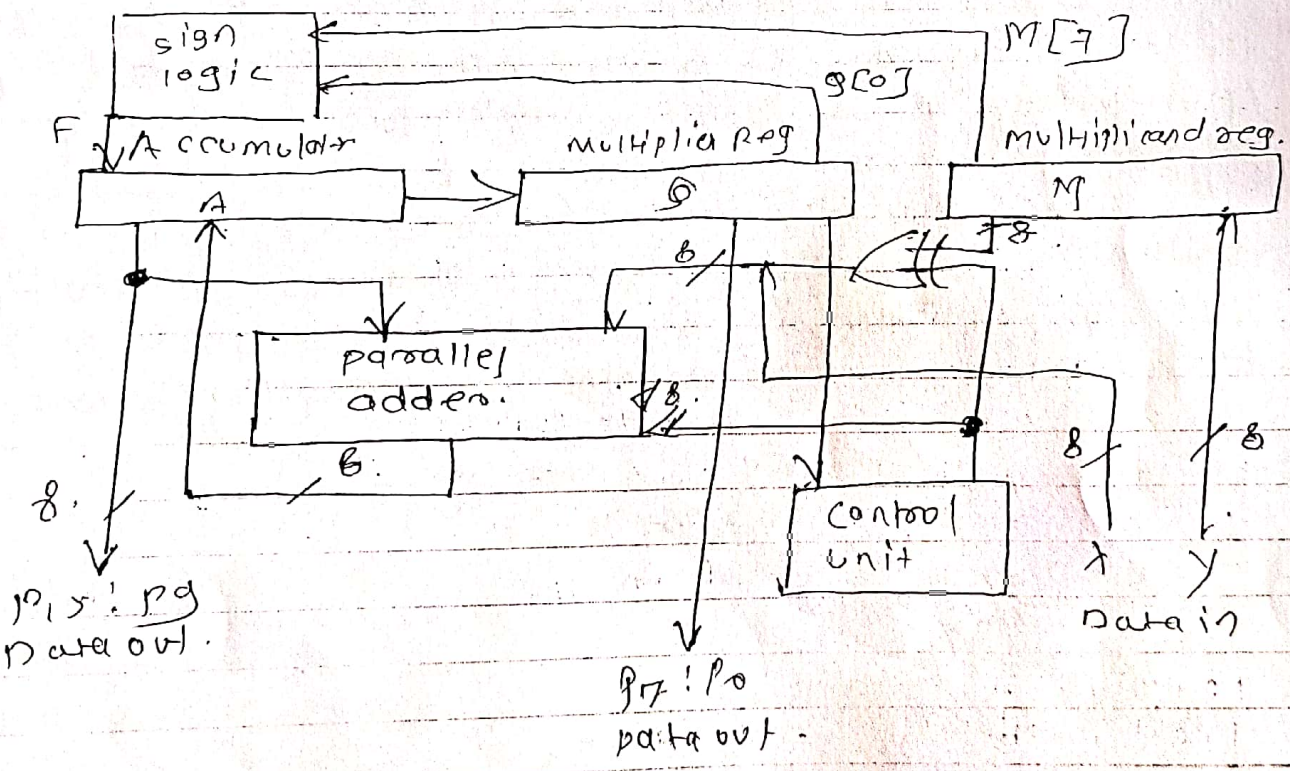


Fig : The datapath of the twos complement Multipli

ii) Booth's Algorithm

```
BoothMult       (in: INBUS; out: OUTBUS);
                Register A[7:0], M[7:0], Q[7:-1], COUNT[2:0];
                bus INBUS[7:0], OUTBUS[7:0];
Begin:          A := 0, COUNT := 0
INPUT:          M := INBUS;
                Q[7:0] := INBUS, Q[-1] := 0;
SCAN:           If Q[1] Q[0] = 01 then A[7:0] := A[7:0] + M[7:0], go to TEST;
                else if Q[1] Q[0] = 10 then A[7:0] := A[7:0] - M[7:0];
TEST:           if COUNT = 7 then go to OUTPUT;
SHIFT:          A[7] := A[7], A[6:0] Q := A Q[7:0];
INCREMENT:      COUNT := COUNT + 1; go to SCAN;
OUTPUT:         OUTBUS := A, Q[0] := 0;
                OUTBUS := Q[7:0];
End BoothMult;
```

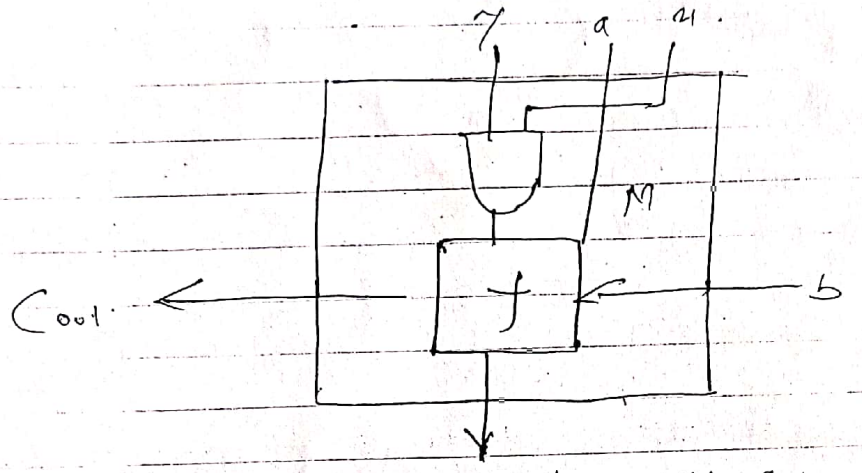
Fig: Description of an 8-bit multiplier implementing the basic Booth algorithm.

ii) Combinational Array Multipliers:

Advances in VLSI technology have made it possible to build combinational circuits that perform n x n-bit multiplication for fairly large values of n.

Suppose that two binary numbers $X = x_{n-1} x_{n-2} \ldots x_1 x_0$ & $Y = y_{n-1} y_{n-2} \ldots y_1 y_0$ are to be multiplied. For simplicity, assume that X & Y are unsigned integers. The product $P = X \times Y$ can therefore be expressed as

$$P = \sum_{i=0}^{n-1} 2^i x_i Y$$



5 Fig: Cell M for an unsigned array multiplier.

Fixed point division :-

In fixed point division two numbers, a divisor V and a dividend D, are given. The object is to compute a third number Q, the quotient, such that Q x V equ or is very close to D.

eg: $D = Q \times V + R$.

where R = remainder, is required to be less than V, that is, $0 \leq R \leq V$. We can then write

$$D/V = Q + R/V$$

R/V = small quantity representing the error in using Q alone to represent D/V; this error is zero if R

i) Restoring and Non restoring algorithm

The restoring and non-restoring division techniques be extended to signed numbers in much the same wa multiplication.

Non-restoring division therefore requires n additio subtraction, whereas restoring division requires an average of 3n/2 additions + subtraction.

Rdivider : (In : INBUS ; OUt : OUTBUS);
register S, A [n-1:0], M [n-1:0], Q[n-1:0],
COUNT [logₑn] : 0]; bus INBUS [n-1:0],
OUTBUS [n-1:0] COUNT = 0, S = 0,

EGIN ;

NPUT : A := INBUS; {Input the left half of the dividend}
Q := INBUS ; {Input the right half of the dividend}

BTRACT : S.A := S.A - M; {S is the sign of the result}

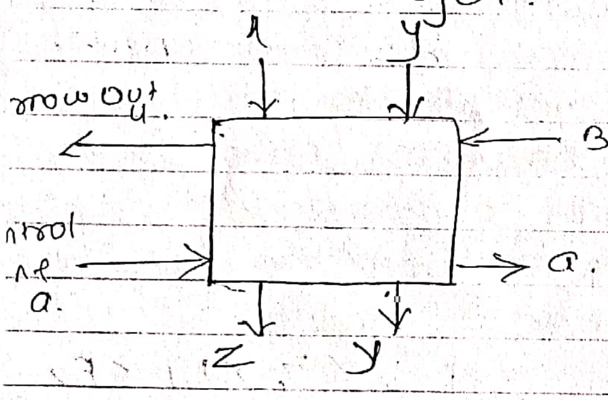
EST : if S = 0 then
begin Q [0] = 1;
If COUNT = n-1 then go to CORRECTION ; else
begin COUNT := COUNT + 1, S.A.Q [n-1:1] := AQ;
end Else {if S = 1}
begin Q [0] := 0;
If COUNT = n-1 then go to CORRECTION; ELSE
S.A := S.A + M, go to TEST, End

RRECTION : if := S1 then S.A := S.A + M.

UTPUT : OUTBUS := Q ; {O/P the quotient Q}
OUTBUS := A; {O/P the remainder R}

nd NRdivider ;

Fig : Nonrestoring division Algo for unsigned Integer.



q	Function
0	U z = x minus y minus t
1	z = x .

Fig : A cell D for array implementation of restoring

ii) combinational array divider:

combinational array circuits can be used for division as well as for multiplication.

Above fig. shows a cell suitable for implementing a version of the restoring division algorithm.

This cell is basically a full subtracter with t & u being the borrow-in and borrow-out bits, respectively.

The main output z is controlled by input a. when a=1, z is the difference bit defined by the arithmetic equation

$$z = x \text{ Minus } y \text{ minus } t.$$

when a=0, z=x. Thus the behavior of the cell is given by the logic equations.

$$z = x \oplus \bar{a}(y \oplus t)$$
$$u = \bar{x}y + \bar{x}t + yt.$$

iii) Division by repeated multiplication:

In system containing a high speed multiplier, division can be performed efficiently & at low cost using repeated multiplication. In each iteration a factor F_i is generated & used to multiply both the divisor V and the dividend D.

$$\therefore Q = \frac{DXF_0XF_1XF_2X\ldots}{VXF_0XF_1XF_2X\ldots}$$

F_i is chosen so that the sequence $VXF_0XF_1XF_2\ldots$ converges rapidly towards one.

7. Floating point Arithmetic

i) Basic operations

General formulas for floating-point addition, subtraction, multiplication & division are given in below figure.

Addition	$X+Y = (X_M 2^{X_E - Y_E} + Y_M) \times 2^{Y_E}$
Subtraction	$X-Y = (X_M 2^{X_E - Y_E} - Y_M) \times 2^{Y_E}$
Multiplication	$X \times Y = (X_M \times Y_M) \times 2^{Y_E + Y_E}$
Division	$X/Y = (X_M / Y_M) \times 2^{X_E - Y_E}$

Figure : - The four basic arithmetic operations for floating-point numbers.

Floating point addition & subtraction have 3 main steps:

1. Compute $Y_E - X_E$, a fixed-point subtraction.
2. Shift X_M by $Y_E - X_E$ places to the right to form $X_M 2^{X_E - Y_E}$.
3. Compute $X_M 2^{X_E - Y_E} \pm Y_M$, a fixed-point addition/subtraction.

ii) Difficulties

- Several minor problems are associated with exponent biasing. If biased exponents are added/subtracted using fixed-point arithmetic in the course of a floating-point calculation, the resulting exponent is doubly biased & must be corrected by subtracting the bias.

Another problem arises from the all-0 representation usually required of zero.

A floating-point operation causes overflow or underflow if the result is too large or too small to be represented.

iii) Floating point units

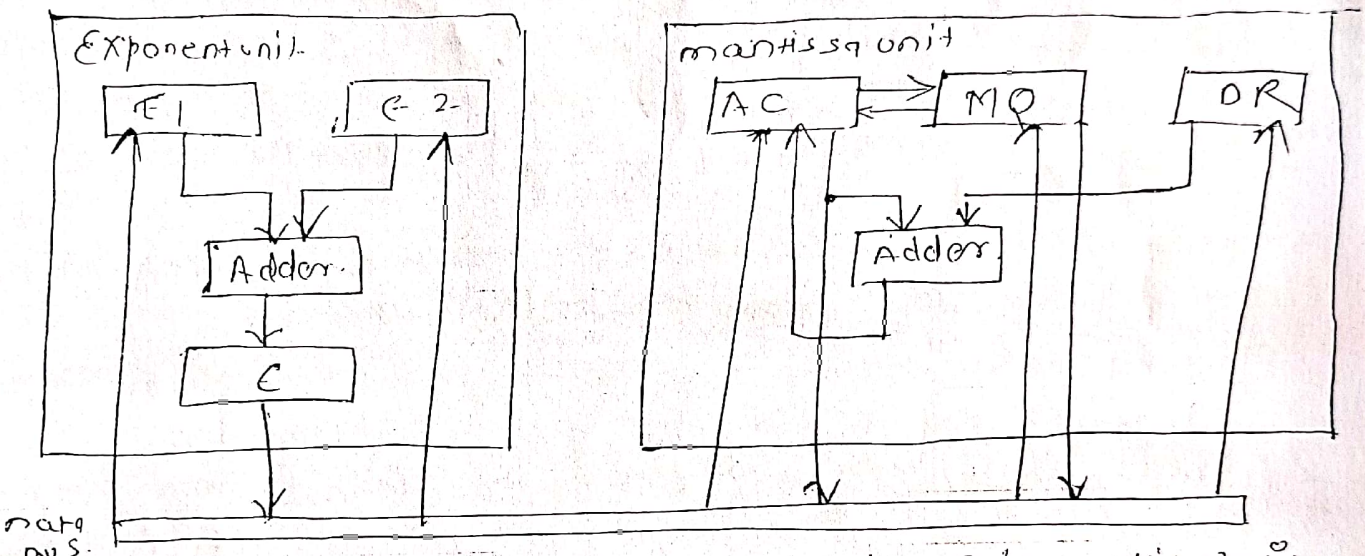


Fig: Datapath of a floating-point arithmetic unit

Floating-point arithmetic can be implemented by two loosely connected fixed-point datapath circuits, an exponent unit and a mantissa unit.

iv) Addition

The numbers have the 32-bit format of the IEEE standard 754. In this format each number N has a 23-bit fractional mantissa M with a hidden bit, an 8-bit exponent E is excess-127 code & a base B=2. The value of N is therefore given by the formula

$$N = (-1)^S 2^{E-127}(1.M)$$

The numbers to be added in this instance are

X = 0 01111111 10000000000000000000000

Y = 0 10000111 00101101010000000000000