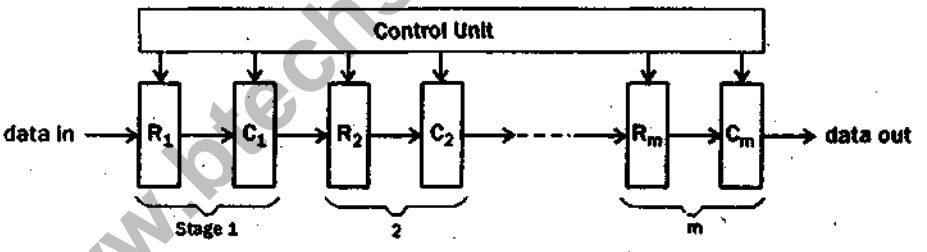
UNIT – III

DATAPATH DESIGN

Q.1. What is pipelining. Explain with the help of a diagram of a pipeline processor structure.

Ans. Pipelining is a general technique for increasing processor through put without requiring large amounts of extra hardware. It is applied to the design of the complex datapath units such as multipliers and floating point adders. It is also used to improve the overall through put of an instruction set processor.

Structure:

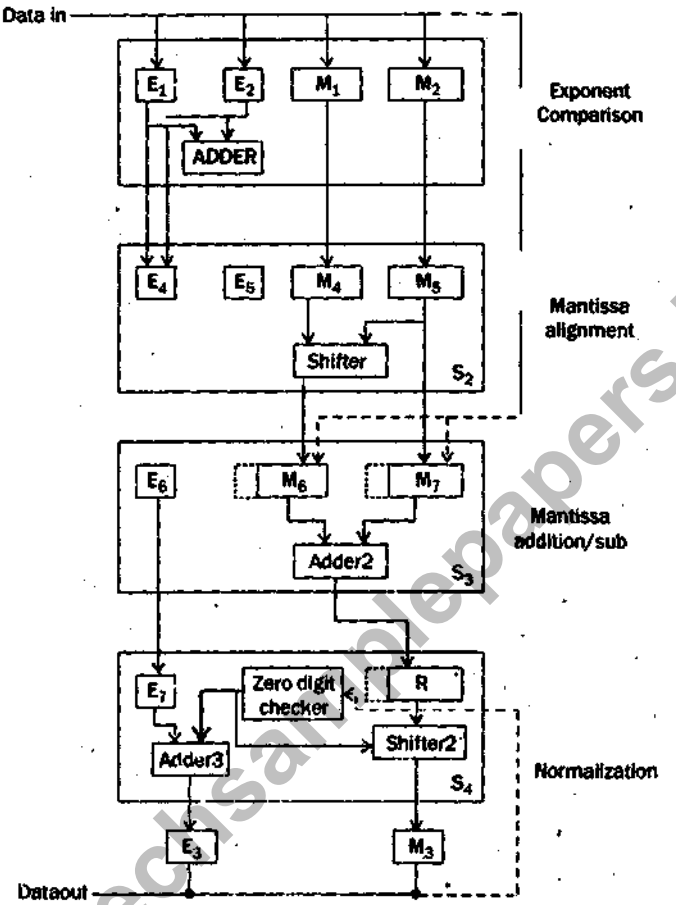


A stage S_i contains a multiword input register to latch R_i, and a datapath circuit C_i that is combinational. The R_i's hold partially processed results as they move through the pipeline; they also serves as buffers. A common clock signal causes the R_i's to change state synchronously. Each R_i receives a new set of input data D_{i-1} from the preceeding stage S_{i-1} except for R_1 whose data is supplied from an external source. D_{i-1} represents the results computed by C_{i-1} during the preceding clock period. Once D_{i-1} has been loaded into R_i, C_i proceeds to use D_{i-1} to compute a new data set D_i. In each clock period, every stage transfers its previous results to the next stage and computes new set of results.

Each stage of the '*M*'-stage pipeline takes 'T' set to perform its local suboperation and store its results. '*T*' is pipeline's clock period.

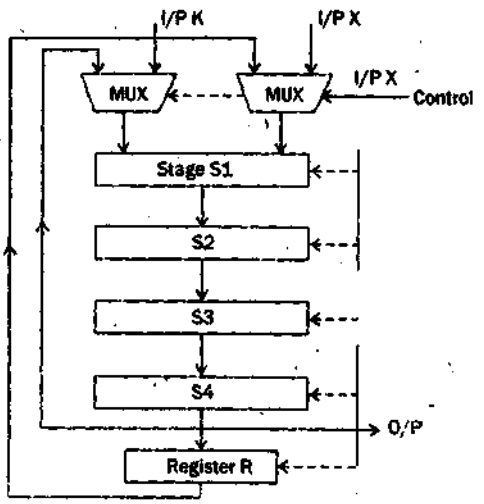
Q.2. Draw the pipelined version of floating point adder.

Ans.

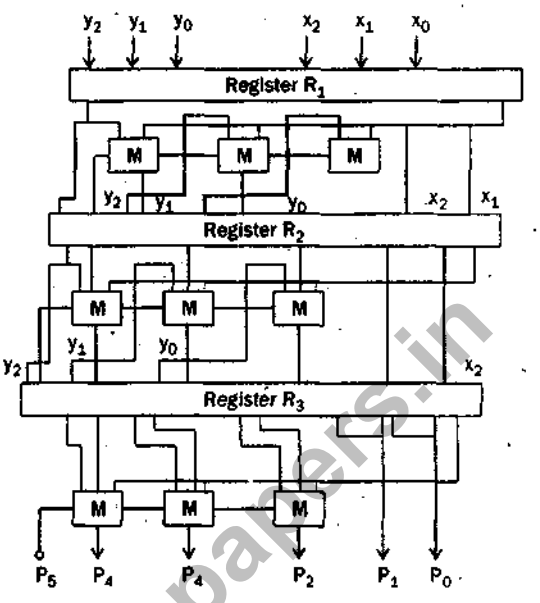


Q.3. What is the advantage of feedback in pipeline processor. Illustrate the importance of feedback path in pipeline adder.

Ans. Feedback enables the results computed by certain stages to be used in subsequent calculations by the pipeline.



Q.4. With the help of diagram explain pipelined array multiplier.

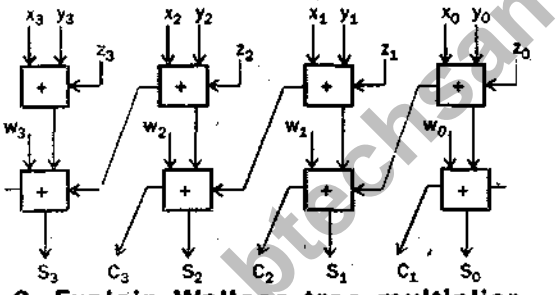


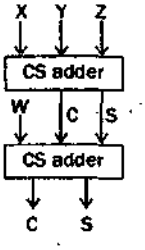
Ans. Two-n bit fixd point binary nos. $x = x_{n-1} x_{n-2} \cdots x_0$, $y = y_{n-1} y_{n-2} \cdots y_0$. It employs the 1-bit multiply and add cell *M*. Each cell '*M*' computers a 1-bit point $x_i y_j$ and adds it to both a path bit from the preceding stage and carry but generated by the cell on its right. Thus the *n*-cells in each stage S_i, compute a partial point of the form $P_i = P_{i-1} + x_i 2^i Y$.

The final point $P_{n-1} = XY$ computed by last stage

Q.5. What is carry save addition technique explain with diagram.

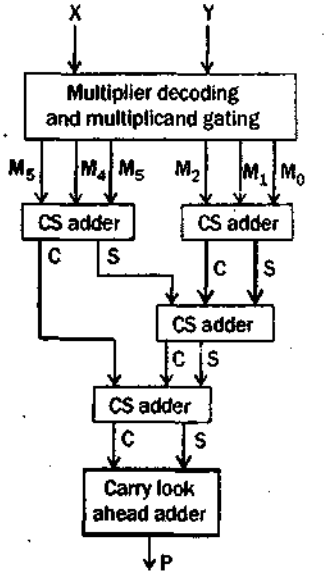
Ans. It is well suited to pipelining. An *n*-bit carry save adder consists of *n*-disjoint full adders. Its input is three *n*-bit nos to be added, white the output consists of the *n*-sum bits forming a word '*S*' and the '*n*' carry bits forming a word '*C*'. There is no cary propagate within the individual adders. The output *S* and *C* can be fed into another *n* bits carry save adder, they can be adder to a third *n*-bit numebr '*w*'. The carry connections are shifted to the left to corresponded to normal carry propagation.





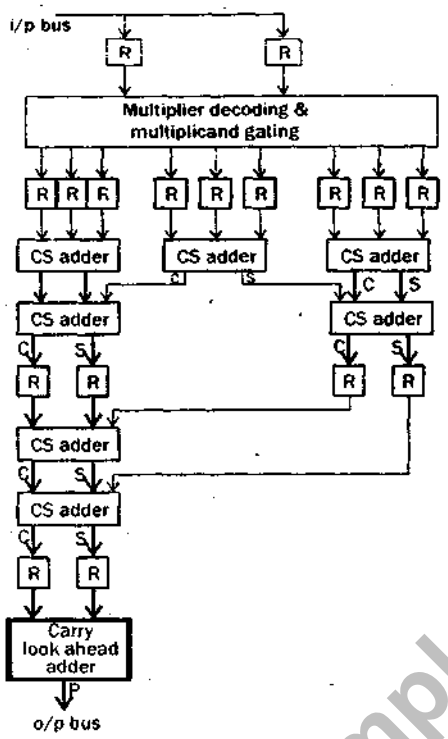
Q.6. Explain Wallace tree multiplier.

Ans. The inputs to the adder tree are *n*-terms of the form $M_i = x_i y 2^k$ where $M_i \rightarrow$ multiplicand '*Y*' multiplied by the *i*th multiplier but wieghted by the appropriate power of '2'. The desired point is $P = \sum_{i=0}^{n-1} M_i$. This sum is computed by the carry save adder tree, which produces 2*n* bit sum and a 2*n*-bit carry word. The final carry assimilation is performed by fast ader a carry lookahead adder.



Q.7. Briefly explain with diagram four stage pipelined version of carry save multiplier.

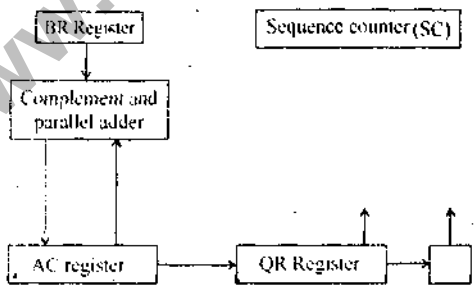
Ans. The Ist stage decodes the multiplier and transfers appripriately shifted copies of the multiplicand into the carry save adders. The outut of the Ist stage is a set of nos. that are then summed by cary save adder tree. The fourth and final stage contains a carry look a head adder to assimilate the carriers.



Q.8. State Booth Algorithm for multiplication of two numbers. Draw a logic diagram for the implementation of the Booth Algorithm for determining the product of two 8-bit signed numbers.

[UP Tech. 2004-2005, 2007-2008]

Ans. The logic diagram for the implementation is as follows:



In above diagram multiplicand is stored in register BR. Multiplier is stored in register QR. Q_n designates the least significant bit of the multiplier which is in register QR. Q_{n+1} is a flip flop. The combined value of AC and QR gives the final result.

The flow chart of Booth algorithm is shown in diagram given below.

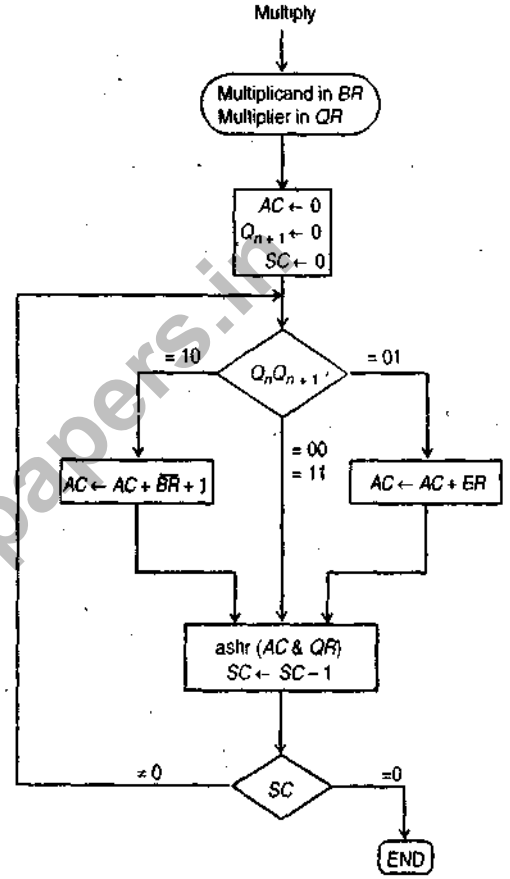


Fig. *Booth algorithm for multiplication of signed-2's complement numbers.*

AC and the appended bit Qn+1 are initially cleared to 0. The sequence counter (SC) is set to a number n which is equal to the number of bits in the multiplier. The two bits of the multiplier in Q_n and Q_{n+1} are inspected. If the two bits are equal to 10, that means the first 1 in the string of 1's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC. If the two bits are equal to 01 it means the first 0 in a string of 0's has been encountered. Then addition of multiplicand to the partial product in AC is done. When the two bits are equal the addition and subtraction of the multiplicand follow each other. As a result the two numbers which are added always have opposite signs, a condition which excludes an overflow.

The next step is to shift right the partial product and the multiplier including Q_{n+1}. This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC unchanged. The SC is decremented and the loop is repeated n times.

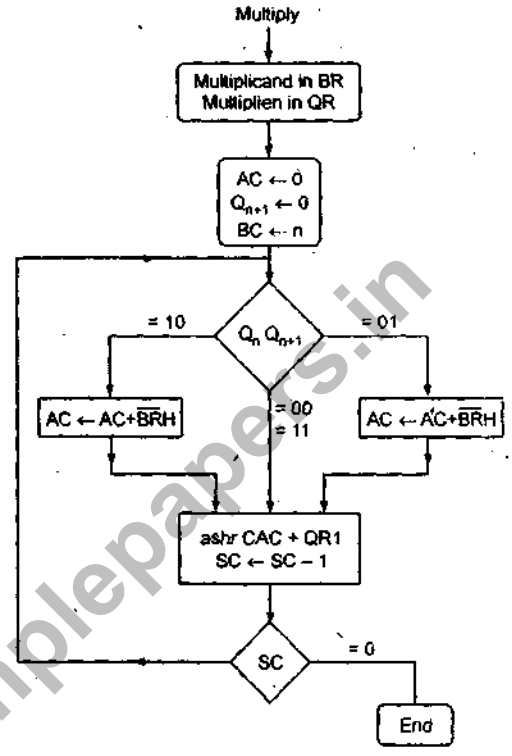
Q.9. Show the multiplication process using Booth's Algorithm when following Binary numbers are multiplied.

(-12) * (-18) [UP Tech. 2005-2006]

Ans. For $n = 5$. Both algorithm shows step by step multiplication of $(-12) * (-18) = +216$. Note that the multiplier in *QR* is negative and multiplicand in *BR* is also negative. The total 10-bit product appears in *AC* and *QR* is positive the final value of Q_{n+1} is the original sign bit of the multiplier and should not be taken as part of product.

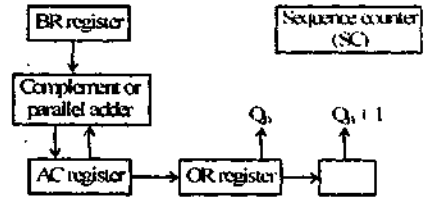
Qn QnH	BR = 10 111 BRH = 01001	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
10	Subtract BR	01001 01001			
11	ashr	00100	11001	1	100
01	ashr	00010 11001	01100	1	011
	ashr	11100	10110	0	010
00	ashr	11110	01011	0	001
10	subt BR	01001 00111			
	ashr	00011	10101	1	000

Booth's algorithm:



Q.10. Discuss the required hardware, hardware algorithm for Booth Multiplication. Iterate your algorithm for the product (+13) * (-15). [UP Tech. 2006-2007]

Ans. Hardware for booth algorithm :



$Q_n Q_{n+1}$	BR = 010010 BR + 1 = 101110	AC	QR	Q_{n+1}
	Initial	000000	111100	0
00	ashr	000000	011110	0
00	ashr	000000	001111	0
10	sub BR	101110 101110		
	ashr	110111	000111	1
11	ashr	111011	100011	1
11	ashr	111101	110001	1
11	ashr	111110	111000	1