RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES, BASAR
(A.Y. 2021-2022)
Computer Science and Engineering Department

Subject:COA Unit-2 subject code:cs2201
Faculty Name:Revya Naik V(Assistant Professor)

Signed magnitude addition or subtraction hardware Implementation

For two signed magnitude numbers, perform an addition or subtraction operation. Two numbers' magnitude parts are stored in the B register and the A register. Sign bits are stored in As and Bs flipflops. Either As or Bs, the flipflop holds only one bit of information, 0 or 1. Addition overflow flipflop(AVF) holds only one bit of information. If the result value is greater than the register storage capacity, then overflow will occur. This overflow bit of information is stored in AVF.

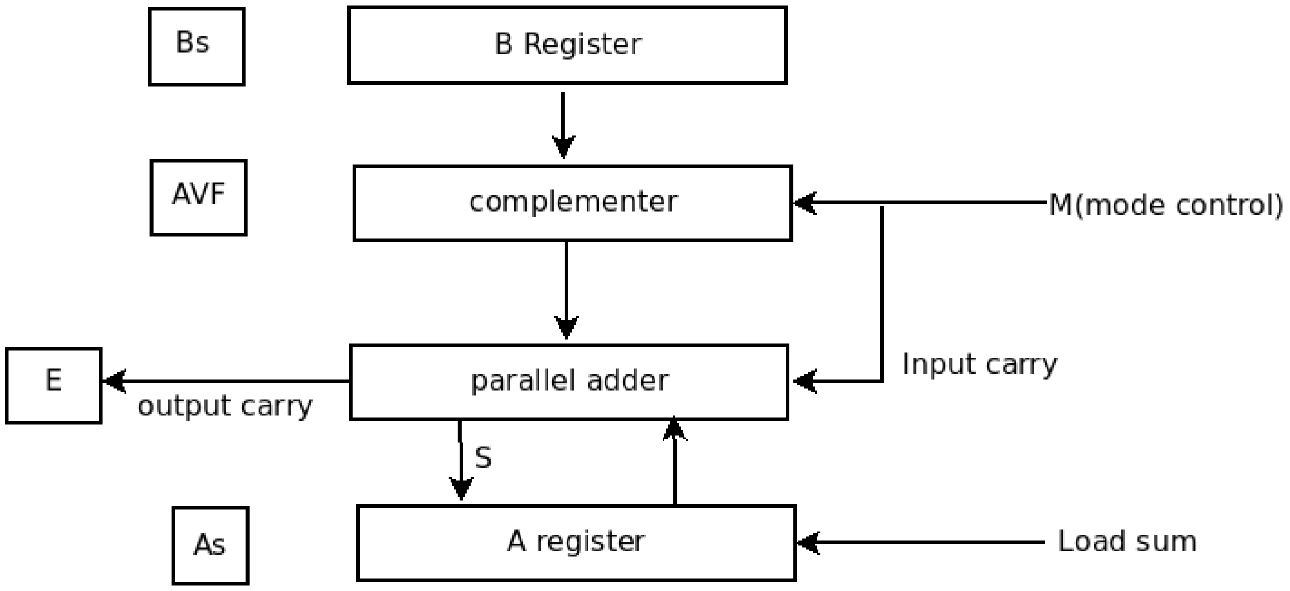

Figure 1 signed magnitude addition or subtraction Hardware Implementation

A parallel adder circuit is used to combine the B and A register values for addition. M (mode control) bit is set to 0 for addition operation, indicating that the complementer circuit transfers the B register value without complementing. A register and B register values are added and the result is stored back in A and As. When the M (mode control) bit is set to 1, the complementer circuit complements the B register value and transfers it to a parallel adder circuit. The completed B register value, A register value, and input carry are added by the parallel adder circuit and the result is stored in A and As.

Flowchart

In the below flowchart, for subtraction operations, minuend in A and subtrahend in B. Use augend in A and addend in B for addition operations. Before either add or subtract operation, sign bits are compared using the XOR operation, and if the XOR output is 1, it means both signs are different. If the XOR output is 0, it means both signs are the same. Before addition, signs are compared and XOR output is 0. Then A value and B value are added, and the result is stored in EA. The output of this operation is EA <--A + B is stored in E, and the result is stored in A register. After the addition operation, overflow bits are stored in AVF. In the addition operation, if signs are different (XOR output is 1), then the subtraction operation(EA <-- A - B) performed. The result is stored in the A register, and the output is carried in the E register. In a subtraction operation, overflow will not occur, so zero is assigned to AVF.

Before the subtraction operation, sign bits of magnitude are compared with the XOR operation. If the XOR output is 1, it means both signs are different and the operation is subtraction, then two magnitudes are added. If the XOR output is 0, it means both magnitude signs are the same and the operation is subtraction. A subtraction operation is performed. After the subtraction operation, the result is stored in the E and A registers. Output carries E compared with 0 and 1. If E is 1, it means that A > = B and that magnitude A is greater than or equal to magnitude B. The value of A is compared to the value of 0. If the result value A is zero, which means magnitude A is equal to magnitude B, then As is set to 0 and the result is stored in A and As. If E is zero, magnitude A is less than magnitude B, and the result is 2's complemented stored in A register and Assign bit 1's complemented stored in B register.

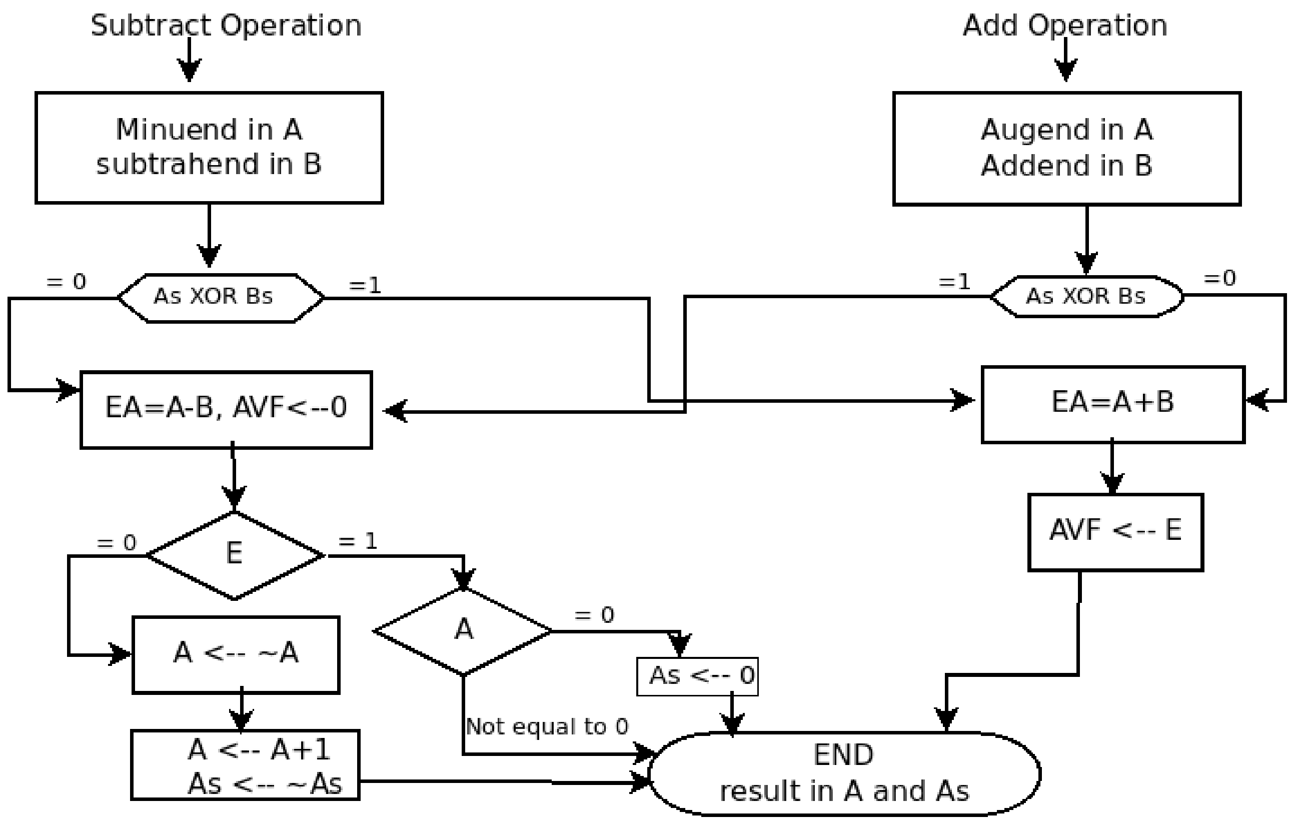

Figure 2 signed magnitude addition and subtraction operation flowchart

2’s complement number addition and subtraction

In 2’s complement addition and subtraction operations, sign bits of numbers are not separated. Including sign bits, numbers are stored in AC (Accumulator register) and in BR (Base register). The sign bits are added or subtracted with other bits in the complementer circuit and the parallel adder circuit. The overflow V is set to 1 if there is an overflow. The output is carry discarded.

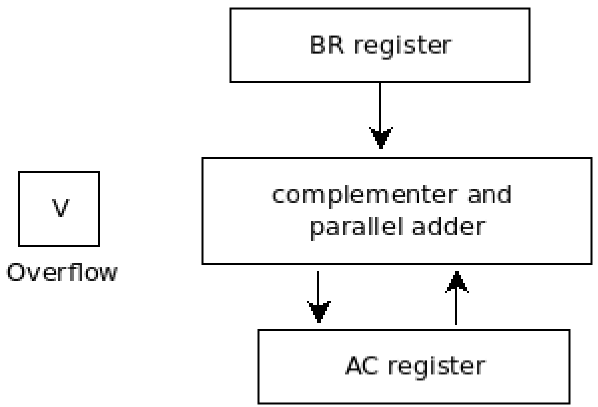

Figure 3. 2’s complement addition and subtraction hardware implementation

The algorithm for 2’s complement binary number addition operation and subtraction operation In Figure 4, including sign bits, the Accumulator register (AC) binary value and the BR (Base register) binary value are added. The overflow V is set to 1 if the exclusive OR of the last two bits carries 1, and otherwise it is set to 0. AC register value is added to 2's complement of BR register value for subtraction operation and vice versa.

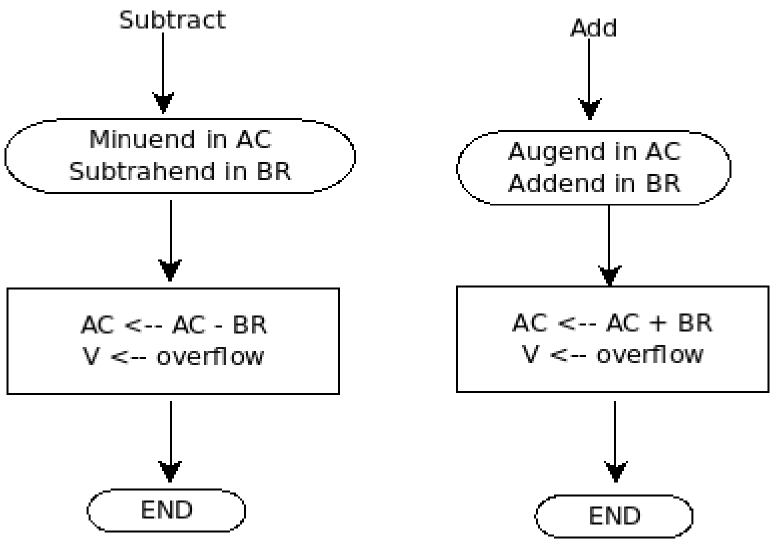

figure 4. 2’s complement addition and subtraction flowchart

Multiplication Algorithm

Signed magnitude multiplication hardware implementation

Hardware components used for signed magnitude multiplication are A register, B register, Q register, sequence counter register (SC), complementer and parallel adder circuit, and As, Qs, Bs, Qn flipflops.

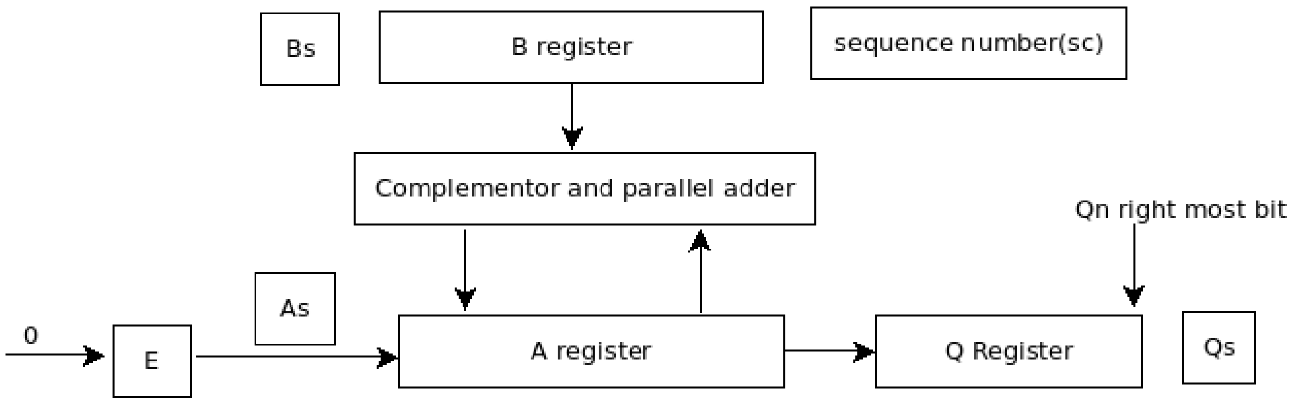

figure 5. signed magnitude number multiplication hardware implementation

Figure 5 shows the multiplicand in the B register and the sign bit Bs, the multiplier in the Q register, and the sign bit Qs. A register initial value is set to 0 and after A and B register values, a partial product is stored in A register. The sequence counter is set to n-1 (the number of bits in the multiplier excluding the sign bit).

In figure 6 (flowchart), A register initial value is 0, E initial value is 0. The XOR operation is performed with the multiplicand sign bit and multiplier sign bit, with the result bit stored in As and Qs flipflops. The flipflop Qn is the rightmost bit in the multiplier. If the Qn bit is set to 0, the shift right operation on E, A, and Q is performed, and the SC value is decremented by one. If Qn is 1, A and B register values are added by a parallel adder circuit and the partial product is stored in A register. This process is repeated until the sequence counter (SC) value reaches 0 and the final product is stored in the A and Q registers.

Example: In table 1, -6 * 3 multiplicand is -6, Bs is 1 and magnitude in B register binary value is 0110, and 3 is the multiplier in Q register binary value is 0011. SC is the number of bits in the multiplier excluding the sign bit. The final product is stored in A and Q registers (A = 0001 & Q = 0010). AQ = 00010010 in decimal value is 18.

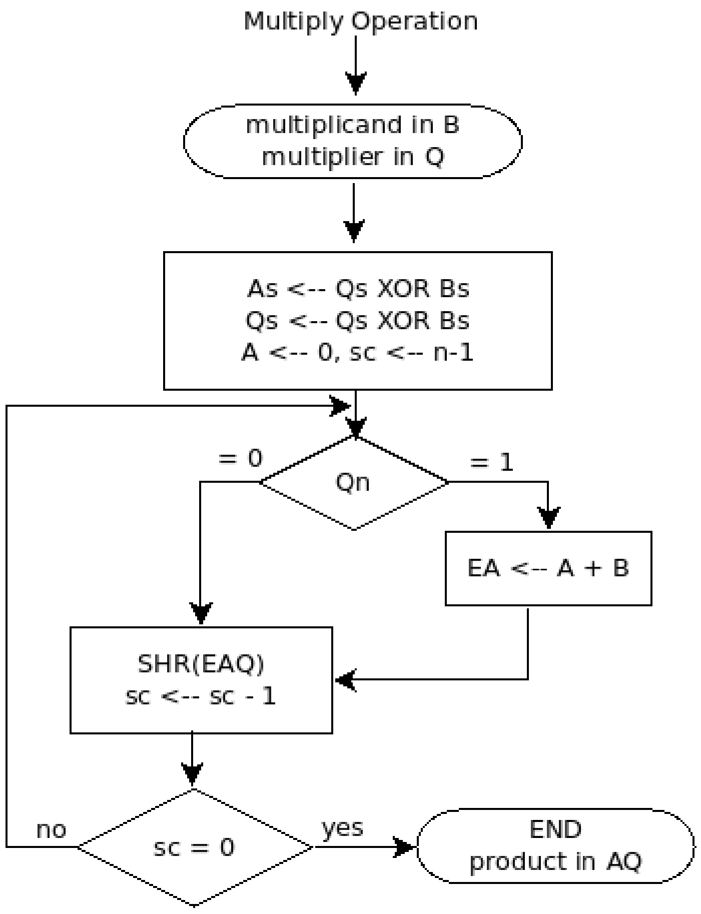

Figure 6. signed magnitude multiplication flowchart

Multiplicand in B=0110	**E**	**A**	**Q**	**SC**
Multiplier in Q	0	0000	0011	4
Qn = 1; add B to A		0110		
First partial product in A	0	0110	0011	
Shift right(EAQ)	0	0011	0001	3
Qn=1; add B to partial product A		0110		
Partial product in A	0	1001	0001	
Shift right (EAQ)	0	0100	1000	2
Qn=0; shift right(EAQ)	0	0010	0100	1
Qn=0; shift right(EAQ)	0	0001	0010	0

Table 1. signed magnitude numbers multiplication example (-6*3)

Booth's multiplication algorithm for 2's complement number

Booth's multiplication algorithm makes use of the AC (accumulator) register, the BR (base register) register, the Quotient register (QR), the sequence counter (sc), and the complementer and parallel adder circuits.Flipflop Qn is the rightmost bit in the multiplier, and Qn+1 is another flipflop.

In this multiplication operation, the multiplicand is in the BR register, the multiplier is in the QR register, the initial value of the AC register is 0, and the SC value is the number of bits in the multiplier, including the sign bit.

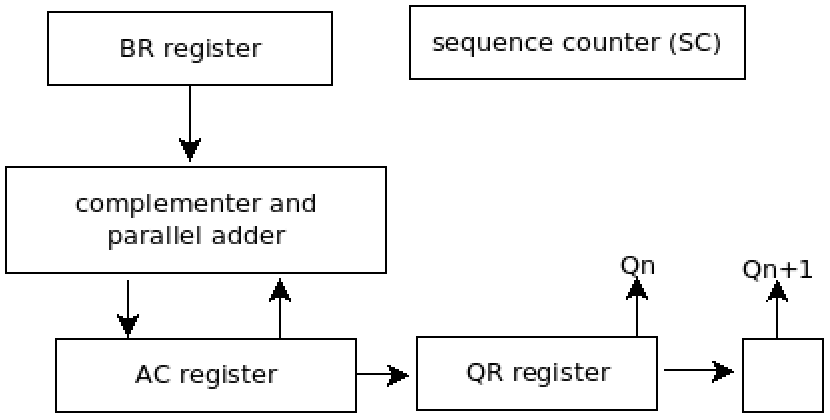

figure 7 booth's multiplication hardware implementation

In Figure 8, the Qn+1 flipflop initial value is zero and Qn is the rightmost bit in the multiplier. An addition or subtraction operation is performed based on the QnQn+1 value. If QnQn+1 is 00 or 11, then an arithmetic right shift operation is performed on AC, QR, and Qn+1 binary values and the SC value is decremented by one. If QnQn+1 is 01, then addition is performed. After adding the arithmetic right shift on AC, QR, and Qn+1, SC is decremented by one. If QnQn+1 is 10, then a subtraction operation is performed. After subtraction, the arithmetic right shift on AC,QR, Qn+1, and SC is decremented by one. These steps are repeated until the sequence counter(SC) reaches 0.

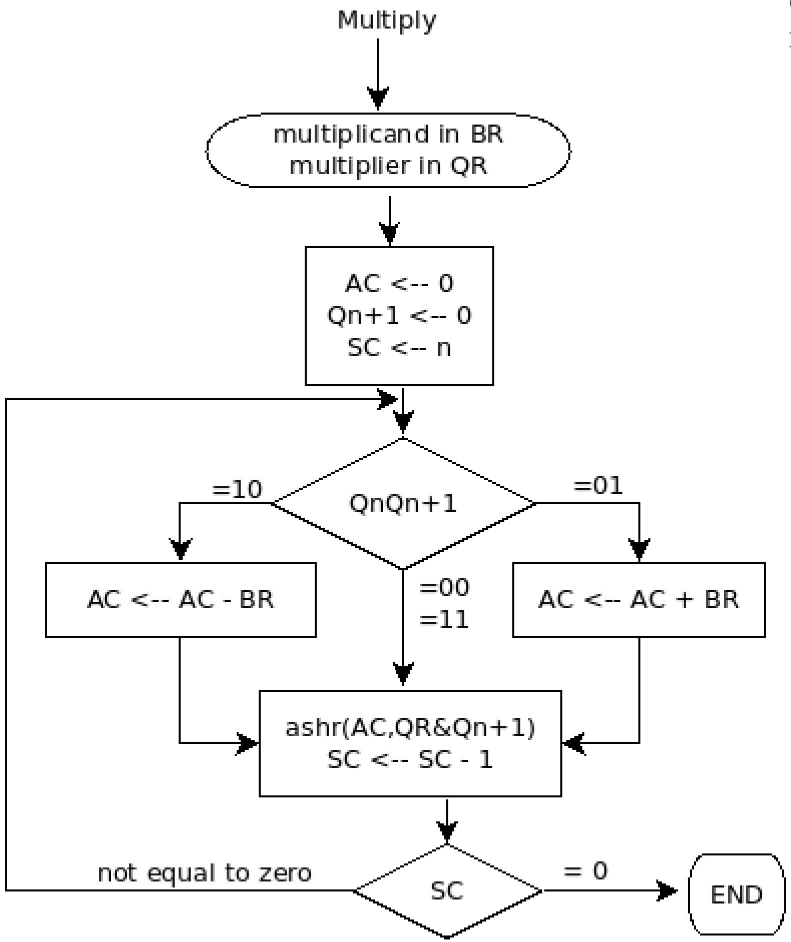

figure 8. Booth’s multiplication flowchart

Example: -5*-4, -5 is multiplicand and -4 is multiplier

Qn Qn+1	**BR=1011** **2’s comp of BR = 0101**	**AC**	**QR**	**Qn+1**	**SC**
	Initial	0000	1100	0	4
0 0	ashr(AC,QR,Qn+1)	0000	0110	0	3
0 0	ashr(AC,QR,Qn+1)	0000	0011	0	2
1 0	subtract	0101			
		0101			
	ashr(AC,QR,Qn+1)	0010	1001	1	1
1 1	ashr(AC,QR, Qn+1)	0001	0100	1	0

Table 2 booth multiplication example

In Table 2, 5 is multiplicand 2’s complement binary value of 1011 in the BR register, multiplier -4 2’s complement binary value in the QR register, AC register initial value is 0000 and SC set to 4 because in the multiplier number of bits 4. Based on the QnQn+1 value, either an addition or subtraction operation is performed, followed by an arithmetic shift right operation. If QnQn+1 is 00 or 11, in this case, the arithmetic shift right operation is performed. These steps are repeated until the SC value reaches 0. The final product is stored in the AC and QR registers.

Division Algorithms

One is the restoring method, and the other is the non-restoring method. **restoring method:** In this method, partial remendiar is restored by adding the divisor to the negative difference. In the restoring method, M is a positive divisor (n bit binary value) and Q is a positive dividend (n bit binary value).

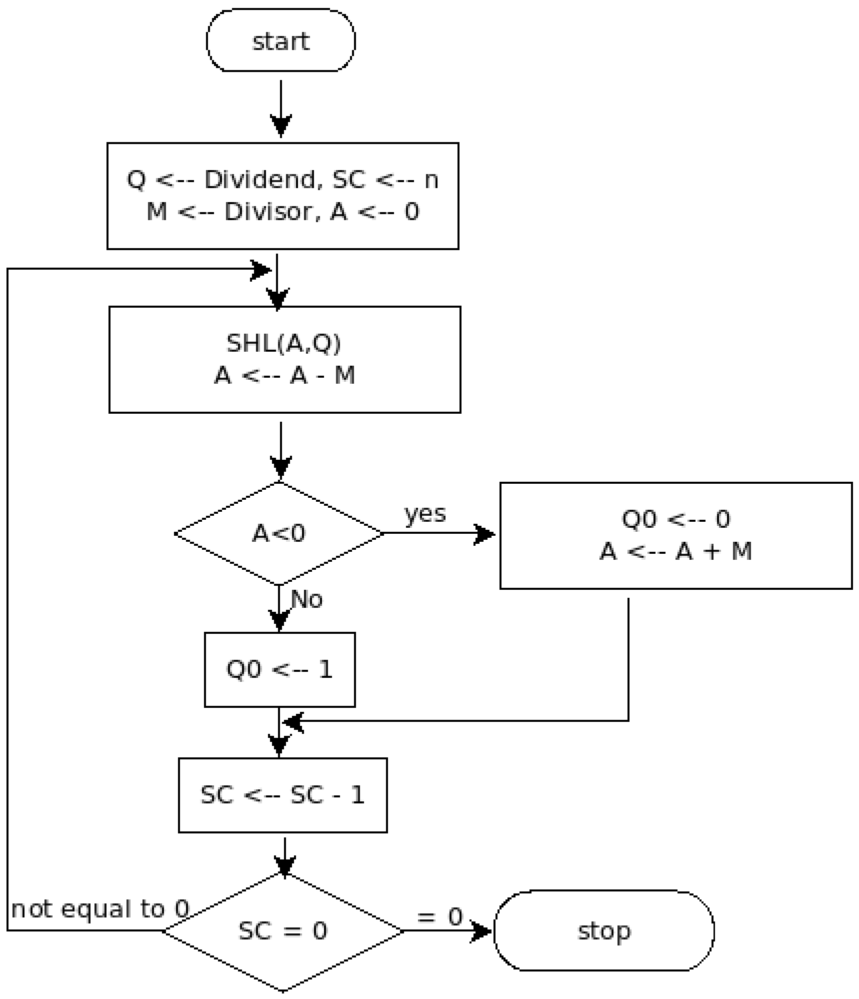

figure 8. restoring division flowchart

Figure 8 shows a shift left operation on the values of A and Q registers, followed by a partial remainder subtracted with the divisor and the result stored in A register. After subtraction, the A value is compared with zero. If A is less than zero, Q0 is set to zero, and the partial remainder is restored by adding the divisor value, which is decremented by one. If A value is greater than zero, then Q0 is set to 1 and the SC value is decremented by one . After these SC values are compared with 0, if not equal to zero, the above steps are repeated until the SC value reaches 0.

Example: 7/3 M divisor binary value is 0011, 2’s complement of M value is 1101.

Steps	A	Q	SC
Initial	0000	0111	4
Shift left (A,Q)	0000	1110	
subtract	1101		
A<0; Q0<--0	1101	1110	
add M to A	0011		
	0000	1110	3
Shift left (A,Q)	0001	1100	
subtract(A <-- A - M)	1101		
A<0 ; restore Partial Remainder	1110		
	0011		
A=A+M	0001	1100	2
Shift left (A,Q)	0011	1000	
subtract	1101		
Carry is discarded	0000		
A is not less than zero; Q0<-- 1	0000	1001	1
Shift left (A,Q)	0001	0010	
Subtract (A=A-M)	1101		
A<0 is true then Q0<--0;	1110		
Restore partial remainder	0011		
	0001	0010	0

Table 3. restoring division algorithm example

Table 3 shows the remainder in A register (0001) and the quotient in Q register (0010) after a 7/3 division operation.

Non restoring division method

M is the divisor and Q is the dividend in the non-restoring division method. Following the division operation, the remainder is stored in A register and the quotient in Q register. A register initial value is 0, and for a sequence counter, the number of bits in the divisor is the same as assigned to SC.

In Figure 9, if A is less than 0, then a shift left operation is performed after that, and the partial remainder is subtracted with the divisor. If A is greater than zero, a shift left operation is performed after the partial remainder is multiplied by the divisor. If the A value is less than zero, the last bit in the quotient register is set to 0 (q0<--0), and the SC value is decremented by one. If A value is not less than zero, then in the quotient register, the last bit is set to 1 (q0<--1) and the SC value is decremented by one. These steps are repeated until the SC value reaches zero. If the SC value is zero, the A value is checked, and if A is less than zero, the partial remainder is multiplied by the divisor. If A value is not less than zero, then the operation stops.

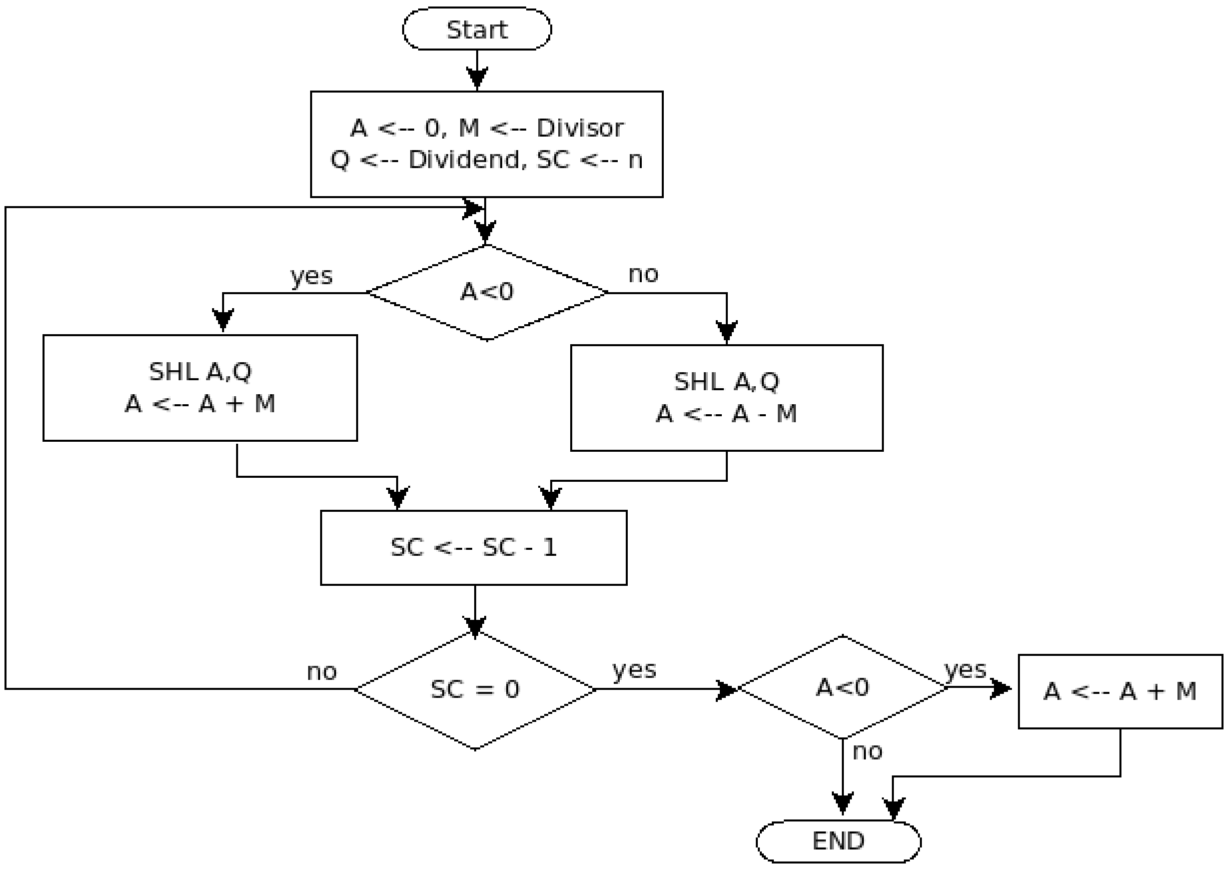

Figure 9 non restoring division algorithm flowchart

Example :7/3 M divisor binary value is 0011, 2’s complement of M is 1101.

Steps	**A**	**Q**	**SC**
Initial	0000	0111	4
Shift left (A,Q)	0000	1110	
Subtract (A=A-M)	1101		
	1101		
First bit 1 means A<0; q0<--0	1101	1110	3
Shift left (A,Q), A=A+M	1011	1100	
	0011		
A<0 is true then q0<--0	1110	1100	2
Shift left (A,Q)	1101	1000	
Add A=A+M	0011		
	0000		
A<0 is false then q0<--1	0000	1001	1
Shift left (A,Q)	0001	0010	
Subtract (A=A-M)	1101		
A<0 is true then q0<--0	1110	0010	0
If sc is 0 and A<0 is true then add A=A+M	0011		
	0001	0010	

Table 4 non restoring division example

Fraction number representation

Fraction numbers are represented in two forms: fixed point representation and floating point representation.

Fixed point representation: In fixed point representation, the decimal point is fixed. Before the decimal point, the number is converted into binary format. After the decimal point, digits are converted into binary format.

Example:

In this example, the 41.6875 decimal point is fixed. So digits before the decimal point are represented in binary format, and digits after the decimal point are converted to binary format. If the total size is 25 bits, for before decimal point numbers, it is 16 bits, 8 bits for after decimal point digits, and 1 bit for sign representation.

|---25 bits---|

1 bit for sign	16 bits for before decimal	8 bits for after decimal digits

Floating point representation

In floating point representation, the decimal point moves towards the left direction or moves towards the right direction. Based on decimal point movement, the exponent is either incremented or decremented. The drawback with fixed point representation is that we can not represent very large and very small fractional numbers. But using floating point representation, very small and very large numbers are represented. A floating point number is represented in the IEEE 754 format.

IEEE 754 single precision representation

|---32 bits---|

Sign	Exponent	Mantissa

|--1 bit ---------|---------8 bits--------------|----------------------------23 bits ----------------------------------|

In single precision representation, the first bit represents a sign bit that indicates whether a floating point number is positive or negative. With 8 bit exponent unsigned numbers, the possible range is 0 to 255, but 0 & 255 are reversed for special purposes. Aside from these two numbers, the exponent range is 1 to 254.

actual exponent = biased exponent – excess 127

Excess-127 = (28-1-1) for single precision representation with this actual exponent range from -126 (1-127) to 127 (254-127). For getting actual exponent from the biased exponent excess -127 is subtracted.

Example : -14.25 in single precision representation

- 14.25's binary value is 1110.01 and this binary number with exponent is 1110.01*20 .
- Normalization format is 1.M*2(exponent – 127) above number in normalization form is 1.11001*23. Here the actual exponent is 3, but we need the exponent in an 8-bit biased exponent format.
- Biased exponent = actual exponent + excess 127, i.e biased exponent = 3+127=130. The biased exponent in 8 bit binary format is 10000010 and the sign bit is 1.
- -14.25 in IEEE 754 single precision format is 1 10000010 11001000000000000000000.

IEEE 754 Double precision representation

In IEEE 754 double precision floating point representation, a 64-bit system is used. The first bit indicates a sign bit; 11 bits for the exponent; and 52 bits for the mantissa.

|---------------------------------------64 bits---|

Sign	Exponent	Mantissa

|--1bit-----|---------11 bits ------------------|------------------------52 bits ------------------------------------|

Biased exponent =actual exponent + excess 1023.

So excess =211-1-1=1023. The biased exponent unsigned range is 0 to 2048, but 0 & 2048 are reserved for special purposes. After excluding 0 & 2048, the remaining range is 1 to 2047. If excess value is subtracted, then the range is-1022 to 1023.