Arithmetic and logic Unit (ALU)

Arithmetic and logic Unit (ALU)

ALU is responsible to perform the operation in the computer.

The basic operations are implemented in hardware level. ALU is having collection of two types of operations:

- Arithmetic operations
- Logical operations

Consider an ALU having 4 arithmetic operations and 4 logical operations.

To identify any one of these four logical operations or four arithmetic operations, two control lines are needed. Also to identify the any one of these two groups- arithmetic or logical, another control line is needed. So, with the help of three control lines, any one of these eight operations can be identified.

Consider an ALU is having four arithmetic operations. Addition, subtraction, multiplication and division. Also consider that the ALU is having four logical operations: OR, AND, NOT & EX-OR.

We need three control lines to identify any one of these operations. The input combination of these control lines are shown below:
Control line C_2 is used to identify the group: logical or arithmetic, i.e.
$C_2 = 0$: arithmetic operation $C_2 = 1$: logical operation.
Control lines C_0 and C_1 are used to identify any one of the four operations in a group. One possible combination is given here.

C_1	C_0	Arithmetic $C_2 = 0$	Logical $C_2 = 1$
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	**EX-OR**

A 3×8 decode is used to decode the instruction. The block diagram of the ALU is shown in figure 2.1.

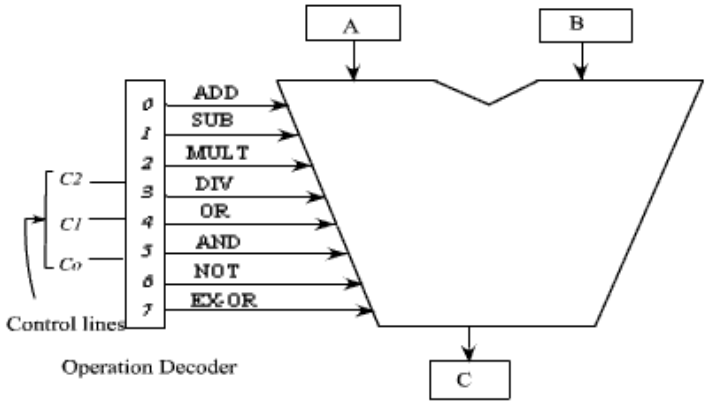


Figure 2.1: Block Diagram of the ALU

The ALU has got two input registers named as A and B and one output storage register, named as C. It performs the operation as:

$$C = A \text{ op } B$$

The input data are stored in A and B, and according to the operation specified in the control lines, the ALU perform the operation and put the result in register C.

As for example, if the contents of controls lines are, 000, then the decoder enables the addition operation and it activates the adder circuit and the addition operation is performed on the data that are available in storage register A and B . After the completion of the operation, the result is stored in register C.

We should have some hardware implementations for basic operations. These basic operations can be used to implement some complicated operations which are not feasible to implement directly in hardware.

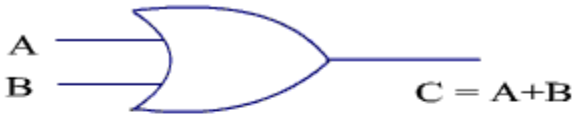
There are several logic gates exists in digital logic circuit. These logic gates can be used to implement the logical operation. Some of the common logic gates are mentioned here.

AND gate: The output is high if both the inputs are high. The AND gate and its truth table is shown in Figure 2.2.

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.2: AND gate and its truth table.

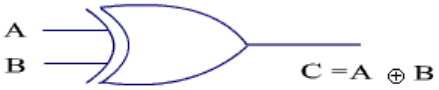
OR gate: The output is high if any one of the input is high. The OR gate and its truth table is shown in Figure 2.3.



A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2.3: OR gate and its truth table.

EX-OR gate: The output is high if either of the input is high. The EX-OR gate and its truth table is given in Figure 2.4.



A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2.4: EX-OR gate and its truth table.

If we want to construct a circuit which will perform the AND operation on two 4-bit number, the implementation of the 4-bit AND operation is shown in the Figure-2.5.

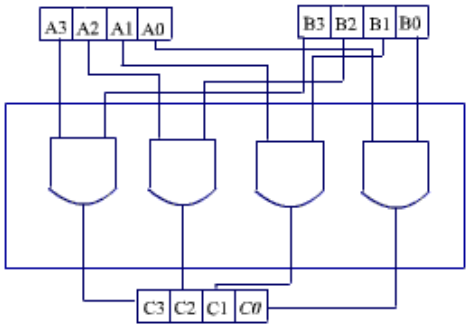


Figure2.5: 4-bit AND operator

Arithmetic Circuit

Binary Adder:

Binary adder is used to add two binary numbers.

In general, the adder circuit needs two binary inputs and two binary outputs. The input variables designate the augends and addend bits; the output variables produce the sum and carry.

The binary addition operation of single bit is shown in the truth table

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

C: Carry Bit

S: Sum Bit

The simplified sum of products expressions are

$$S = x'y + xy'$$
$$C = xy$$

The circuit implementation is

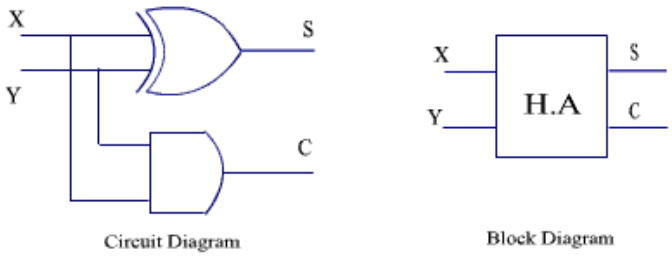


Figure 2.6: Circuit diagram and Block diagram of Half Adder

This circuit cannot handle the carry input, so it is termed as **half adder**. The circuit diagram and block diagram of Half Adder is shown in Figure 2.6.

Full Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs.

Two of the input variables, denoted by *x* and *y*, represent the two bits to be added. The third input *Z*, represents the carry from the previous lower position. The two outputs are designated by the symbols *S* for sum and *C* for carry.

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The simplified expression for *S* and *C* are

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$
$$= xy + xy'z + x'yz$$

We may rearrange these two expressions as follows:

$$S = z \oplus (x \oplus y)$$
$$= z'(xy' + x'y) + z(xy' + x'y)'$$
$$= z'(xy' + x'y) + z(xy + x'y')$$
$$= xy'z' + x'yz' + xyz + x'y'z$$
$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

The circuit diagram full adder is shown in the figure.

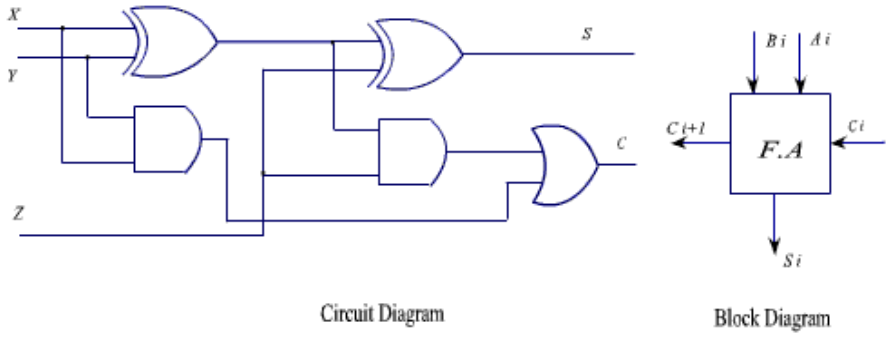


Figure 2.7: Circuit diagram and block diagram of Full Adder

The circuit diagram and block diagram of a Full Adder is shown in the Figure 2.7. *n*-such single bit full adder blocks are used to make *n*-bit full adder.

To demonstrate the binary addition of four bit numbers, let us consider a specific example.

Consider two binary numbers

A =1 0 0 1 B = 0 0 1 1

Subscript	i	3	2	1	0
Input carry	C_i	0	1	1	0
Augend	A_i	1	0	0	1
Addend	B_i	0	0	1	1
Sum	S_i	1	1	0	0
Output Carry	C_{i+1}	0	0	1	1

To get the four bit adder, we have to use 4 full adder block. The carry output the lower bit is used as a carry input to the next higher bit.

The circuit of 4-bit adder shown in the Figure 2.8.

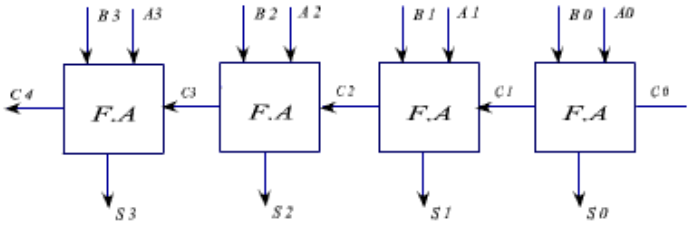


Figure 2.8: A 4-bit adder circuit.

Binary Subtractor:

The subtraction operation can be implemented with the help of binary adder circuit, because

$$A - B = A + (-B)$$

We know that 2's complement representation of a number is treated as a negative number of the given number.

We can get the 2's complements of a given number by complementing each bit and adding 1 to it.

The circuit for subtracting A-B consist of an added with inverter placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when performing subtraction.

The operation thus performed becomes A, plus the 1's complement of B, plus 1. This is equal to A plus 2's complement of B.

With this principle, a single circuit can be used for both addition and subtraction. The 4 bit adder subtractor circuit is shown in the figure. It has got one mode (M) selection input line, which will determine the operation,

If M = 0, then A + B

If M = 1, Then A – B = A + (-B) = A + 1's complement of B+1

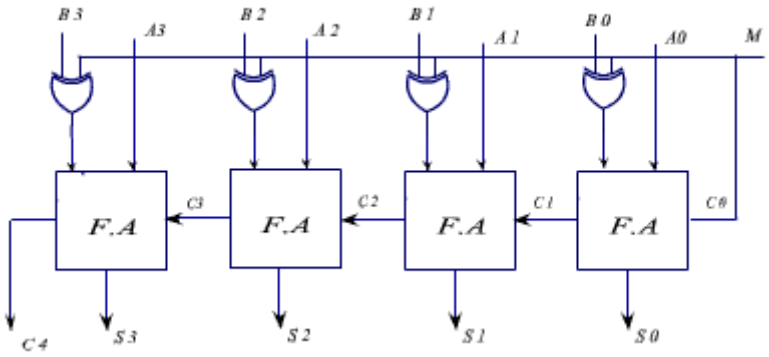


Figure 2.9: 4-bit adder subtractor

The circuit diagram of a 4-bit adder substractor is shown in the Figure 2.9.

The operation of OR gate:

$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$

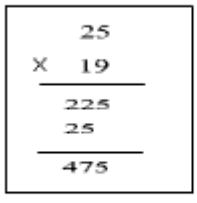
If M = 0, $B_i \oplus 0 = B_i$

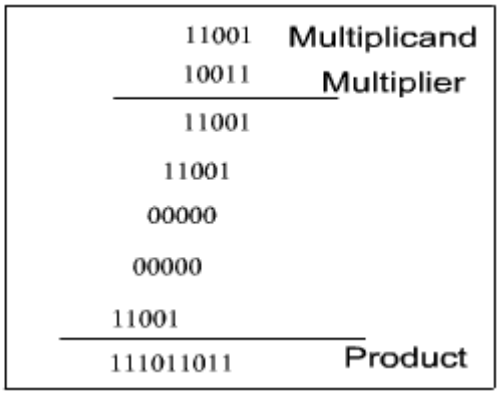
If M=1, $B_i \oplus 1 = \bar{B}_i$

Multiplication

Multiplication of two numbers in binary representation can be performed by a process of SHIFT and ADD operations. Since the binary number system allows only 0 and 1's, the digit multiplication can be replaced by SHIFT and ADD operation only, because multiplying by 1 gives the number itself and multiplying by 0 produces 0 only.

The multiplication process is illustrated with a numerical example.





The process consists of looking at successive bits of the multiplier, least significant bit first. If the multiplier bit is a 1, the multiplicand is copied down; otherwise, zeros are copied down. The numbers copied down in successive lines are shifted one position to the left from the previous number. Finally, the numbers are added and their sum forms the product.

When multiplication is implemented in a digital computer, the process is changed slightly.

Instead of providing registers to store and add simultaneously as many binary numbers as there are bits in the multiplier, it is convenient to provide an adder for the summation of only two binary numbers and successively accumulate the partial products in a register. It will reduce the requirements of registers.

Instead of sifting the multiplicand to the left, the partial product is shifted to right.

When the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product.

An algorithm to multiply two binary numbers. Consider that the ALU does not provide the multiplication operation, but it is having the addition operation and shifting operation. Then we can write a micro program for multiplication operation and provide the micro program code in memory. When a multiplication operation is encountered, it will execute this micro code to perform the multiplication.

The micro code is nothing but the collection of some instructions. ALU must have those operation; otherwise we must have micro code again for those operations which are not supported in ALU.

Consider a situation such that we do not have the multiplication operation in a primitive computer. Is it possible to perform the multiplication? Of course, yes, provided the addition operation is available.

We can perform the multiplication with the help of repeated addition method; for example, if we want to multiply 4 by 5 (4×5), then simply add 4 five times to get the result.

If it is possible by addition operation, then why we need a multiplication operation.

Consider a machine, which can handle 8 bit numbers, then we can represent number from 0 to 255. If we want to multiply 175×225, then there will be at least 175 addition operation.

But if we use the multiplication algorithm that involves shifting and addition, it can be done in 8 steps, because we are using an 8-bit machine.

Again, the micro program execution is slightly slower, because we have to access the code from micro controller memory, and memory is a slower device than CPU.

It is possible to implement the multiplication algorithm in hardware.

Binary Multiplier, Hardware Implementation

The block diagram of binary multiplier is shown in the Figure 2.10...

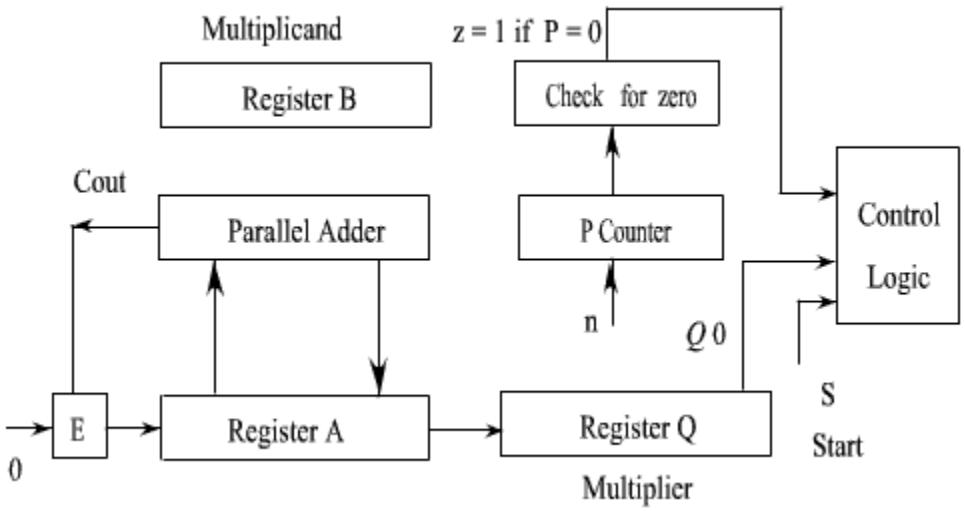


Figure 2.10: Block diagram of binary multiplier

The multiplicand is stored in register B and the multiplier is stored in register Q.

The partial product is formed in register A and stored in A and Q.

The counter P is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.

Initially, the multiplicand is in register B and the multiplier in Q. The register A is reset to 0.

The sum of A and B forms a partial product- which is transferred to the EA register.

Both partial product and multiplier are shifted to the right. The least significant bit of A is shifted into the most significant position of Q; and 0 is shifted into E.

After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right.

The right most flip flop in register Q, designated by Q_0 will hold the bit of the multiplier which must be inspected next. If the content of this bit is 0, then it is not required to add the multiplicand, only shifting is needed. If the content of this bit is 1, then both addition and shifting are needed.

After each shifter, value of counter P is decremented and the process continues till the counter value becomes 0.

The final result is available in (EAQ) registers combination.

To control the operation, it is required to design the appropriate control logic that is shown in the block diagram.

The flow chart of the multiplication operation is given in the Figure 2.11.

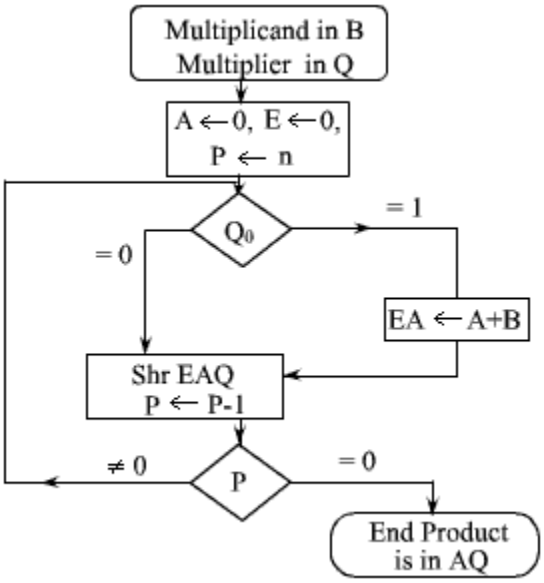


Figure 2.11: Flow chart of the multiplication operation

The working of multiplication algorithm is shown here with the help of an example.

Multiplicand B = 11001

	E	A	Q	P
Multiplier in Q	0	0 0 0 0 0	1 0 0 1 1	5
$Q_0 = 1$, Add B		1 1 0 0 1		
		---		
first partial product	0	1 1 0 0 1		
Shr EAQ	0	0 1 1 0 0	1 1 0 0 1	4
$Q_0 = 1$,Add B		1 1 0 0 1		

Second partial product	1	0 0 1 0 1		
Shr EAQ	0	1 0 0 1 0	1 1 1 0 0	3
$Q_0 = 0$, Shr EAQ	0	0 1 0 0 1	0 1 1 1 0	2
$Q_0 = 0$, Shr EAQ	0	0 1 0 0 1	0 1 1 1 0	1
$Q_0 = 1$, Add B		1 1 0 0 1		
		---		
fifth partial product	0	1 1 1 0 1		
Shr EAQ	0	0 1 1 1 0	1 1 0 1 1	0
Stop				
Final products:	0	1 1 1 0	1 1 0 1 1	