UNIT 3 ALU ORGANISATION

Structure **Page No.**

3.0 INTRODUCTION

By now we have discussed the instruction sets and register organisation followed by a discussion on micro-operations and instruction execution. In this unit, we will first discuss the ALU organisation. Then we will discuss the floating point ALU and arithmetic co-processors, which are commonly used for floating point computations.

This unit provides a detailed view on implementation of simple micro-operations that include register–transfer, arithmetic, logic and shift micro-operation. Finally, the construction of a simple ALU is given. Thus, this unit provides you the basic insight into the computer system. The next unit covers details of the control unit. Together these units describe the two most important components of CPU: the ALU and the CU.

3.1 OBJECTIVES

After going through this unit, you will be able to:

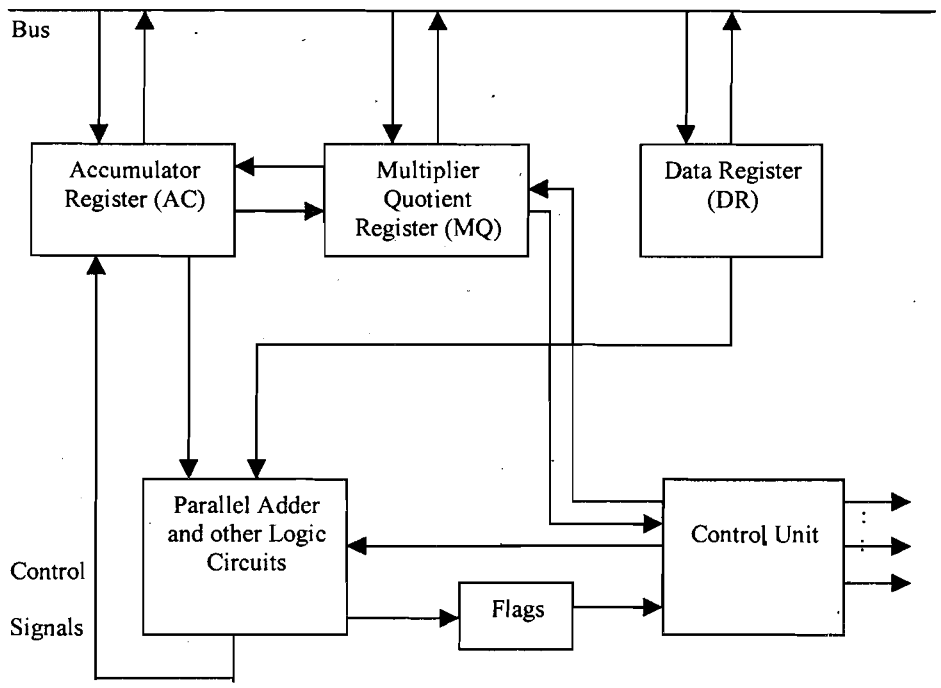
- describe the basic organisation of ALU;
- discuss the requirements of a floating point ALU;
- define the term arithmetic coprocessor; and
- create simple arithmetic logic circuits.

3.2 ALU ORGANISATION

As discussed earlier, an ALU performs simple arithmetic-logic and shift operations. The complexity of an ALU depends on the type of instruction set which has been realized for it. The simple ALUs can be constructed for fixed-point numbers. On the other hand the floating-point arithmetic implementation requires more complex control logic and data processing capabilities, i.e., the hardware. Several micro-processor families utilize only fixed-point arithmetic capabilities in the ALUs. For floating point arithmetic or other complex functions they may utilize an auxiliary special purpose unit. This unit is called arithmetic co-processor. Let us discuss all these issues in greater detail in this section.

3.2.1 A Simple ALU Organisation

An ALU consists of circuits that perform data processing micro-operations. But how are these ALU circuits used in conjunction of other registers and control unit? The simplest organisation in this respect for fixed point ALU was suggested by John von Neumann in his IAS computer design (Please refer to Figure 1).



Bus
Accumulator Register (AC)
Multiplier Quotient Register (MQ)
Data Register (DR)
Parallel Adder and other Logic Circuits
Control Unit
Flags
Control Signals

Figure 1: Structure of a Fixed point Arithmetic logic unit

The above structure has three registers AC, MQ and DR for data storage. Let us assume that they are equal to one word each. Please note that the Parallel adders and other logic circuits (these are the arithmetic, logic circuits) have two inputs and only one output in this diagram. It implies that any ALU operation at most can have two input values and will generate single output along with the other status bits. In the present case the two inputs are AC and DR registers, while output is AC register. AC and MQ registers are generally used as a single AC.MQ register. This register is capable of left or right shift operations. Some of the micro-operations that can be defined on this ALU are:

Addition	: AC ← AC + DR
Subtraction	: AC ← AC – DR
AND	: AC ← AC ^ DR
OR	: AC ← AC v DR
Exclusive OR	: AC ← AC (+) DR
NOT	: AC ← AC

In this ALU organisation multiplication and division were implemented using shift-add/subtract operations. The MQ (Multiplier-Quotient register) is a special register used for implementation of multiplication and division. We are not giving the details of how this register can be used for implementing multiplication and division algorithms. For more details on these algorithms please refer to further readings. One such algorithm is Booth's algorithm and you must refer to it in further readings.

For multiplication or division operations DR register stores the multiplicand or divisor respectively. The result of multiplication or division on applying certain algorithm can finally be obtained in AC.MQ register combination. These operations can be represented as:

Multiplication : $AC.MQ \leftarrow DR \times MQ$

Division : $AC.MQ \leftarrow MQ \div DR$

DR is another important register, which is used for storing second operand. In fact it acts as a buffer register, which stores the data brought from the memory for an instruction. In machines where we have general purpose registers any of the registers can be utilized as AC, MQ and DR.

Bit Slice ALUs

It was feasible to manufacture smaller such as 4 or 8 bits fixed point ALUs on a single IC chip. If these chips are designed as expendable types then using these 4 or 8 bit ALU chips we can make 16, 32, 64 bit array like circuits. These are called bit- slice ALUs.

The basic advantage of such ALUs is that these ALUs can be constructed for a desired word size. More details on bit-slice ALUs can be obtained from further readings.

Check Your Progress 1

State True or False

	T	F
1. A multiplication operation can be implemented as a logical operation.	☐	
2. The multiplier-quotient register stores the remainder for a division operation.	☐	
3. A word is processed sequentially on a bit slice ALU.	☐	

3.2.2 A Sample ALU Design

The basis of ALU design starts with the micro-operation implementation. So, let us first explain how the bus can be used for Data transfer micro-operations.

A digital computer has many registers, and rather than connecting wires between all registers to transfer information between them, a common bus is used. Bus is a path (consists of a group of wires) one for each bit of a register, over which information is transferred, from any of several sources to any of several destinations. In general the size of this data bus should be equal to the number of bits in a general purpose register.

A register is selected for the transfer of data through bus with the help of control signals. The common data transfer path, that is the bus, is made using the multiplexers. The select lines are connected to the control inputs of the multiplexers and the bits of one register are chosen thus allowing multiplexers to select a specific source register for data transfer.

The construction of a bus system for four registers using 4×1 multiplexers is shown below. Each register has four bits, numbered 0 through 3. Each multiplexer has 4 data inputs, numbered 0 through 3, and two control or selection lines, C_0 and C_1. The data inputs of 0th MUX are connected to the corresponding 0th input of every register to form four lines of the bus. The 0th multiplexer multiplexes the four 0th bits of the registers, and similarly for the three other multiplexers.

Since the same selection lines C_0 and C_1 are connected to all multiplexers, therefore they choose the four bits of one register and transfer them into the four-line common bus.

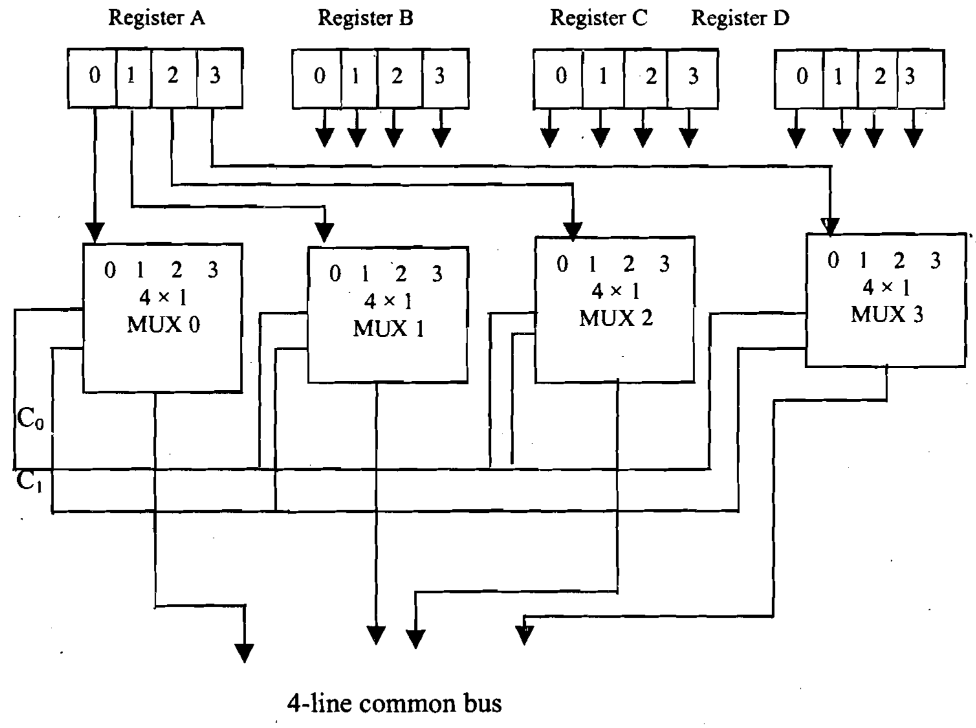


Figure 2: Implementation of BUS

When C_1 C_0 = 00, the 0th data input of all multiplexers are selected and this causes the bus lines to receive the content of register A since the outputs of register A are connected to the 0th data inputs of the multiplexers which is then applied to the output that forms the bus. Similarly, when C_1 C_0 = 01, register B is selected, and so on. The following table shows the register that is selected for each of the four possible values of the selection lines:

C_1	C_0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

Figure 3: Bus Line Selection

To construct a bus for 8 registers of 16 bits each, you would require 16 multiplexers, one for each line in the bus. The number of multiplexers needed to construct the bus is equal to the number of bits in each register. Each multiplexer must have eight data input lines and three selection lines ($2^3 = 8$) to multiplex one bit in the eight registers.

Implementation of Arithmetic Circuits for Arithmetic Micro-operation

An arithmetic circuit can be implemented using a number of full adder circuits or parallel adder circuits. Figure 4 shows a logical implementation of a 4-bit arithmetic circuit. The circuit is constructed by using 4 full adders and 4 multiplexers.

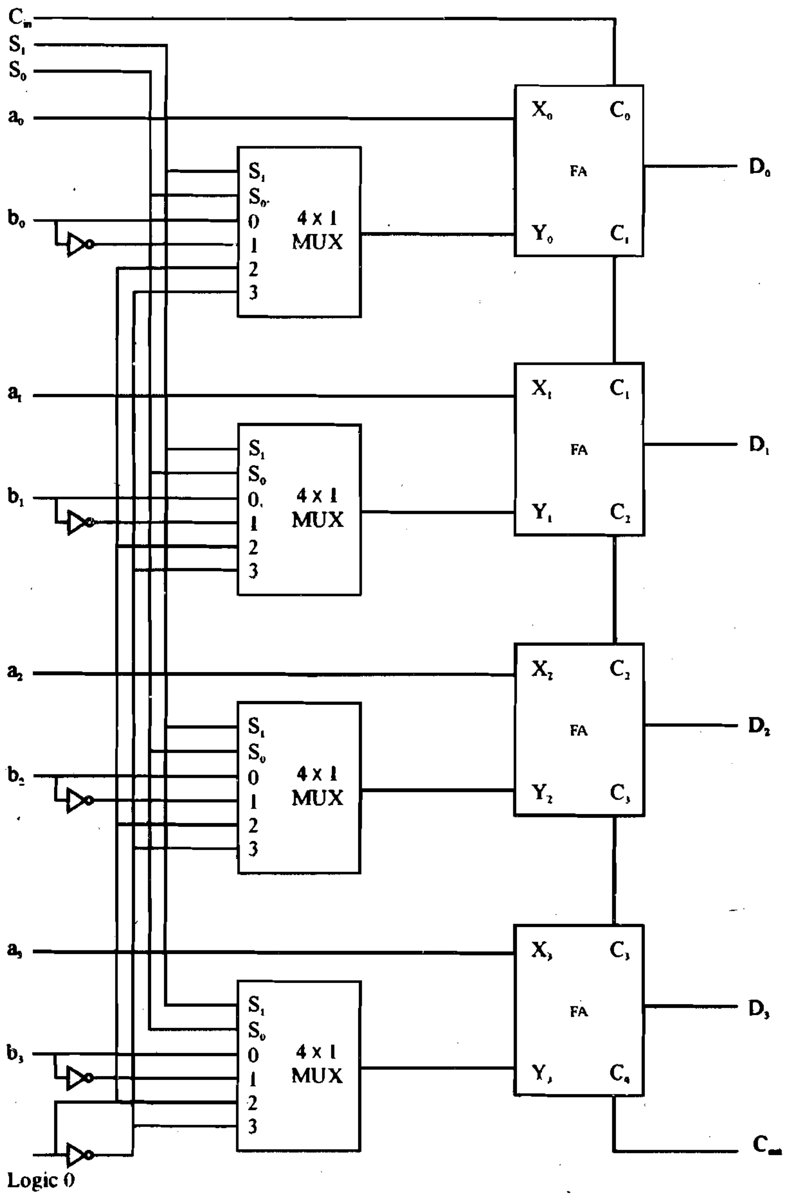


Figure 4: A Four-bit arithmetic circuit

The diagram of a 4-bit arithmetic circuit has four 4×1 multiplexers and four full adders (FA). Please note that the FULL ADDER is a circuit that can add two input bits and a carry-in bit to produce one sum-bit and a carry-out-bit.

So what does the adder do? It just adds three bits. What does the multiplexer do? It controls one of the input bits. Thus, such combination produces a series of micro-operations.

Let us find out how the multiplexer control lines will change one of the Inputs for Adder circuit. Please refer to the following table. (Please note the convention VALID ONLY FOR THE TABLE are that an uppercase alphabet indicates a Data Word, whereas the lowercase alphabet indicates a bit.)

Control Input		Output of 4 × 1 Multiplexers				Y input to Adder	Comments
S_1	S_0	MUX(a)	MUX(b)	MUX(c)	MUX(d)		
0	0	b_0	b_1	b_2	b_3	B	The data word B is input to Full Adders
0	1	$\overline{b_0}$	$\overline{b_1}$	$\overline{b_2}$	$\overline{b_3}$	$\overline{B}$	1's complement of B is input to Full Adders
1	0	0	0	0	0	0	Data word 0 is input to Full Adders
1	1	1	1	1	1	F_H	Data word 1111 = F_H is input to Full Adders

Figure 5: Multiplexer Inputs and Output of the Arithmetic Circuit of Figure 4

Now let us discuss how by coupling carry bit (C_{in}) with these input bits we can obtain various micro-operations.

Input to Circuits

- Register A bits as a_0, a_1, a_2 and a_3 in the corresponding X bits of the Full Adder (FA).
- Register B bits as given in the Figure 5 above as in the corresponding Y bits of the FA.
- Please note each bit of register A and register B is fed to different full adder unit.
- Please also note that each of the four inputs from A are applied to the X inputs of the binary adder and each of the four inputs from B are connected to the data inputs of the multiplexers. It means that the A input directly goes to adder but B input can be manipulated through the Multiplexer to create a number of different input values as given in the figure above. The B inputs through multiplexers are controlled by two selection lines S_1 and S_0. Thus, using various combinations of S_1 and S_0 we can select data bits of B, complement of B, 0 word, or word having All 1's.
- The input carry C_{in}, which can be equal to 0 or 1, goes to the carry input of the full adder in the least significant position. The other carries are cascaded from one stage to the next. Logically it is the same as that of addition performed by us. We do pass the carry of lower digits addition to higher digits. The output of the binary adder is determined from the following arithmetic sum:

$$D = X + Y + C_{in}$$

OR

$$D = A + Y + C_{in}$$

By controlling the value of Y with the two selection lines S_1 and S_0 and making C_{in} equal to 0 or 1, it is possible to implement the eight arithmetic micro-operations listed in the truth table.

S_1	S_0	C_{in}	Y val	$D = A+Y+C_{in}$	Equivalent Micro-Operation	Micro-Operation Name
0	0	0	B	$D = A + B$	$R \leftarrow R1 + R2$	Add
0	0	1	B	$D = A + B + 1$	$R \leftarrow R1 + R2 + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A+B$	$R \leftarrow R1 + \overline{R2}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	$R \leftarrow R1$ + 2's complement of R2	Subtract
1	0	0	0	$D = A$	$R \leftarrow R1$	Transfer
1	0	1	0	$D = A + 1$	$R \leftarrow R1 + 1$	Increment
1	1	0	1	$D = A - 1$	$R \leftarrow R1$ + (All 1s)	Decrement
1	1	1	1	$D = A$	$R \leftarrow R1$	Transfer

Figure 6: Arithmetic Circuit Function Table

Let us refer to some of the cases in the table above.

When $S_1S_0 = 00$, input line B is enabled and its value is applied to the Y inputs of the full adder. Now,

If input carry $C_{in} = 0$, the output will be $D = A + B$
If input carry $C_{in} = 1$, the output will be $D = A + B + 1$.

When $S_1S_0 = 01$, the complement of B is applied to the Y inputs of the full adder. So If $C_{in} = 1$, then output $D = A + \overline{B} + 1$. This is called subtract micro-operation. (Why?)

Reason: Please observe the following example, where A = 0111 and B=0110, then $\overline{B}$ =1001. The sum will be calculated as:

```
    0111        (Value of A)
    1001        ( Complement of B)
  1 0000 + (Carry in =1) = 0001
```

Ignore the carry out bit. Thus, we get simple subtract operation.

If $C_{in} = 0$, then $D = A + \overline{B}$. This is called subtract with borrow micro-operation. (Why?). Let us look into the same addition as above:

```
    0111       (Value of A)
    1001       ( Complement of B)
  1 0000 + (Carry in =0) = 0000
```

This operation, thus, can be considered as equivalent to:

$D = A + \overline{B}$
=> $D = (A - 1) + (\overline{B} + 1)$
=> $D = (A - 1)$ + 2's complement of B
=> $D = (A - 1) - B$ Thus, is the name complement with Borrow

When $S_1S_2 = 10$, input value 0 is applied to Y inputs of the full adder.

If $C_{in} = 0$, then output $D = A + 0 + C_{in}$ => $D = A$
If $C_{in} = 1$, then $D = A + 0 + 1$ => $D = A + 1$

The first is a simple data transfer micro-operation; while the second is an increment micro-operation.

When $S_1S_2 = 11$, input word all 1's is applied to Y inputs of the full adder.

If $C_{in} = 0$, then output $D = A + \text{All (1s)} + C_{in} \Rightarrow D = A - 1$ (How? Let us explain with the help of the following example).

Example: Let us assume that the Register A is of 4 bits and contains the value 0101 and it is added to an all (1) value as:

$$\begin{array}{r} \mathbf{0101} \\ 1111 \\ \hline \mathbf{1\ \ 0100} \\ \hline \end{array}$$

The 1 is carry out and is discarded. Thus, on addition with all (1's) the number has actually got decremented by one.

If $C_{in} = 1$, then $D = A + \text{All(1s)} + 1 \Rightarrow D = A$

The first is the decrement micro-operation; while the second is a data transfer micro-operation.

Please note that the micro-operation D = A is generated twice, so there are only seven distinct micro-operations possible through the proposed arithmetic circuit.

Implementation of Logic Micro-operations

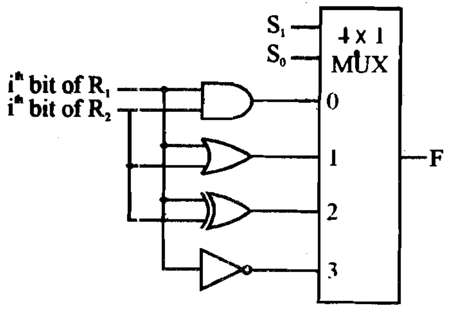
For implementation, let us first ask the questions how many logic operations can be performed with two binary variables. We can have four possible combinations of input of two variables. These are 00, 01, 10, and 11. Now, for all these 4 input combinations we can have $2^4 = 16$ output combinations of truth-values for a function. This implies that for two variables we can have 16 logical operations. The above stated fact will be clearer by going through the following figure.

I_3	I_2	I_1	I_0	Function	Operation	Comments
0	0	0	0	$F_0 = 0$	$R \leftarrow 0$	Clear
0	0	0	1	$F_1 = x.\ y$	$R \leftarrow R_1 \wedge R_2$	AND
0	0	1	0	$F_2 = x.\ \overline{y}$	$R \leftarrow R_1 \wedge \overline{R_2}$	R_1 AND with complement R_2
0	0	1	1	$F_3 = x$	$R \leftarrow R_1$	Transfer of R_1
0	1	0	0	$F_4 = \overline{x}\ .\ y$	$R \leftarrow \overline{R_1} \wedge R_2$	R_2 AND with complement R_1
0	1	0	1	$F_5 = y$	$R \leftarrow R_2$	Transfer of R_2
0	1	1	0	$F_6 = x \oplus y$	$R \leftarrow R_1 \oplus R_2$	Exclusive OR
0	1	1	1	$F_7 = x + y$	$R \leftarrow R_1 \vee R_2$	OR
1	0	0	0	$F_8 = \overline{(x + y)}$	$R \leftarrow \overline{(R_1 \vee R_2)}$	NOR
1	0	0	1	$F_9 = \overline{(x \oplus y)}$	$R \leftarrow \overline{(R_1 \oplus R_2)}$	Exclusive NOR
1	0	1	0	$F_{10} = \overline{y}$	$R \leftarrow \overline{R_2}$	Complement of R_2
1	0	1	1	$F_{11} = x + \overline{y}$	$R \leftarrow R_1 \vee \overline{R_2}$	R_1 OR with complement R_2
1	1	0	0	$F_{12} = \overline{x}$	$R \leftarrow \overline{R_1}$	Complement of R_1
1	1	0	1	$F_{13} = \overline{x} + y$	$R \leftarrow \overline{R_1} \vee R_2$	R_2 OR with complement R_1
1	1	1	0	$F_{14} = \overline{(x.y)}$	$R \leftarrow \overline{(R_1 \wedge R_2)}$	NAND
1	1	1	1	$F_{15} = 1$	$R \leftarrow$ All 1's	Set all the Bits to 1

Figure 7: Logic micro-operations on two inputs

Please note that in the figure above the micro-operations are derived by replacing the x and y of Boolean function with registers R1 and R2 on each corresponding bit of the registers R1 and R2. Each of these bits will be treated as binary variables.

In many computers only four: AND, OR, XOR (exclusive OR) and complement micro-operations are implemented. The other 12 micro-operations can be derived from these four micro-operations. Figure 8 shows one bit, which is the ith bit stage of the four logic operations. Please note that the circuit consists of 4 gates and a 4 × 1 MUX. The ith bits of Register R1 and R2 are passed through the circuit. On the basis of selection inputs S_0 and S_1 the desired micro-operation is obtained.



(a) Logic Diagram

S_1	S_0	Output	The Operation
0	0	$F = R_1 \wedge R_2$	AND Operation
0	1	$F = R_1 \vee R_2$	OR Operation
1	0	$F = R_1 \oplus R_2$	XOR Operation
1	1	$F = \overline{R_1}$	Complement of Register R_1

(b) Functional representation

Figure 8: Logic diagram of one stage of logic circuit

Implementation of a Simple Arithmetic, Logic and Shift Unit

So, by now we have discussed how the arithmetic and logic micro-operations can be implemented individually. If we combine these two circuits along with shifting logic then we can have a possible simple structure of ALU. In effect ALU is a combinational circuit whose inputs are contents of specific registers. The ALU performs the desired micro-operation as determined by control signals on the input and places the results in an output or destination register. The whole operation of ALU can be performed in a single clock pulse, as it is a combinational circuit. The shift operation can be performed in a separate unit but sometimes it can be made as a part of overall ALU. The following figure gives a simple structure of one stage of an ALU.

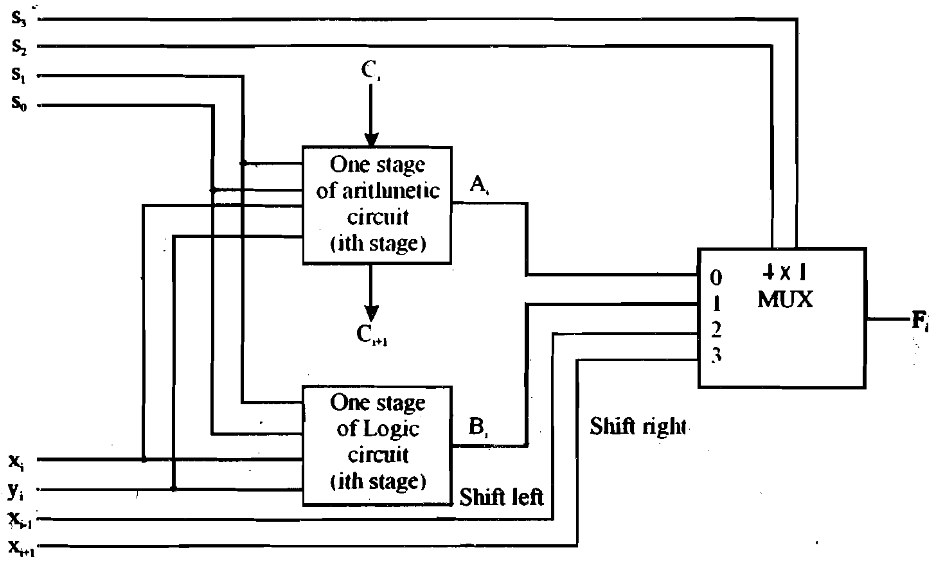


Figure 9: One stage of ALU with shift capability

Please note that in this figure we have given reference to two previous figures for arithmetic and logic circuits. This stage of ALU has two data inputs; the ith bits of the registers to be manipulated. However, the (i – 1)th or (i+1)th bit is also fed for the case of shift micro-operation of only one register. There are four selection lines, which determine what micro-operation (arithmetic, logic or shift) on the input. The F_i is the resultant bit after desired micro-operation. Let us see how the value of F_i changes on the basis of the four select inputs. This is shown in Figure 10:

Please note that in Figure 10 arithmetic micro-operations have both S_3 and S_2 bits as zero. Input C_i is important for only arithmetic micro-operations. For logic micro-operations S_3, S_2 values are 01. The values 10 and 11 cause shift micro-operations. For this shift micro-operation S_1 and S_0 values and C_i values do not play any role.

S_3	S_2	S_1	S_0	C_i	F	Micro-operation	Name	
0	0	0	0	0	$F = x$	$R \leftarrow R_1$	Transfer	Arithmetic Micro-operation
0	0	0	0	1	$F = x+1$	$R \leftarrow R_1+1$	Increment	
0	0	0	1	0	$F = x+y$	$R \leftarrow R_1+R_2$	Addition	
0	0	0	1	1	$F = x+y+1$	$R \leftarrow R_1+R_2+1$	Addition with carry	
0	0	1	0	0	$F = x+\overline{y}$	$R \leftarrow R_1+\overline{R_2}$	Subtract with borrow	
0	0	1	0	1	$F = x+(\overline{y}+1)$	$R \leftarrow R_1 - R_2$	Subtract	
0	0	1	1	0	$F = x - 1$	$R \leftarrow R_1 - 1$	Decrement	
0	0	1	1	1	$F = x$	$R \leftarrow R_1$	Transfer	
0	1	0	0	-	$F = x.y$	$R \leftarrow R_1 \wedge R_2$	AND	Logic Micro-operation
0	1	0	1	-	$F = x+y$	$R \leftarrow R_1 \vee R_2$	OR	
0	1	1	0	-	$F = x \oplus y$	$R \leftarrow R_1 \oplus R_2$	Exclusive OR	
0	1	1	1	-	$F = \overline{x}$	$R \leftarrow \overline{R_1}$	Complement	
1	0	-	-	-	$F = Shl(x)$	$R \leftarrow Shl(R_1)$	Shift left	Shift Micro-operations
1	1	-	-	-	$F = Shr(y)$	$R \leftarrow Shr(R_1)$	Shift right	

Figure 10: Micro-operations performed by a Sample ALU

3.3 ARITHMETIC PROCESSORS

The questions in this regard are: "What is an arithmetic processor?" and, "What is the need for arithmetic processors?"

A typical CPU needs most of the control and data processing hardware for implementing non-arithmetic functions. As the hardware costs are directly related to chip area, a floating point circuit being complex in nature is costly to implement. They need not be included in the instruction set of a CPU. In such systems, floating-point operations were implemented by using software routines.

This implementation of floating point arithmetic is definitely slower than the hardware implementation. Now, the question is whether a processor can be constructed only for arithmetic operations. A processor, if devoted exclusively to arithmetic functions, can be used to implement a full range of arithmetic functions in the hardware at a relatively low cost. This can be done in a single Integrated Circuit. Thus, a special purpose arithmetic processor, for performing only the arithmetic operations, can be constructed. This processor physically may be separate, yet can be utilized by the CPU to execute complex arithmetic instructions. Please note in the absence of arithmetic processors, these instructions may be executed using the slower software routines by the CPU itself. Thus, this auxiliary processor enhances the speed of execution of programs having a lot of complex arithmetic computations.

An arithmetic processor also helps in reducing program complexity, as it provides a richer instruction set for a machine. Some of the instructions that can be assigned to arithmetic processors can be related to the addition, subtraction, multiplication, and division of floating point numbers, exponentiation, logarithms and other trigonometric functions.

How can this arithmetic processor be connected to the CPU?

Two mechanisms are used for connecting the arithmetic processor to the CPU.

If an arithmetic processor is treated as one of the Input / Output or peripheral units then it is termed as a peripheral processor. The CPU sends data and instructions to the peripheral processor, which performs the required operations on the data and communicates the results back to the CPU. A peripheral processor has several registers to communicate with the CPU. These registers may be addressed by the CPU as Input /Output register addresses. The CPU and peripheral processors are normally quite independent and communicate with each other by exchange of information using data transfer instructions. The data transfer instructions must be specific instructions in the CPU. This type of connection is called loosely coupled.

On the other hand if the arithmetic processor has a register and instruction set which can be considered an extension of the CPU registers and instruction set, then it is called a tightly coupled processor. Here the CPU reserves a special subset of code for arithmetic processor. In such a system the instructions meant for arithmetic processor are fetched by CPU and decoded jointly by CPU and the arithmetic processor, and finally executed by arithmetic processor. Thus, these processors can be considered a logical extension of the CPU. Such attached arithmetic processors are termed as co-processors.

The concept of co-processor existed in the 8086 machine till Intel 486 machines where co-processor was separate. However, Pentium at present does not have a separate co-processor. Similarly, peripheral processors are not found as arithmetic processors in general. However, many chips are used for specialized I/O architecture. These can be found in further readings.

Check Your Progress 2

1. Draw the logic circuit for a ALU unit.

2. What is an Arithmetic Processor?

   ....

   ....

   ....

3.4 SUMMARY

In this unit, we have discussed in detail the hardware implementation of micro-operations. The unit starts with an implementation of bus, which is the backbone for any register transfer operation. This is followed by a discussion on arithmetic circuit and micro-operation thereon using full adder circuits. The logic micro-operation implementation has also been discussed. Thus, leading to a logical construction of a simple arithmetic – logic –shift unit. The unit revolves around the basic ALU with the help of the units that are constructed for the implementation of micro-operations.

In the later part of the unit, we discussed the arithmetic processors. Finally, we have presented a few chipsets that support the working of a processor for input/output functions from key board, printer etc.

3.5 SOLUTIONS/ ANSWERS

Check Your Progress 1

1. False
2. False
3. True

Check Your Progress 2

1. The diagram is the same as that of Figure 9.
2. Arithmetic processor performs arithmetic computation. These are support processors to a computer.