UNIT 2 REGISTER ORGANISATION AND MICRO-OPERATIONS

Structure

2.0 Introduction
2.1 Objectives
2.2 Basic structure of the CPU
2.3 An Advanced Structure
2.4 Register Organisation
  2.4.1 Programmer Visible Registers
  2.4.2 Status and Control Registers
2.5 Micro-Operations
  2.5.1 Register transfer Micro-operations
  2.5.2 Arithmetic Micro-operation
  2.5.3 Logic Micro-operations
  2.5.4 Shift Micro- operations
  2.5.5 Implementation of a Simple Arithmetic, Logic and Shift Unit
2.6 Instruction Execution and Micro-operations
2.7 Summary
2.8 Model Answers

2.0 INTRODUCTION

As discussed earlier that the main task performed by the CPU is the execution of instructions. In the previous unit, we have discussed about the instruction set of computer system. But, one thing which remained unanswered is: how these instructions will be executed by the CPU?

The above question can be broken down in two slightly simpler questions. These are:

What are the steps required for execution of an instruction?

How these steps are performed by the CPU?

The answer to the first question lies in the fact that each instruction execution consist of several steps. Together they constitute an instruction cycle. We have already given a state diagram about instruction cycle in Unit 1, Block 1 of this course. In this unit we will present a more structured view of the instruction cycle. We will also discuss about the micro-operations, the smallest operations performed by the CPU.

For answering second question, we must have understanding of the basic structure of a computer. As discussed earlier, the CPU consist of an Arithmetic Logic Unit, the control unit and operational registers. We will be discussing about the register organisation in this unit, whereas the arithmetic-logic unit and control unit organisation are discussed in the subsequent unit.

However, we have sequenced this unit starting from the very basic structure about the CPU followed by the discussion about the register organization in general. This is followed by the discussion on the micro-operations and their implementation. The discussion on micro-operations will gradually lead us towards the discussion of a very simple ALU structure. We will finally wind up the unit with the discussions on instruction cycle.

2.1 OBJECTIVES

At the end of this unit, you should be able to

- describe the register organisation of the CPU
- differentiate among various structure of the CPU
- define what is a micro-operation
- differentiate among various micro-operations

- discuss how the micro-operations can be implemented
- discuss about an instruction cycle.

2.2 BASIC STRUCTURE OF THE CPU

As discussed earlier, the CPU basically consists of two main components.

- An Arithmetic and Logic Unit to perform the arithmetic or logic operation on data,
- A Control Unit which plays important role for the functioning of the CPU itself and transfer of data/information from/to another device to/from CPU.

In addition CPU contains several operational registers. The basic task performed by the CPU is the instruction execution. An instruction is executed using several small operations called micro-operations.

The basic issues relating to CPU can be:

- it should be as fast as possible only the available technology should limit the CPU speed but the number of components should be small in a good CPU.
- the capacity of main memory needed by the CPU should be quite large. Since it is large, therefore, it should be constructed using a slower technology than that of a CPU.

Let use define the cycle time of the CPU (t_{cpu}) as the time taken by the CPU to execute a well-defined shortest micro-operation, and memory cycle time as the speed at which the memory can be accessed by the CPU. It has been found that memory cycle time (t_{mem}) is approximately 1 - 10 times higher than that of CPU cycle time. That is t_{mem}/t_{cpu} = 1 to 10. Therefore, within the CPU temporary storage is provided in the form of CPU registers. The CPU registers are used to store instructions and operands within the CPU. The CPU registers can be accessed almost instantaneously. In other words, the ratio of the time to access a register by CPU in comparison to the time to access memory by the CPU is approximately equal to t_{mem}/t_{cpu}. Thus, the instructions whose operands are stored in the fast CPU registers can be executed rapidly in comparison to the instructions whose operands are in the main memory of a computer. Thus, the instruction execution is implemented as:

- Bring in the operands required by the instruction from main memory to the CPU registers.
- Perform the operation desired by the instruction on the operands stored in the registers.
- Finally the results are transferred back to the memory if needed.

The input/output from the devices can also be carried out in the same way using I/O controllers.

The design of CPU in modern form was first proposed by John von Neumann and his colleagues for the IAS computer. The IAS computer had a minimal number of registers along with the essential circuits. This computer had a small set of instruction and an instruction was allowed to contain only one operand address. A register called Accumulator was used as a key register for execution of most of the instructions as it was used for storing the input or output operand for the arithmetic logic unit. Figure 1 gives the structure of the IAS computer.

This type of architecture is used in some modern mini and micro-computers. The structure shown in figure 1 consist of the following registers.

Accumulator (AC): It interacts with ALU and stores the input or output operand.

Data Register (DR): It acts as a buffer storage between the main memory and the CPU. It also stores the operand for the instructions such as ADD DR or AC $\leftarrow$ AC + DR, that is content of AC and Data Register are added by ALU and the results are stored in the accumulator. Thus, data register can also store one of the input operand.

Program Counter (PC): It contains the address of the next instruction word to be executed.

Instruction Register (IR): Holds the current instruction.

Memory Address Register (MAR): Used to provide address of memory location from where data is to be retrieved or to which data is to be stored.

The contents of PC is modified either after fetching an instruction or by a branch or skip instruction. MAR and DR plays important roles in transfer of data between CPU and the memory. In the computer systems which use system bus, MAR is directly connected to address bus, while DR is directly connected to data bus. DR is also used to interchange data among several other registers.

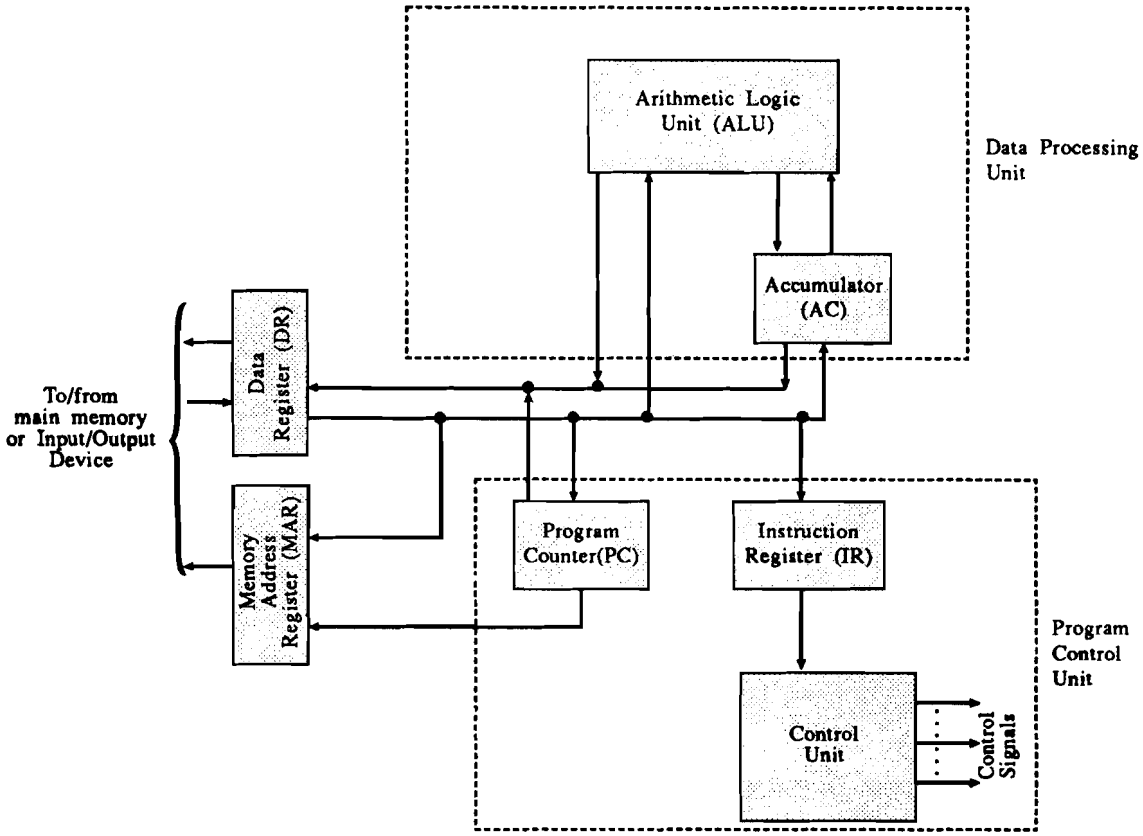


Figure 1: Basic Structure of the CPU

So, we have a simple basic CPU which can perform various computational tasks easily. How to make this simple structure more powerful in terms of processing efficiency and instructional style. In the next section we will discuss how the simple structure can be made more powerful.

2.3 AN ADVANCED STRUCTURE

Let us have few additional features in the simple structure of the CPU such that it can be made more powerful. Some of these aspects are:

- Provide additional registers for storing operands and addresses, thus, replacing a single accumulator by a set of registers. These registers can be used as general purpose registers which are multi-purpose in nature (e.g. can store either operands or addresses). A set of these general registers may be termed as a register file. One of the key function of these registers is to store the operands which are needed for calculation of memory address. The IBM S/360-370 computers had this general register organisation. In some microprocessor, some special address registers namely index register, base register, etc. are used. An example of such case is Motorola 68020 microprocessor which have a set of 8 data registers along with a matching set of 8 address registers.
- Increase the capabilities of ALU circuits. For example, most microprocessor have capabilities for performing addition and subtraction on fixed point numbers. This capability with only little extra circuitry can be used for multiplication and division on fixed point numbers also. However for implementing arithmetic on floating point numbers a substantial increase of circuitry is needed.
- Include special register to facilitate conditional jumps within a program. A status register which gives information about various conditions such as the sign of the result,

whether the result is zero, arithmetic overflow, etc. in the preceding instruction execution, can be used. This status register can be checked for typical condition for execution of a branch instruction.

- Include special registers for transfer of control between different subroutines or subprograms or interrupts. One such register used in IBM 360/370 series is called PSW (Program Status Word) which stores program counters and various condition flags. Thus, PSW can record the execution status of the program. On occurrence of interrupt or a call to subroutine this PSW is stored in a specific area in the memory and is restored when the execution of this program is to be resumed. Many computers use a stack in the main memory for controlling transfer of control. This is a very flexible approach. It uses a special area in the main memory and a register called stack pointer (SP) points to the top of the stack. The stack is a LIFO (Last In First Out) data structure. It has been found to be very useful for implementing program transfer of control, very efficiently.
- Provisions for execution of more than one instruction simultaneously. These provisions are not discussed at present but will be taken up for discussion of Block 4 of this course.

Considering the requirements, let us have a more enhanced view of the CPU. Figure 2 give the detailed view of the CPU.

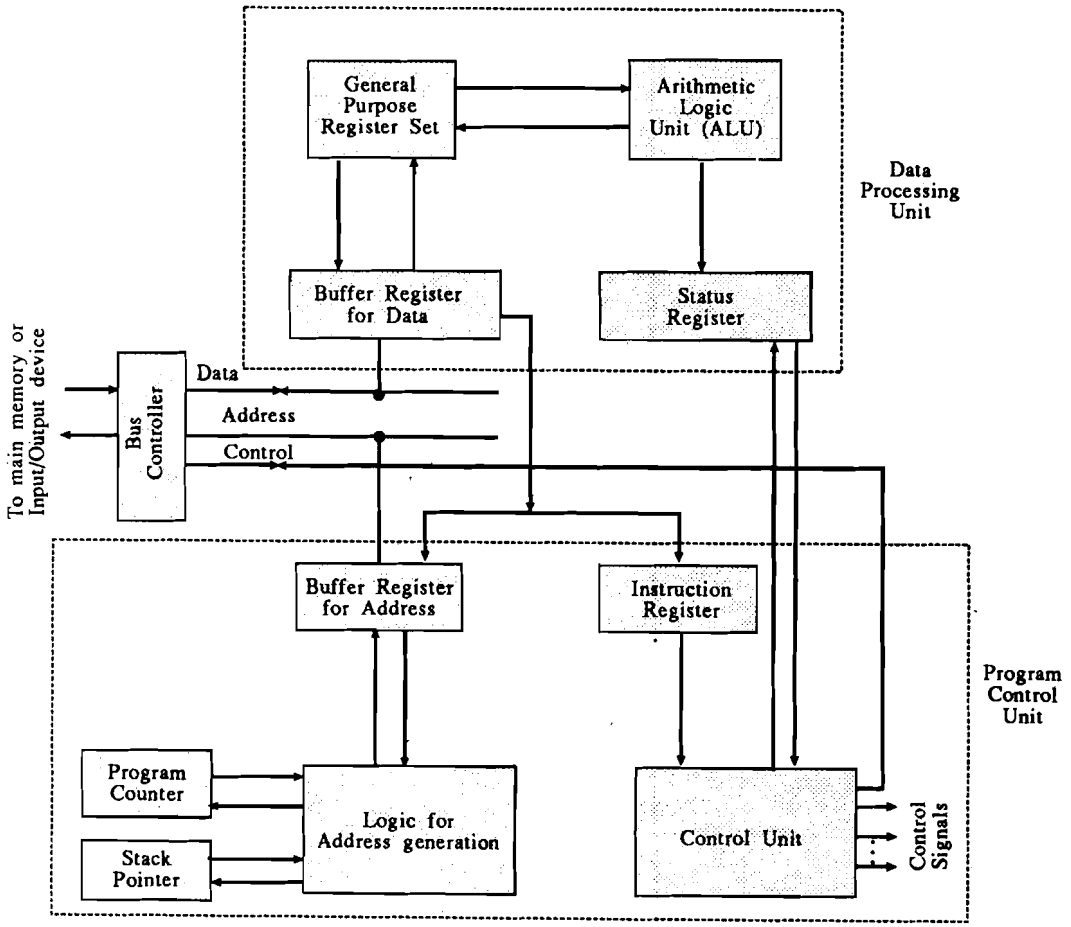


Figure 2 : CPU with general register organisation

The general purpose registers can be in general 8-16. ALU performs operation on the data stored in these general purpose registers and also stores results in these registers only. There are few special-purpose address registers. Two of these are Program Counter (PC) and Stack Pointer (SP). The status register stores the key characteristics or conditions of the result of the last ALU operation. A special simple arithmetic logic unit can be attached as an address generation logic performing the simple fixed point computations. The inputs to the control unit are from the instruction register which contains the operation code of the instruction to be executed and from status register which help in generating proper control signals on branch operation. The system bus plays the role of communication media. Please note the direction of arrows on address bus. There are several intermediate buffer registers also which help in intermediate storage of information. In this organisation of CPU parallelism can be implemented within the ALU operation or through the overlapped operations of data processing and program control unit.

After discussing so much about the CPU structure let us discuss about its components. In the subsequent sections we will discuss about the Register organisation in details followed by the discussions on microoperations. This will lay a foundation for discussions or Arithmetic Logic Unit and control unit organisation which are the topics in Unit 3 and 4 of this Block.

Check Your Progress 1

State true or false.

1. The program counter (PC) register is used for storing an instruction.

   True ☐ False ☐

2. Program Status Word (PSW) is used for storing the instruction which is to be executed next.

   True ☐ False ☐

3. Accumulator machines may require more memory references than that of a general purpose machine for executing an optimised program.

   True ☐ False ☐

4. Stack pointer points to the top of register file.

   True ☐ False ☐

2.4 REGISTER ORGANISATION

In the previous subsection, we have given some hints to the types of Registers a modern day CPU must have. Let us take a more general view on register organisation in this section. The internal processor memory of a CPU is served by its registers. One of the key difference among various computers is the difference in their register sets. Some computers have very large while some has smaller sets. But on the whole, from a user's point of view the register set can be classified under two basic categories.

- **Programmer Visible Registers:** These registers can be used by machine or assembly language programmers to minimize the references to main memory.
- **Status Control and Registers:** These registers can not be used by the programmers but are used to control the CPU or the execution of a program.

Different vendors has used some of these registers interchangeably, therefore you should not stick to these definitions rigidly. Yet this categorisation will help in better understanding of register sets of machine. Therefore, let us discuss more about these categories.

2.4.1 Programmer Visible Registers

These registers can be accessed using machine language. In general we encounter four types of programmer visible registers.

- General Purpose Registers
- Data Registers
- Address Registers
- Condition Codes Registers

The general purpose registers are used for various functions desired by the processor. A true general purpose register can contain operand or can be used for calculation of address of operand for any operation code of an instruction. But trends in today's machines show drift towards dedicated registers, for example, some registers may be dedicated to floating point operations. In some machines there is a distinct separation between data and address registers.

The data registers are used only for storing intermediate results or data. These data registers are not used for calculation of address of the operand.

An address register may be a general purpose register, but some dedicated address registers are also used in several machines. Examples of dedicated address registers can be:

Segment Pointer : Used to point out a segment of memory.

Index Register : These are used for index addressing scheme.

Stack Pointer (when programmer visible stack addressing is used.)

There are several issues related to programmer visible registers. Some of the issues are :

Do we use general purpose registers or dedicated register in a machine?

Does the number of registers effect the design of an instruction of a computer?

Well, in case of specialised register the number of bits needed for register specific details are reduced as here we need to specify only few registers out of a set of registers. However, this specialisation does not allow much flexibility to the programmer. Although there is no best solution to this problem, yet the trends are in favour of use of specialised registers.

Another issue related to the register set design is the number of general purpose registers or data and address registers to be provided in a micro-processor. The number of registers also effect the instruction design as the number of registers determines the number of bits needed in an instruction to specify a register reference. In general, it has been found that optimum number of registers in a CPU is in the range 8 to 32. In case registers fall below the range then more memory reference per instruction on an average will be needed, as some of the intermediate result then have to be stored in the memory. On the other hand, if the number of registers go above 32, then there is no appreciable reduction in memory references. However, in some computers hundreds of registers are used. These systems have special characterisiics. Reduced Instruction Set Computers (RISC) exhibit this property. RISC computers are discussed in Block 4 of this course. What is the importance of having less memory references? As the time required for memory reference is more than that of a register reference, therefore, the increased number of memory references results in slower execution of a program.

Register Length: Another important characteristic related to registers is the length of a register. Normally, the length of a register is dependen. on its use. For example, a register which is used to calculate address must be long enough to hold the maximum possible address. Similarly, the length of data register should be long enough to hold the data type it is supposed to hold. In certain cases two consecutive registers may be used to hold data whose length is double of the register length.

Condition code registers may only be partially available to the programmers. These register contains condition codes which are also known as flags. These flags are set by the CPU hardware while performing an operation. For example, an addition operation may set the overflow flag or on a division by 0 the overflow flag can be set etc. These codes may be tested by a program for a typical conditional branch operation. The condition codes are collected in one or more registers. RISC machines have several set of conditional code bits. In these machines an instruction specifies the set of condition codes which is to be used. Independent sets of condition code enable the provisions of having parallelism within the instruction execution unit.

One of the key operation which is needed with the programmer usable registers happens when a subroutine call is issued. On a subroutine call all the registers either are saved by the call statement itself and restored on encountering a return statement from the subroutine. This operation in most machines is automatic yet in certain machines this is done by the programmers. Similarly while writing interrupt service routine you need to save some or all programmer usable registers.

2.4.2 Status and Control Registers

For control of various operations several registers are used. These registers can not be used in data manipulations, however, the content of some of these registers can be used by the programmer. Some of the control register for a von Neumann machine can be the Program Counter (PC), Memory Address Register (MAR) and Data Register (DR).

Almost all the CPUs, as discussed earlier, have status register, a part of which may be programmer visible. A register which may be formed by condition codes is called condition code register. Some of the commonly used flags or condition codes in such a register may be:

Sign flag : This indicates whether the sign of previous arithmetic operation was positive (0) or negative (1).

Zero flag : This flag bit will be set if the result of last arithmetic operation was zero.

Carry flag : This flag is set, if a carry results from the addition of the highest order bits or a borrow is taken on subtraction of highest order bit.

Equal flag : This bit flag will be set if a logic comparison operation finds out that both of its operands are equal.

Overflow flag : This flag is used to indicate the condition of arithmetic overflow.

Interrupt Enable/disable flag : This flag is used for enabling or disabling interrupts.

Supervisor flag : This flag is used in certain computers to determine whether the CPU is executing in supervisor or user mode. In case the CPU is in supervisor mode it will be allowed to execute certain privileged instructions.

In most CPUs, on encountering a subroutine call or interrupt handling routine, it is desired that the status information such as conditional codes and other register information be stored and restored on initiation and end of these routines respectively. The register often known as Program Status Word (PSW) contains condition code plus other status information. There can be several other status and control registers such as interrupt vector register in the machines using vectored interrupt, stack pointer if a stack is used to implement subroutine calls, etc.

The status and control register design is also dependent on the Operating System (OS) support. The functional understanding of OS helps in tailoring the register organisation. In fact, some control information is only of specific use to the operating system.

One major decision to be taken for designing status and control registers organisation is : how to allocate control information between registers and the memory. Generally first few hundred or thousands words of memory are allocated for storing control information. It is the responsibility of the designer to determine how much control information should be in registers and how much in memory. This in fact is a trade off between the cost and the speed.

Check Your Progress 2

1. What is an address register?

   .....

   .....

2. A machine has 20 general purpose registers. How many bits will be needed for register address of this machine?

   .....

   .....

3. What is the advantage of having independent set of conditional codes?

   .....

   .....

4. Can we store status and control information in the memory?

   .....

2.5 MICRO-OPERATIONS

After discussing about the structure and register organisation. Our next task is to look at the functionality of ALU and control unit. However as the main task of computer is instruction execution, therefore, the details on how instruction can be executed will help in better

understanding of functionality of ALU and control unit. In this section we will first define the various micro-operations and their hardware implementation from where we will move on to design of a very simple circuit of an arithmetic logic unit as it will logically follow the discussion. This discussion will be followed by a new look at instruction execution in the next section.

A micro-operation, in general, is a primitive action performed by a machine on the data stored in the registers. In digital computers, in general, there are four types of micro-operations.

- Register transfer micro-operations
- Arithmetic micro-operations
- Logic micro-operations
- Shift micro-operations

2.5.1 Register Transfer Micro-operations

These micro-operations as the name suggest transfer information from one register to another. The information does not change during this micro-operation. A register transfer micro-operation may be designed as:

$$R1 \leftarrow R2$$

which implies that transfer the content of register R2 to register R1. R2 here is a source register while R1 is a destination register. We will use this notation through out this section. For a register transfer micro-operation there must be a path for data transfer from the output of the source register to the input of destination register. In addition, the destination register should have a parallel load capability, as we expect the register transfer to occur in a pre-determined control condition. We will discuss more about the control in the later units of this Block.

A common path for connecting various registers is through a common internal data bus of the processor. In general the size of this data bus should be equal to the number of bits in a general register.

Let us briefly find out how this internal processor bus can be constructed. We will give you a very simple way of constructing the bus for four bit registers using 4×1 multiplexers (please refer to figure 3).

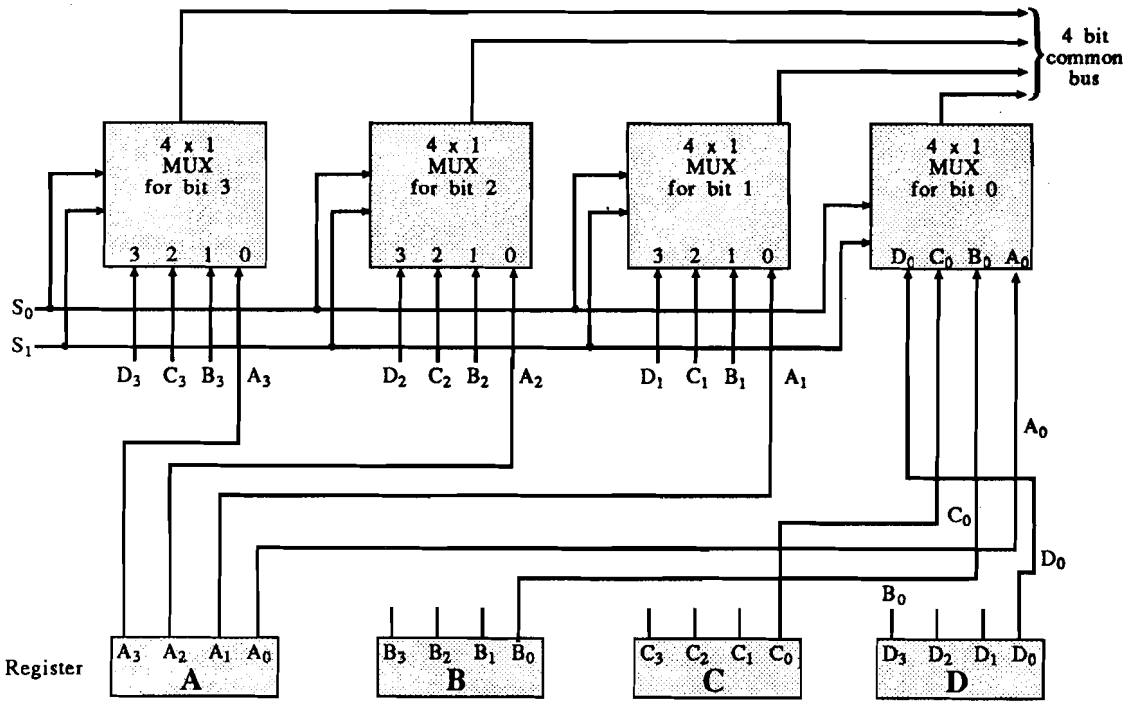


Figure 3 : A four bit Bus system for four bit four registers

Please note that the logic circuit of figure 3 uses four multiplexers (MUX), two selection lines and four registers. Also note that, the output of a MUX is one of the four inputs, that is, for i^{th} bit MUX the output can be either A_i or B_i or C_i or D_i, that is the i^{th} bit of any one of the four registers. The select line determines which of these register bit is to be selected. This is shown as follows:

S_1	S_0	Decimal equivalent of S_1 S_0 (Decimal Select (DS))	Output bit selected
0	0	0	A_i
0	1	1	B_i
1	0	2	C_i
1	1	3	D_i

Since, the same control bits S_1S_0 are passed to all the multiplexers, therefore, all the MUX will output bits of the same register. Thus, on having Decimal Select (DS) as 0 bus will output register A, on DS as 1 bus will output B and so on.

For receiving the data from this bus all the registers, input should be connected to this bus and all these registers should have a parallel load capability. Thus, by enabling load signal, a register can take data from the bus.

Now, let us focus on another important transfer which does not take place through the internal data bus, but through the system bus. These transfers are related to memory and Input/Output modules. The data transfer from Input/Output module to CPU or vice-versa is not very different from the transfer from memory to CPU or vice-versa. For a special case of memory mapped I/O both are handled in the same way. Also, the input/output operation is treated as a separate activity, where normally a program and, therefore, instructions are executed. Thus, let us focus our discussion on memory transfer which is the most important transfer for instruction execution as it has to take place at least once for every instruction.

Memory Transfer: Memory transfer is achieved via a system bus. Since, the main memory is a random access memory, therefore, address of the location which is to be used is to be supplied. This address is supplied by the CPU on the address bus. There are two memory transfer operations : Read and Write. Let us consider the CPU structure as shown in Figure 1, then the two memory operations will be performed as:

Memory Read :

- Put the memory address in the Memory address register (MAR)
- Read the data of the location. This operation is achieved by putting the MAR data on address bus along with a memory read control signal on the control bus. The resultant of memory read is put into the data bus which in turn stores the read data in the data register (DR). This whole operation can be shown as :

$$DR \leftarrow M(MAR)$$

Memory Write :

- Put the desired memory address in MAR and the data to be written in the DR.
- Write the data into the location : MAR puts the address on address bus and DR puts the data on data bus. A write control signal along with these two signals enables the data on data bus to be written into the memory location addressed by MAR.

$$M(MAR) \leftarrow DR$$

Normally, a memory read or write operation requires more clock cycles than a typical register transfer operation. The logic circuit of memory is shown in unit 3 of block 1 of this course.

2.5.2 Arithmetic Micro-operations

These micro-operations perform some basic arithmetic operations on the numeric data stored in the registers. These basic operations may be: addition, subtraction, incrementing a number, decrementing a number and arithmetic shift operation. An "add" micro-operation can be specified as :

$$R3 \leftarrow R1 + R2$$

It implies : add the contents of registers R1 and R2 and store them in register R3.

The add operation mentioned above requires three registers along with the addition circuit at the ALU. An alternate add micro-operation for the structure shown in Figure 1 will be :

$$AC \leftarrow AC + DR.$$

Subtraction in many machines, is implemented through complement and addition operation as:

$$\begin{aligned} & R3 \leftarrow R1 - R2 \\ \Rightarrow\ & R3 \leftarrow R1 + (2\text{'s complement of } R2) \\ \Rightarrow\ & R3 \leftarrow R1 + (1\text{'s complement of } R2 + 1) \\ \Rightarrow\ & R3 \leftarrow R1 + R2 + 1 \end{aligned}$$

An increment operation can be symbolised as:

$$R1 \leftarrow R1 + 1$$

While a decrement operation can be symbolised as:

$$R1 \leftarrow R1 - 1$$

These increment and decrement operations can be implemented by using a combinational circuit or binary up/down counter. What about the multiply and division operations? Are not they micro-operations? In most of the computer multiply and division are implemented using add/subtract and shift micro-operations. If a digital system has implemented division and multiplication by means of combinational circuits then we can call these as the micro-operations for that system.

Implementation of Arithmetic Circuits for Arithmetic Micro-operation

An arithmetic circuit is normally implemented using 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.

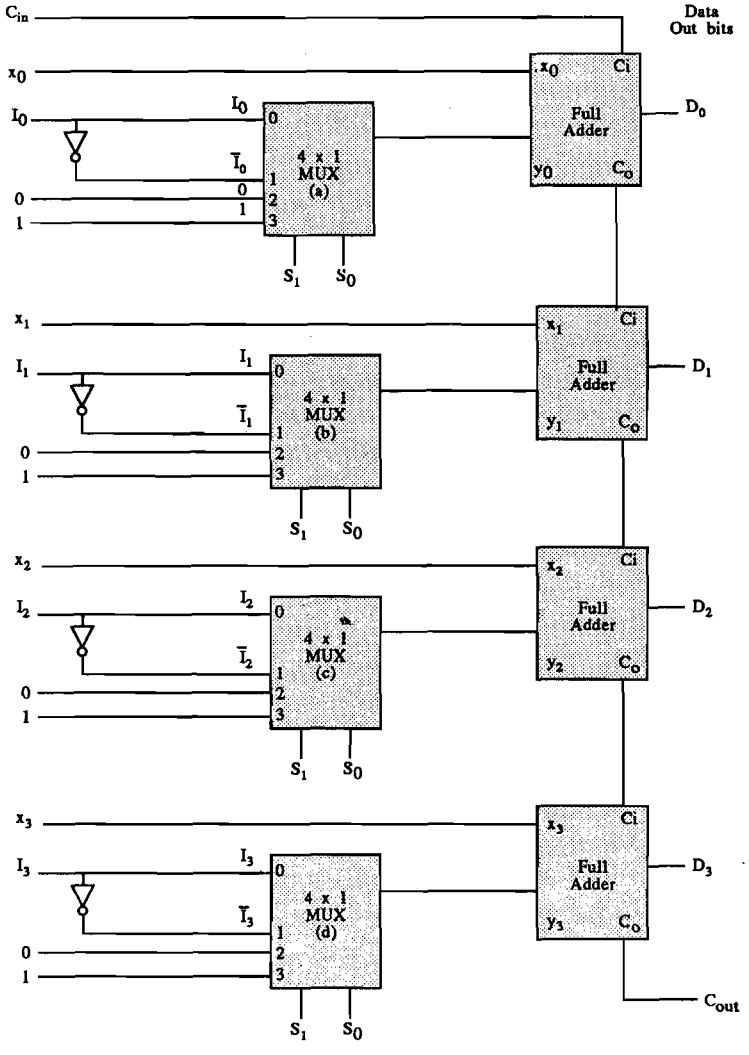


Figure 4 : A Four-bit arithmetic circuit

Please note that each of the multiplexer (MUX) of the given circuit have two select inputs, S_0 and S_1 . We have drawn separate selection lines for each MUX for simplifying the circuit.

The two selection lines to the MUX along with the overall carry in (Cin) bit determines the type of micro-operation to be performed by the circuit. How ?

This four-bit circuit take input of two 4-bit data values and a carry in bit and outputs the four resultant data bits and a carry out bit. Please note in the circuit that one input, that is x is fed directly to the full adder, while the second input I is fed through a multiplexer. The S_0 and S_1 inputs to the multiplexer determines what type of y input is to be selected for the full adder. Let us see how S_0 and S_1 determine the y input through the multiplexer with the help of following truth table.

S_1	S_0	Output of 4×1 MUX = MUX (a)	MUX (b)	MUX (c)	MUX (d)	y input	Comments
0	0	I_0	I_1	I_2	I_3	I	The data word I is input to Full Adder.
0	1	$\bar{I}_0$	$\bar{I}_1$	$\bar{I}_2$	$\bar{I}_3$	$\bar{I}$	1's complement of I is input to Full Adder.
1	0	0	0	0	0	0	Data word 0 is input to Full Adder.
1	1	1	1	1	1	F_H	Data Word 1111 is input to Full Adder.

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

Now let us discuss how by coupling C_{in} input bit with the selection input values we can obtain various micro-operations.

S_1	S_0	C_{in}	x	y	Equivalent Function	Equivalent Micro-operation	Micro-operation name
0	0	0	x	I	$F = x + I$	$R \leftarrow R_1 + R_2$	Add
0	0	1	x	I	$F = x + I + I$	$R \leftarrow R_1 + R_2 + 1$	Add with carry
0	1	0	x	$\bar{I}$	$F = x + \bar{I}$	$R \leftarrow R_1 + \bar{R}_2$	Subtract with borrow
0	1	1	x	$\bar{I}$	$F = x + (\bar{I} + 1)$	$R \leftarrow R_1 +$ 2's Complement of R_2	Subtract
1	0	0	x	0	$F = x$	$R \leftarrow R_1$	Transfer
1	0	1	x	0	$F = x + 1$	$R \leftarrow R_1 + 1$	Increment
1	1	0	x	F_H	$F = x + F_H$	$R \leftarrow R_1 +$ (All 1s)	Decrement
1	1	1	x	F_H	$F = x$	$R \leftarrow R_1$	Transfer

Figure 6: Micro-operations which can be performed using the Arithmetic Circuit of Figure 4

Some of the micro-operations given above need no explanation but let us discuss few of them.

In subtract with borrow we have

$$F = x + \bar{I}$$

or $$F = (x - 1) + (\bar{I} + 1)$$

$(\bar{I} + 1)$ is 2's complement notation for $-I$

$$F = (x - 1) - I$$

This implies that we are taking a borrow out of x before subtraction of I.

In Decrement x we have function as

$$F = x + F_H$$

or $$F = x + (F_H + 1) - 1$$

For this four bit words

$$F_H = 1111$$
$$+ \quad 0001$$
$$(1)\ 0000$$

carry out (ignored)

Therefore, $F = x + 0000 - 1$

$\Rightarrow F = x - 1$

2.5.3 Logic Micro-operations

Logic operations are basically the binary operations which are performed on the string of bits stored in the registers. For a logic micro-operation each bit of a register is treated as a variable. A logic microoperation:

$R1 \leftarrow R1.R2$ specifies AND operation to be performed on the contents of R1 and R2 and store the results in R1. For example, if R1 and R2 are 8 bits registers and

R1 contains 10010011 and
R2 contains 01010101
Then R1 will contain 00010001 after AND operation.

Some of the common logic micro-operations are AND, OR, NOT or Complement, Exclusive OR, NOR, NAND.

Implementation of Logic Micro-operations

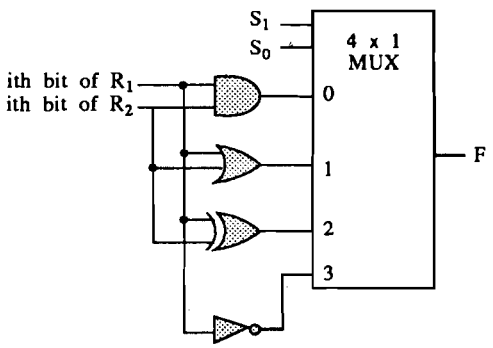
For implementation, let us first ask the questions how many logic operations can be performed with two binary variables. We can have four possible combination of input of two variable. These are 00, 01, 10, and 11. Now, for all these 4 input combination we can have $2^4 = 16$ output combinations of a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

Output for Combination (xy)					Equivalent Boolean	Equivalent Micro-operation	Micro-operation Name
00	01	10	11	Function	Function		
0	0	0	0	F_0	$F_0 = 0$	$R \leftarrow 0$	Clear
0	0	0	1	F_1	$F_1 = x.y$	$R \leftarrow R_1 \wedge R_2$	AND
0	0	1	0	F_2	$F_2 = x.\overline{y}$	$R \leftarrow R_1 \wedge \overline{R_2}$	R_1 AND with complement R_2
0	0	1	1	F_3	$F_3 = x$	$R \leftarrow R_1$	Transfer of R_1
0	1	0	0	F_4	$F_4 = \overline{x}.y$	$R \leftarrow \overline{R_1} \wedge R_2$	R_2 AND with complement R_1
0	1	0	1	F_5	$F_5 = y$	$R \leftarrow R_2$	Transfer of R_2
0	1	1	0	F_6	$F_6 = x \oplus y$	$R \leftarrow R_1 \oplus R_2$	Exclusive OR
0	1	1	1	F_7	$F_7 = x+y$	$R \leftarrow R_1 \vee R_2$	OR
1	0	0	0	F_8	$F_8 = \overline{(x+y)}$	$R \leftarrow \overline{(R_1 \vee R_2)}$	NOR
1	0	0	1	F_9	$F_9 = \overline{(x \oplus y)}$	$R \leftarrow \overline{(R_1 \oplus R_2)}$	Exclusive NOR
1	0	1	0	F_{10}	$F_{10} = \overline{y}$	$R \leftarrow \overline{R_2}$	Complement of R_2
1	0	1	1	F_{11}	$F_{11} = x + \overline{y}$	$R \leftarrow R_1 \vee \overline{R_2}$	R_1 OR with complement R_2
1	1	0	0	F_{12}	$F_{12} = \overline{x}$	$R \leftarrow \overline{R_1}$	Complement of R_1
1	1	0	1	F_{13}	$F_{13} = \overline{x} + y$	$R \leftarrow \overline{R_1} \vee R_2$	R_2 OR with complement R_1
1	1	1	0	F_{14}	$F_{14} = \overline{(x.y)}$	$R \leftarrow \overline{(R_1 \wedge R_2)}$	NAND
1	1	1	1	F_{15}	$F_{15} = 1$	$R \leftarrow$ 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 R_1 and R_2 on each corresponding bit of the registers R_1 and R_2. 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. Other 12 micro- operations can be derived from these four micro-operations. Figure 8 shows one bit, that is the i^{th} bit stage of the four logic operations. Please note that the circuit consist of 4 gates and a 4×1 MUX. The i^{th} bits of Register R_1 and R_2 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

Let us now discuss how these four micro-operations can be used in implementing some of the important applications of manipulation of bits of a word, such as, changing some bit values or deleting a group of bits . We are assuming that the result of logic micro-operations go back to Register R_1 and R_2 contains the second operand.

We will play a trick with the manipulations we are performing. Let us select 1010 as 4 bit data for register R_1 and 1100 data for register R_2. Why? Because if you see the bit combinations of R_1 and R_2, they represent the truth table entries 00, 01, 10 and 11. Thus the resultant of the logical operation on them will tell us which logic micro-operation is needed to be performed for that data manipulation. The following table gives details on some of these operations.

Operation name	**What is the operation?**
Selective Set 1010 (R_1) 1100 (R_2) 1110	Sets those bits in Register R_1 for which the corresponding R_2 bit is 1. The value 1110 suggests that selective set can be done using logic OR micro-operation. Please note that all those bits of R_1, for which we have 0 bit in R_2 have remained unchanged. The bits in R_1 which need to be set selectively must have the corresponding R_2 bits as 1.
Selective Clear 1010 (R_1) 1100 (R_2) 0010	Clear those bits in register R_1 for which corresponding R_2 bits are 1. The R1 value after the operation is 0010 which suggests that corresponding micro-operation is $R_1 \leftarrow R_1$ AND $\overline{R}_2$. The bits in R_1 which need to be cleared selectively must have corresponding R_2 bits as 1.
Selective Complement 1010 (R_1) 1100 (R_2) 0110	Complement those bits in register R_1 for which the corresponding R_2 bits are 1. The R_1 value 0110 after the operation suggests that the selective complement can be done using exclusive – OR micro- operation. The bits in R_1 which need to be complemented selectively must have the corresponding R_2 bits as 1.
Mask Operations 1010(R_1) 1100(R_2) 1000	Clears those bits in Register R_1 for which the corresponding R_2 bits are 0. The R_1 value after the operation is 1000 which suggests that the mask operation can be performed using AND micro-operation. However, the bits in R_1 which are cleared

or masked correspond to the bits on R_2 having a 0 value. The mask operation is preferred over selective clear as most of the computers provide AND micro-operation while the micro-operation required for implementing selective clear is normally not provided in computers.

Insert :

For inserting a new value in a bit. It is a two step process :

step 1 : Mask out the existing bit value

step 2 : Insert the bit using OR micro-peration with the bit which is to be inserted.

Example :

Say contents of R_1 = 0011 1011

Suppose, we want to insert 0110 in place of left most 0011 then :

	0011 1011	(R_1 before)
	0000 1111	(R_2 for masking)
	————	Perform AND operation (mask)
	0000 1011	(R_1 after)
Now insert :	0110 0000	(R_2 for insertion)
	————	Perform OR operation
	0110 1011	R_1 after insert

Clear

Clear all the bits. Implemented by taking exclusive OR with the same number.

1101 (R_1 before)
1101 (R_2)
0000 (R_1 after clear by using exclusive OR)

The exclusive OR, thus, can also be used for checking whether two numbers are equal or not.

2.5.4 Shift Micro-operations

Shift is a useful operation which can be used for serial transfer of data. Shift operations can also be used along with other (arithmetic, logic, etc.) operations. For example, for implementing a multiply operation arithmetic micro-operation (addition) can be used along with shift operation. The shift operation may result in shifting the contents of a register to the left or right. In a shift left operation a bit of data is input at the right most flip-flop while in shift right a bit of data is input at the left most flip-flop. In both the cases a bit of data enters the shift register. Depending on what bit enters the register and where does the shift out bit goes, the shifts are classified in three types. These are:

- logical
- arithmetic and
- circular

In logical shift the data entering by serial input to left most or right most flip-flop (depending on right or left shift operations respectively) is a 0.

If we connect the serial output of a shift register to its serial input then we encounter a circular shift. In circular shift left or circular shift right information is not lost, but is circulated.

In arithmetic shift a signed binary number is shifted to the left or to the right. Thus, an arithmetic shift-left causes a number to be multiplied by 2, on the other hand a shift-right causes a division by 2. But as in division or multiplication by 2 the sign of a number should not be changed, therefore, arithmetic shift must leave the sign bit unchanged. We have already discussed about shift operations in the unit 1.

Implementation of Shift Micro-operation

As far as implementation of shift micro-operation is concerned it can be implemented by a left-right shift registers having parallel load capabilities. Shift registers have already been discussed in the Unit 2 of Block 1 of this course.

Check Your Progress 3

1. Draw a logic diagram for a 2 bit bus for 2 bit 2 registers.

2. How does the memory read and write operation carried out using system bus?
3. Are multiplication and division arithmetic micro- operations?
4. What will be the value for R_2 operand if:
   (i) Mask operation, clears register R_1
   (ii) Bits 1011 0001 is to be inserted in an 8 bit R_1 register.
5. What are the differences between circular and logical shift micro-operations?

2.5.5 Implementation of a simple arithmetic, logic and shift unit

So, by now we have discussed how all the micro- operations can be implemented individually. If we combine all these circuits somehow 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 a ALU.

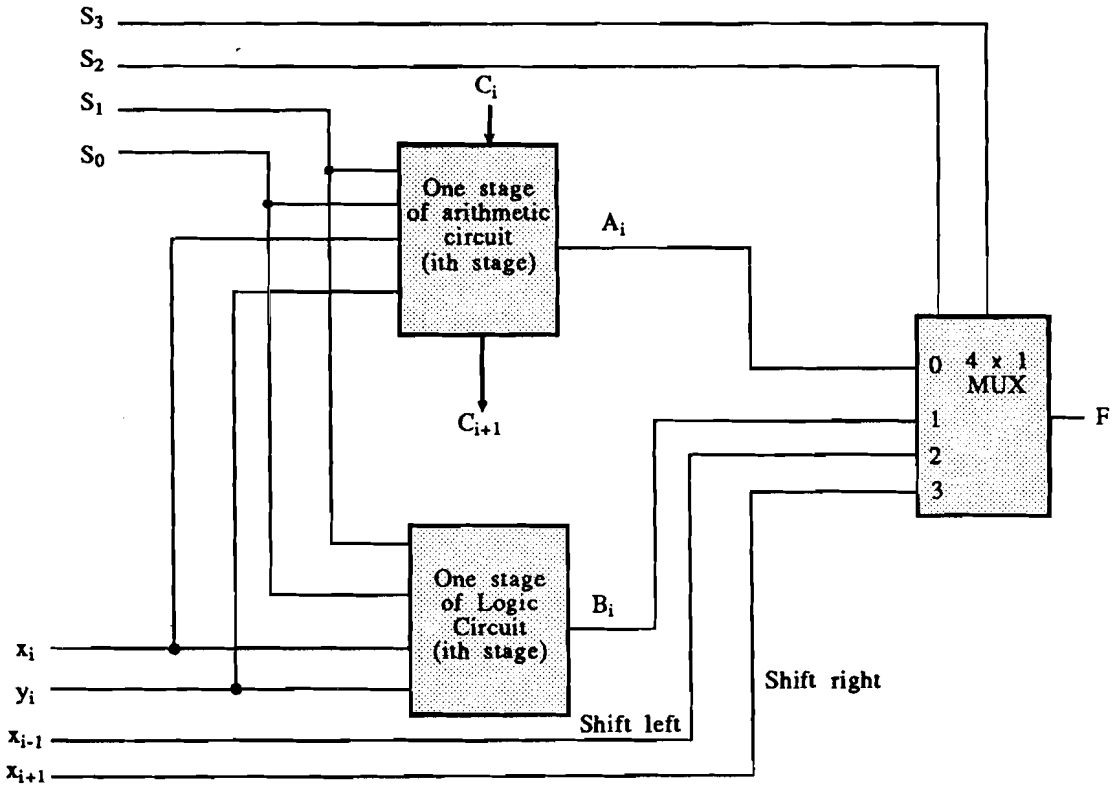


Figure 9 : One stage of ALU with shift capability

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 i^{th} bits of the registers to be manipulated. However, the i-i^{th} or i+i^{th} bit is also fed for the case of shift micro-operation of only one register. There are four selection lines which determines what micro-operation (arithmetic, logic or shift) on the input. The F_i is the resultant bit after a 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 the 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_3S_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 donot play any role. Now let us wind up our discussions about the micro-operations with the discussion on instruction execution using micro-instructions.

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(x)	$R \leftarrow Shr(R_1)$	Shift right	

Figure 10 : Micro-operations performed by ALU shown in Figure 9

2.6 INSTRUCTION EXECUTION AND MICRO-OPERATIONS

Let us once again look at the instruction cycle after we have discussed about instructions, register sets and micro-operations. A simple instruction cycle will consist of the following steps :

- fetching the instruction from the memory. This is also termed as fetch-cycle
- decode the instruction
- find out the effective address of the operand
- execute the instructions. Normally decoding is also performed in execute cycle.
- perform an interrupt cycle if an interrupt request is pending.

Let us explain how these steps of instruction can be broken down to micro-operations. For simplifying the discussions, let us assume that the machine has the structure as shown in figure 1. In addition, the instruction set of the machine has only two addressing modes : direct and indirect memory addresses.

An instruction cycle in such a machine may consist of four sub- cycles. These four cycles are :

Fetch Cycle : During this cycle the instruction which is to be executed next is brought from the memory to the CPU. The steps performed here are :

- The next instruction address is transferred from PC to MAR.
  $MAR \leftarrow PC$ (Register transfer Micro-operation)
- MAR puts this signal on the address bus for main memory location selection, whereas the control unit uses a memory read signal. The resultant so obtained is placed on the data bus where it is accepted by the Data register DR.
  $DR \leftarrow (M)$ ($(M) \rightarrow$ represents a memory read)
- The PC is incremented by one memory word length (Memory-read using bus. It may take more then one clock pulses depending on the t_{cpu} and t_{mem}).
  $PC \leftarrow PC + 1$
  This operation can be carried out in parallel to the above micro-operation.
- The instruction so obtained is transferred to the Instruction Register.
  $IR \leftarrow DR$

Indirect Cycle : Once the instruction is fetched, the next step is to determine whether it requires a memory reference or a register reference or it is an input/output instruction. An input/output instruction can be used for : a transfer of information from or to AC; or

checking status of I/O module; or enabling or disabling interrupts; or any other I/O related activity. We will not give details on these micro-operations in this unit. You can refer to further readings for details on these micro-operations. These instructions can go directly to execute cycle.

The register reference instructions such as complement AC, clear AC etc. normally donot require any memory reference (assuming register indirect addressing is not being used) and can directly go to the execute cycle.

However, the memory reference instruction can use several addressing modes. Depending on the type of addressing the effective address (EA) of operands in the memory are calculated. In certain cases, the calculation of effective address require one memory reference (for example in the case of indirect addressing) in such cases a special cycle which converts all the indirect addresses to direct address is required. This cycle is termed as an indirect cycle.

The steps involved in indirect cycle are :

- transfer the address bits of instruction to the MAR. This transfer can be achieved using DR only as DR and IR at this point of time contains same value.
  MAR← DR (Address)
- Once again a memory read operation as done in fetch cycle is performed and the desired address of the operand is obtained in the DR.
  DR← (M)
- transfer the address part so obtained in DR as the address part of instruction.
  IR (Address) ← DR (Address)

Thus, the purpose of indirect cycle is to remove the indirection by converting the indirect to the direct address.

Execute Cycle : After the fetch and indirect cycle (if required) an instruction is ready to be executed. In the execute cycle the instruction get actually executed. An, execute cycle depends on the opcode. A different opcode will require different sequence of steps for the execute cycle. Therefore, let us discuss few examples of execute cycle of some simple instruction for the purpose of identifying some of the steps needed during this cycle. Let us start the discussions with a simple case of addition instruction. Suppose, we have an instruction : Add A which adds the content of memory location A to AC storing the result in AC. This instruction will be executed in the following steps :

- Transfer the address portion (this address in symbolic form is A) of the instruction to the MAR.
  (Register transfer) MAR← PC
- Read the memory location A and bring the operand in the DR.
  (Memory-read) DR← (M)
- Add the DR with AC using ALU and bring the results back to AC.
  (Add micro-operation) AC← AC + DR

Now, let us try a complex instruction : a conditional jump instruction. Suppose, an the instruction ISZ A increments A and skips the next instruction if the contents of A has become zero. This is a complex instruction and requires intermediate decision making. The steps involved in this scheme are :

- Transfer the address portion of IR to the MAR.
  MAR ← IR (Address) (Register transfer)
- Read memory as we have done earlier. DR contains the operand A.
  DR ← (M) (Memory read)
- Transfer the contents of DR to AC as all the operating in the machine can be performed on AC.
  AC← DR (Register transfer)
- Increment the AC.
  AC ← AC + 1 (Increment micro-operation)
- Transfer the content of AC to DR.
  DR ← AC (Register transfer)
- Store the contents of DR into the location A using MAR. This operation proceeds

through as : Address bits are applied on address bus by MAR. The data is put into the data bus. The control unit providing control signal for memory write. Thus, resulting in a memory write at a location specified by MAR.
(M) ← DR (Memory write)

- If the contents of AC is zero then increment PC by one, thus skipping the next instruction.
  If AC = 0 then PC← PC + 1 (Increment on a condition)
  This operation can be performed in parallel to the memory write. Please note in the last step a comparison and an action is taken as a single step. This is possible as it is a simple comparison based on status flags.

Let us now take an example of branching operation. Suppose, we are using the first location of subroutine to store the return address then the steps involved in this sub-routine call (CALL A) can be

- Transfer the contents of address portion of IR to MAR.
  MAR ← IR(address) (Register Transfer)
- Transfer the return address, that is the contents of PC to DR.
  DR ← PC (Register transfer)
  This micro-operation can be performed in parallel to the previous micro-operation.
- Transfer the branch address that is A to program counter.
  PC ← IR(address) (Register transfer)
- Store the DR using MAR. Thus, the return address is stored at location A.
  (M)←DR (Memory write)
  This micro-operation can be performed in parallel to previous micro-operation.
- Increment the PC as it contains address A, whereas the first instruction of subroutine starts from the next location.
  PC←PC + 1 (Increment)

Thus, the execute cycle is not a predictable cycle.

Interrupt Cycle : After completion of the execute cycle, the machine checks whether an interrupt that was enabled has occurred or not. If an enabled interrupt has occurred then interrupt cycle is performed. The nature of interrupt cycle varies from machine to machine. However, let us discuss one of the simplest illustration of interrupt cycle events. A simple sequence of steps followed in interrupt cycle are :

- transfer the contents of PC to DR as this is the return address from the interrupt and it is to be saved.
  DR ←PC (Register transfer)
- Place the address of location, where the return address is to be saved, into MAR. Please note that this address is normally predetermined in computers.
  MAR← Address of location for saving return address.
- Store the contents of PC in the memory using DR and MAR.
  (M) ← DR (Memory write)
- Transfer the address of the first instruction ofinterrupt servicing routine to the PC.
  PC← service programs first instruction address interrupt
  This micro-operation can be performed in parallel to the second micro-operation.

After completing interrupt cycle the CPU will fetch the next instruction. During this time CPU might be doing the interrupt processing or executing the user program. Please note each instruction of interrupt service routine is executed as an instruction in an instruction cycle. Please note that instruction cycle discussed here involve many of the register transfer micro-operations. However, for an advanced structure, this requirement will be reduced because of general nature of the registers.

Please note for a complex machine the instruction cycle will not be as easy as this. You can refer to further readings for more complex instruction cycles.

Check Your Progress 4

State true or false

1. An instruction cycle does not include indirect cycle if the operands are stored in the register.

True ☐ False ☐

2. For variable length instructions, each instruction fetched is of length = maximum instruction length/word size.

True ☐ False ☐

3. Register transfer micro-operations are not needed for instruction execution.

True ☐ False ☐

4. Interrupt cycle results only in jumping to an interrupt service routine. The actual processing of the instructions of this routine is performed in instruction cycle.

True ☐ False ☐

2.7 SUMMARY

In this unit, we have discussed in details about the register organisation and a simple structure of the CPU. After this we have discussed in details about the micro-operations and their implementation in hardware using simple logical circuits. While discussing about micro-operations our main emphasis was on simple arithmetic, logic and shift micro-operations, in addition to register transfer and memory transfer. However, the knowledge we have acquired about register sets and conditional codes, helped us in giving us an idea, that conditional micro-operations can be implemented by simply checking flags and conditional codes. This idea will be more clear after we go through the unit 3 and unit 4. We have completed the discussions on this unit, with providing a simple approach of instruction execution with micro-operations. We will be using this approach for discussing control unit details in unit 3 and unit 4. You can refer to further readings for register organisation examples and for more details on the micro-operations and instruction execution.

2.8 MODEL ANSWERS

Check Your Progress 1

1. False, 2. False, 3. True, 4. False

Check Your Progress 2

1. A register which is used only for the calculation of operand addresses are called address registers.
2. 5 bits
3. It helps in implementing parallelism in the instruction execution unit.
4. Yes. Normally, the first few hundreds of words of memory are allocated for storing control information.

Check Your Progress 3

1. No model answer
2. Refer to section 2.5.1
3. Refer to section 2.5.2
4. (i) 0000 0000

   (ii) Initially 0000 0000 followed by 1011 0001 for second step of insert operation.
5. Refer to section 2.5.4

Check Your Progress 4

1. True 2. True 3. False 4. True