UNIT 10 REGISTERS, MICRO-OPERATIONS AND INSTRUCTION EXECUTION

Structure **Page No.**

10.0 INTRODUCTION

In the previous unit, you have gone through the concepts relating to various types of instructions and operands that a computer can have. The main task performed by the CPU is the execution of the instructions. This Unit focusses on the process of execution of these instructions by the CPU. This Units tries to answers the following two questions regarding instruction execution.

What are the steps required for the execution of an instruction? And

How are these steps performed by the CPU?

Execution of an instruction can be divided into sequence of steps, together they constitute an instruction execution sequence, called instruction cycle. Each of these steps can be termed as a micro-operation. A micro-operation is the smallest operation performed by the CPU. These operations put together execute an instruction.

For answering the second question, youmayrecall the basic structure of a computer. The CPU of a computer consists of an Arithmetic Logic Unit (ALU), the Control Unit (CU) and operational registers. We will be discussing the register organisation and ALU in this unit, whereas the control unit organisationis discussed in next unit.

In this unit we will first discuss the basic CPU structure and the register organisation in general. This is followed by a discussion on micro-operations that include register–transfer, arithmetic, logic and shift micro-operation and their implementation, which forms the basis of design of a ALU. The discussion on micro-operations will gradually lead us towards the discussion of an ALU structure. The unit will also discuss about the arithmetic processor, which are commonlyused for floating point computations.

10.1 OBJECTIVES

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

- describe the register organisation of the CPU;
- define what is a micro-operation;
- discuss an instruction execution using the micro-operations; and
- describe the basic organisation of ALU;
- discuss the requirements of a floating point ALU;
- create simple arithmetic logic circuits.

10.2 REGISTER ORGANISATION

The number and the nature of registers is a key factor that differentiates among computers. For example, Intel Pentium has about 32 registers. Some of these registers are special registers and others are general-purpose registers. Some of the basic registers in a machine are:

- All von-Neumann machines have a program counter (PC) (or instruction counter IC), which is a register that contains the address of the instruction that is expected to be executed next.
- Most computers use special registers to hold the instruction(s) currently being executed. They are called instruction register (IR).
- There are a number of general-purpose registers, which can be used for arithmetic computations or any other purpose.
- Memory-address register (MAR) holds the address of next memory operation (load or store).
- Memory-buffer register (MBR) or Memory data Register (MDR) holds the content of memory operation (load or store).
- Processor status bits indicate the current status of the processor. Sometimes it is combined with the other processor status bits and is called the program status word (PSW). Some processors also use flags register, which store different flags set by the processor like carry flag, overflow flag, zero flag etc.

The CPU registers have the following characteristics:

- CPU can access registers faster than it can access main memory.
- Register addressing requires less bits in the instructions for addressing than that of memory addressing. For example, for addressing 256 registers you just need 8 bits, whereas the memory size of 1MB would requires 20 address bits, a difference of 60%.
- Compilers tend to use a small number of registers, as large numbers of registers are difficult to use effectively. A general good number of registers is 32 in a general machine.
- Registers are more expensive than memory but far less in number.

From a user's point of view, computers have two different kinds of registers. These are:

Programmer Visible Registers: These registers can be used by machine or assembly language programmers while programming. A good program minimizes the references to main memory.

Status Control and Registers: These registers cannot be used by the programmers but are used to control the CPU or the execution of a program.

Different chip designer companiesuse some of these registers interchangeably; therefore, you should not stick to these definitions rigidly. Yet this categorization will help in better understanding of register sets of a machine. Therefore, let us discuss more about these categories.

10.2.1 Programmer Visible Registers

These registers can be accessed using machine language. In general,there are four types of programmer visible registers.

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

A comprehensive example of registers of 8086 is given in Unit 1 Block 4.

The general-purpose registers as the name suggests can be used for various functions. For example, they may contain operands or can be used for calculation of address of operand etc. However, to simplify the task of programmers and computers dedicated registers can be used. For example, registers may be dedicated to floating point operations. Such dedication may lead to design of data and address registers.

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

The address registers are used for address computation. Some dedicated address registers are:

Segment Pointer	:	Used to point out a segment of memory.
Index Register	:	These are used for index addressing scheme.
Stack Pointer	:	Points to top of the stack when programmer visible stack addressing is used.

One of the basic issues with register design is the number of general-purpose registers or data and address registers to be provided in a microprocessor. The number of registers also affects the instruction design as the number of registers determines the number of bits needed in an instruction to specify a register reference. In general, the optimum number of registers in a CPU is in the range 16 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 results then must be stored in the memory. On the other hand, if the number of registers goes above 32, then there is no appreciable reduction in memory references. However, in some computers hundreds of registers are used. These systems have special characteristics. These are called Reduced Instruction Set Computers (RISC) and they exhibit this property. RISC computers are discussed in a later unit.

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: An important characteristic related to registers is the length of a register. Normally, the length of a register is dependent on its use. For example, a register, which is used to calculate address, must be long enough to hold the maximum possible addresses. If the size of memory is 1 MB than a minimum of 20 bits are required to store an instruction address. Please note how this requirement can be optimized in 8086 in the block 4. Similarly, the length of data register should be long enough to hold the data type it is supposed to hold. If the length of a data register is half of the size of data, then it is possible that two consecutive registers, rather than on single register, are used to store the data.

10.2.2 Status and Control Registers

For control of various operations several registers are used. These registers cannot be used in data manipulation; however, the content of some of these registers can be used by the programmer.Almost all the CPUs have a 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:

Flag	Comments
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 (contain a value 1) if the result of the last arithmetic operation was zero.
Carry flag	This flag is set, if a carry results from the addition of the highest order bits or 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	This flag is used for enabling or disabling interrupts. Enable/ disable flag.
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.

Figure 10.1: Flags or conditional codes

These flags are set by the CPU hardware while performing an operation. For example, an addition operation may set the carry flag and zero flag; or on a division by 0 the overflow flag can be set etc. These flags or conditional codes are tested by a program while performing typical operations like conditional branch operation. The condition codes are collected in one or more registers. RISC machines have several sets 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.

The flag register is often known as Program Status Word (PSW). It 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 etc.

☞Check Your Progress 1

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 you store status and control information in the memory?

   ....

   ....

   ....

10.3 MICRO-OPERATION CONCEPTS

We have discussed the general introduction to register set. A computer executes an instruction using the arithmetic logic unit and control unit in several steps. Each of these micro steps of instruction execution is elementary in nature and is termed as a micro-operation. In general, a micro-operation should be executed in a single clock pulse by the ALU or bus transfer unit using the data stored mostly in registers.

A machine instruction is equivalent to an assembly language instruction, with the main difference being that assembly instructions use mnemonics, while machine instructions use opcode, operand addresses etc. Thus, for simplicity of explanation, we will use assembly language instructions rather than machine instructions. It may be noted that a high-level language program, first is converted to machine instructions and is executed thereafter. For example, a C programming language expression A=A+B; may require the following sequence of machine/assembly instructions on a hypothetical machine that uses AC and DR registers:

```
LDA AC, A  ; Load the content of memory location A to AC register.
LDA DR, B  ; Load the memory operand B to DR register
ADD AC, DR ; The content of AC and DR is added and stored in AC
STR  A, AC ; Store the content of AC to memory location A.
```

Each of the above instructions is required to be executed by the computer. Consider the instruction LDA AC, A how will it be executed? A von Neumann machine may execute instructions in the following steps:

Step 1: Get the instruction from memory to the Instruction Register (IR): This step itself will consists of several micro-steps, which will require transfer of content among registers and using the bus.

Step 2: Decode the instruction: This will be the job of control unit (CU) and it will issue the necessary set of control signals. This step will be discussed in Unit 11.
Step 3: Fetch the operands, if needed, in the processing unit registers.
Step 4: Execute the instructions using arithmetic logic unit (ALU) and store the results back to processing unit registers.
Step 5: Store the result back to memory, if needed.

Please note that each of these steps may require several micro-steps and those micro-steps are the micro-operations. May of these steps are due to the specific functions of registers, for example, the Program counter (PC) register stores the address of instruction that is to be executed next. Thus, in order to get the next instruction from the memory, you are required to transfer this information to the register that is used as an memory address register. Similarly, instruction once fetched from memory may be in a data register, since IR is used as input for decode operation, the instruction must be sent to IR register. The micro-operations that are used for representing data transfer between two registers or one register and one memory location using the bus are termed as register transfer micro-operations.

In addition, to register transfer, instruction execution also involves the use of arithmetic logic unit. Some of these operations which are performed by ALU on register data are add, subtract, increment, decrement etc. These are called arithmetic micro-operations. In addition, ALU can also be used to perform the logic operations, like AND, OR, NOT etc., on the data of registers. These are called the logic micro-operations. Further, shifting of data to left or right by one bit are used for many useful operations like multiplication or division. Such micro-operations may be termed as shift micro-operations.

These microoperation can be written using a register/memory transfer language which is described next.

10.3.1 Register Transfer Language, Bus and Memory Transfers

Register transfer language can be used to represent various micro-operations. In addition, this language also can be used to represent bus and memory transfer. In this language the symbol $\leftarrow$ is used to indicate transfer of content. For example, if two registers are named R1 and R2, then a register transfer micro-operation $R1 \leftarrow R2$ implies that all the bits of register R2 are to be transferred to register R1. This operation will be feasible only if both the registers are connected through a data path and consists of same number of bits. Some of the basic rules/convention of register transfer language are listed below:

1. Naming of Register: Register would be named using capital alphabets and digits. The first character of the name should be alphabet. Figure 10.2 shows the bit organization and naming of register parts with the help of an example register IR, which is having 16 bits. A register can be partitioned in bytes and each byte may be assigned a name. For example, Figure 10.2 represent IR register of 16 bits, which can have two 8-bit partitions. You can refer to each of these bytes separately if so desired. For example, the lower byte of IR can be referred to as IR(L) and higher byte of IR can be referred to as IR(H) (Please refer to Figure 10.2)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR(H)								**IR(L)**							
IR															

Figure 10.2: Register Naming and bit ordering

2. Information transfer from one register to another is designated by the symbol $\leftarrow$, which replaces the content of the destination register by the content of the source register, as explained earlier. The content of source register remains unchanged.

3. The control signal which enables the process of register transfer is shown with the help of a Boolean control function. This feature is very useful as operation can be controlled. For example, if *c* is a Boolean control variable that is it can have a value of 0 or 1,which controls the transfer of content from R2 to R1, then it will be represented as:

c: R1$\leftarrow$ R2 ; Please note this transfer will occur only if *c* = 1.

4. The micro-operations written on the same line are executedin the same clock time. However, such micro-operations should be free from conflict. The following example depicts a conflict in micro-operations; thus, these micro-operations should not be executed in parallel. What is a conflict in micro-operations? It is explained with the help of following example.
   Example: Consider that a register R1 is to be incremented and it is also to be loaded with the content of IR register, then if you represent the following parallel micro-operations would be in conflict:
   c: Rn$\leftarrow$ Rn+1, Rn$\leftarrow$IR
   These two micro-operations would update the register Rn at the same time, so one of these updated values would be lost.

5. All the register transfers occur during the falling edge transition of the clock. For more details, you may refer to further readings.

Use of Bus in register transfer

As explained earlier, a computer system may have about 16-32 or even more (in case of RISC) registers. A computer system, in general, uses an internal Bus, which consists of bus lines or wires, to transfer register data bits between two registers at a time. The control unit selects these two registers – one as the sources register and second as the destination register. Thus, the register transfer operation can be expanded as follows:

c: R1 $\leftarrow$ BUS $\leftarrow$ R2

The control unit causes the register R2 to put its content (one bit on one bus line). In addition, it also enables the load operation on the R1 register. Thus, the data bits of R2 are transferred to data bits of R1.

Memory Transfer

A von Neumann machine stores the program and data in the main memory of a computer. Therefore, instruction execution requires reading and writing operations from the memory. The main memory and the processing units of a computer are connected through the system bus, which includes address, data and control bus. The address bus is used to select the specific RAM word from the memory, which in turn is transferred over the data bus. The control unit controls the entire process of data transfer by sending control signals through control bus. The two basic operations performed on the memory are Read and Write operations.
Memory Read: Read operation on the memory requires the information about the location of the memory, which is to be read. The processor decides which data word or instruction word from the memory is to be read. The address of that memory word is put in an address register (AR) and then applied on the address bus, simultaneously enabling the memory read operation using the control bus. This causes the selected word from the memory unit to be placed on the data bus part of system bus. Control unit also enables data register (DR) to accept the input from the data bus. Thus,

completing the memory read operation. The following micro-operation shows this memory read operation (please note the use of [ ] symbol to represent memory):

mr: DR ← [AR] ; Content of memory addressed by AR is send to DR register

Memory Write: Write operation on the memory requires– the location of the memory, which is to be written and the content, which is to be written. The processor puts the address of memory word, which is to be written, to an address register (AR) and then applies this address on the address bus, simultaneously loading the content of data to be written, which may be stored in a data register (DR) and enabling the memory read operation using the control bus. This causes the selected word on the memory unit to accept the data bus. control unit also enables data register (DR) to accept the data from the data bus. Thus, completing the memory write operation. This operation can be represented using following micro-operation:
mw: [AR] ←DR ;Write the content of memory addressed by AR register by DR

Please note, in this micro-operation the memory location, as addressed by AR, is written into. The content of AR and DR registers remains unchanged

10.3.2 Register Transfer Micro-operations

Register transfer micro-operation is one of the basic micro-operations. Two basic requirements for such a transfer are:

- There should exist a direct path, such as internal bus, from sender register to receiver register. It may be noted that number of bus lines and the sizes of sender and receiver register should be the same.
- Since a micro-operation is proposed to be completed in a single clock pulse, therefore, all the data bits on the receiver register should be loaded at the same time. Thus, the receiver register must support parallel loading of bits (Refer to Unit 4 Block1).

Following are some of the examples of register transfer micro-operations:

R1← R2	; Transfer the content of R2 register to R1 register
AR← PC	; Transfer the content of PC register to AR register

10.3.3 Arithmetic Micro-operations

Arithmetic micro-operations are performed by the arithmetic logic unit on the data stored in the processor registers. The output is also saved in a processor register. Following are some common arithmetic micro-operations:

AC←AC+DR	; Addition of AC and DR, result in AC
AC ← AC-DR	; Subtraction of DR from AC, result in AC
AC ← AC+1	; Increment AC, result in AC
AC ← AC-1	; Decrement AC, result in AC

An adder subtractor circuit as shown in Unit 3 of Block 1 may be used to perform subtraction in machines through complement and add operations. It is represented as:

R3 ← R1 0 R2

Please note that this micro-operation can also be represented as:

R3 ← R1 + R2′ +1

Why? R2′ is complement of R2, on adding 1 to it you get 2's complement of R2. Thus, both the above micro-operations are equivalent.

An addition and subtraction micro-operations can be implemented using an ALU that supports simple arithmetic operations, as discussed in section 10.4. Assuming that this ALU does implement the addition, subtraction, simple logic and shift micro-operation of fixed-point numbers, how will this machine implement multiplication and division operations on fixed point and floating-point numbers? In such a machine these operations can be implemented with the help of programs, which may use micro-operations like addition, shift and so. This kind of implementationrequires that operations like fixed point multiplication and division be implemented using several micro-operation steps using several micro-instructions. Thus, fixed point multiplication and division and floating point arithmetic instructions cannot be considered as micro operations for this kind of machine.

10.3.4Logic Micro-operations

A computer system is a binary device. Itperforms binary operations using bitwise Boolean operations. These bitwise Boolean operations that are implemented in the ALU forms the logic micro-operations.Three most common logic micro-operations are AND(.), OR(+) and NOT(~). Other logic micro-operations, which may be implemented in different computers can be XOR, NAND and NOR. For example, you can perform bit wise AND of two registers R1 and R2 using the AND micro-operation, which can be represented as:

R1 ← R1.R2

The result of the micro-operation, as given above, will be stored in the R1 register. A typical use of this micro-operation is shown in the following example.

Example 1: Assume that two four-bit registers R1 and R2 contains the data 1100 and 1010 respectively. What would be the output if following micro-operations are performed on these two registers:

i. R1 ← R1 . R2
ii. R1 ← R1 + R2
iii. R1 ←~R1

Solution:

i. 1100 .1010 = 1000
ii. 1100 +1010 = 1110
iii. ~1100 = 0011

Example 2: Consider a register A containing an 8-bit value 01010011. Find the value of register B and micro-operation, which can be used to set the upper four bits of the register A, while the lower four bit remains unchanged.
Solution: To set the upper four bits irrespective of the values of A while keeping the lower four bits unchanged, the register B can consist of value 11110000 and the micro-operation OR can be used, as shown below:

Register A	0	1	0	1	0	0	1	1
Register B	1	1	1	1	0	0	0	0
A OR B	1	1	1	1	0	0	1	1

Thus, in the output the upper four bits contains value 1, while lower four bits are same as that of register A.

Example 3: Use the same value of register A, as used in example 2. Find the value of register B and micro-operation, which can be used to clear the upper four bits of the register A, while the lower four bit remains unchanged.
Solution: To clear the upper four bits irrespective of the valueof A while keeping the lower four bits unchanged, the register B can consist of value 00001111 and the micro-operation AND can be used, as shown below:

Register A	0	1	0	1	0	0	1	1
Register B	0	0	0	0	1	1	1	1
A OR B	0	0	0	0	0	0	1	1

Thus, in the output the upper four bits contains value 0, while lower four bits are same as that of register A. This operation is sometimes also referred to as mask operation, where the upper four bits of the register A are masked out.

Example 4: Use the same value of register A, as used in example 2. Find the value of register B and micro-operation, which can be used to clear all the bits of a register.
Solution: To clear the entire register A, you can use register B same as that of register A and use XOR micro-operation, as shown below:

Register A	0	1	0	1	0	0	1	1
Register B	0	1	0	1	0	0	1	1
A XOR B	0	0	0	0	0	0	0	0

Example 5: Use the same value of register A, as used in example 2. Find the value of register B and micro-operation, which can be used to clear all the bits of a register.
Solution: One of the simplest ways to complement a register is to perform NOT micro-operation on register itself. An alternative native method would be to perform XOR with register B containing all 1's, as shown below:

Register A	0	1	0	1	0	0	1	1
Register B	1	1	1	1	1	1	1	1
A XOR B	1	0	1	0	1	1	0	0

10.3.5Shift Micro-operations

Shifting of bits of a register can be used for several useful functions in a computer, such as serial transfer of data, multiplication operation, division operation. A register consists of a linear sequence of bits, which can either be shifted towards the left direction or right direction. Shifting by one bit, irrespective of left or right shift, will result in one bit to move out of the shift register from one end and one bitwill be input to the register over the other end. Shift operations have been discussed in the Unit 9. Shift operations are of three basic types:

1. Logical Shift: In the logical shift, the input bit is kept as 0 and output bit is discarded.
2. Arithmetic shift: In arithmetic shift the sign of the number is kept the same
3. In circular shift the output bit is circulated to the input.

☞ Check Your Progress 2

1. Explain how the memory read and memory write micro-operations can be performed in a von Neumann machine

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2. A simple ALU just performs the addition operation, logic operations and shift operations. Will multiplication on this machine be implemented as a micro-operation? Justify your answer.

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

3. Consider a register R1 contains 00010011. Select a suitable register R2 and sequence of logic micro-operations that can perform the following tasks:

(i) Reset the complete register R1, you must use this using Mask micro-operation and not using XOR.
(ii) Insert the 11001000 data into the register R1. You may use more than one micro-operation to do this task.

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

10.4 INSTRUCTION EXECUTIONS AND MICRO - OPERATIONS

To execute an instruction, a sequence of micro-operations needs to be executed. This sequence of micro-operations is essentially controlled by using a timing signal. In addition, an instruction execution involves operands. These operands can be stored either in the registers orthe memory of a computer. In other words, every computer has register reference and/or memory reference instructions. Finally, in general, input-output on a computer, as discussed in Block 2, may use input-output interrupts. The next sub-section discusses the importance of these concepts in the context of instruction execution. This is followed by discussion on instruction cycle and interrupt cycle.

10.4.1 Relationship of Timing and Control, Memory reference instructions and Input-output Interrupts to instruction execution

Timing and Control: The basic task of a computer system is to execute instructions. Each instruction is executed as a sequence of steps. The execution of instructions follows a typical sequence, which is termed as instruction cycle, which is discussed next sub-section. Thus, each instruction cycle consists of several steps, which are executed in a sequential orderto execute the instruction correctly. How will computer ensure that the exact sequence of instruction is followed?

As discussed in Unit 4, a computer system uses a clock for synchronization. The role of a clock is to continuously emit a sequence of clock pulses. In addition, Unit 4 also highlighted few important sequential circuits like registers, counters, multiplexer etc. Most of the sequential circuits change their state on the falling edge of the clock pulse. Thus, a particular action like loading of a register will be complete at the falling edge of the clock pulse. The next sequential action will be completed in the subsequent clock pulse and so on. For example, loading of a register may require one clock pulse, which may be followed by some other operation on the register in the next clock pulse. The counter is the circuit, which can be used the count the sequence of clock pulses. Thus, timing sequences are utilized to control the sequence of operations in a computer system. You may refer to the further readings for more details on these topics.

Memory Reference Instructions: The memory reference instructions access its operands from the memory. In general, these operands may use direct or indirect addressing schemes. In the direct memory access scheme, the address part of the instruction contains the direct reference of a memory operand, or in other words the address part of the instruction contains the address of the operand. Thus, in order to process such instructions these operands are to be fetched from the memory to some data register of the processor. In the indirect memory reference, the instruction address is the operand address. Thus, two memory accesses would be needed to fetch such operands. First to get the address of the operand and second to get the operand from the memory.

It may be noted that a memory reference, in general, is slower than the reference to a register, as main memory is somewhat slower than the registers. However, with the use of fast cache memories, the difference in speed of the memory access vis-à-vis register access is being reduced. Many memory organizations have also been designed to reduce this speed gap. You may refer to the further readings for more details on these topics.

Input-output Interrupts: Interrupt is a condition, which may result in break in the sequence of instruction execution. Interrupts are used in computers for input-output as well as recovering from a condition that may cause an error, for example, division by zero. An Input-Output interrupt can be used for Input or output of data. For example, a program, while it is being executed by the processor, may have a command for reading of data from a keyboard. The processor would display the prompt for input of data for this program and may go on to execute other programs, while suspending this program as it needs the input. However, as soon as you enter the input from the keyboard, it may issue an interrupt to the processor, which in turn will process the input. More details on Interrupts are already given in Block 2. You may also refer to further readings for more details on interrupts.

Next section discusses the instruction cycle.

10.4.2 Instruction Cycle

As discussed in Unit 1, a von-Neumann machine having a basic set of registers executes instruction. These instructions consist of the following steps of execution:

Step 1: Get the Instruction from the memory to IR, the instruction register, let us call this operation as Fetch the Instruction (FI).

Step 2: Decode the Instruction, which is in the IR register, let us call this operation as DI.

Step 3: In this step operand address is converted to a direct address. are fetched, let us call this step as fetch the direct operand address (FoA).

Step 4: Perform the execution of instruction, let us call this step as execute instruction (ExI)

Step 5: Once execution of instruction is complete, check and acknowledge interrupt if an interrupt has occurred, let us call this step as check and acknowledge interrupt (CaI)

How are these steps can be translated to micro-operations? This section uses the register set as given in Section 10.2. In addition, it assumes the instruction format as given in Unit 9, where an instruction has three fields, viz. Indirect bit, opcode of 7 bits and one operand address of 24 bits. It may please be noted that the assumed instruction set has only direct memory and indirect memory addressing schemes. Further, only for the purpose of simplifying the discussion, it is assumed that memory reference and register reference is performed in almost equal time.

Step 1 (FI cycle): An instruction is available in the memory and program counter register (PC) points to this instruction, which is to be executed. Thus, in order to get the instruction from the memory, bus would be used along with memory address register (MAR). Thus, a sequence of operations would be needed to perform this operation. This sequence is represented with the help of a timing sequence using the timing control T1, T2, etc. Please note the micro-operation with timing control T1 will be performed prior to micro-operations with timing control T2 and so on. Figure 10.4 lists the micro-operation sequence of FI cycle.

… Transfer the content of PC to MAR.	T1: MAR ← PC
… Apply MAR on address BUS; Control unit enables the memory Read operation and DR is enabled to receive content on the data BUS. Thus, the content of memory location pointed by MAR is read to DR.	T2: DR ← (MAR)
… Perform the following two operations in parallel at time T3: o Increment PC so that it points to the next instruction to be executed. (PC is incremented by one memory word length, as it is assumed that each instruction is just one word long and memory address is a word address).	T3: PC←PC +1
o The instructionin DR is sent to IR to complete the FI cycle.	: IR ← DR

Figure 10.4: Fetch cycle

Step 2: DI: Control unit performs the decoding of instruction. It identifies the two important things; first what operation is to be performed and second what addressing modes are used by the instruction. The addressing modes are to be decoded so that the data is brought in the ALU registers for executing the decoded operation. Since decode operation is performed by the control unit, more details related to this operation would be discussed in Unit 11.

Step 3: FoA: In this step the operand address is converted to the direct operand address and is stored in the address part of instruction. This address is used in the next step for instruction execution. In the case of the present instruction format only two possible addressing types – direct and indirect. In the case of direct addressing the address of the operand is already in the operand address part of instruction, thus, no additional micro-operation is needed. However, in case of indirect addressing, the address of the operand is to be fetched using the address portion of the instruction. This fetched address should replace the current address portion of the instruction. This step is shown in Figure 10.5:

When Direct Addressing is used: … No action is needed.	IR (Address) contains the direct address of the operand.
When Indirect Addressing is used:	
… Transfer the operand address portion of instruction to MAR.	T1: MAR ← IR(Address)
… Read the memory using MAR and bring operand address in DR	T2: DR←(MAR)
… Transfer address from DR to IR. Now, IR has a direct address.	T3: IR(Addrss)← DR(Address)

Figure 10.5: An Indirect cycle

Thus, the IR now contains the direct address of the operand.

Step 4: ExI: The instruction execution is also performed with the help of micro-operations. The first step of instruction execution will require the operand to be brought from the address of main memory to the processor register. This is followed by the arithmetic micro-operation as per the requirements of instruction opcode. The following examples explains the micro-operations required to execute certain instructions.

(1) The step required to execute a simple addition instruction of the form (assuming that indirect bit is set to 0, i.e., it is a direct instruction):

ADD ADR

The following sequence of micro-operations, as given in Figure 10.6, would execute this instruction (after FI cycle):

… Transfer the ADR in instruction to the MAR. … Load DR by reading the memory location referred to by ADR. … Control units then enables addition of AC and DR using ALU. Result is put in AC.	T1: MAR ← IR(ADR) T2: DR ← (MAR) T3: R1 ←R1 + DR

Figure 10.6: Execution cycle of Add instruction

(2) Micro-operations for execution of a conditional jump instruction, which skips the next instruction, if on incrementing the operand results in zero. The incremented value is stored back to the operand location.

INCSKP ADR

The sequence of micro-operations required for this instruction execution are given in Figure 10.7.

Step 1: Bring the operand stored in location ADR to a processor register AC. AC is incremented. … Transfer the ADR of IR to the MAR. … Read ADR to DR … Transfer DR to AC, as increment operation can be performed in AC (assumption). … Increment the AC. This will set the Flag register.	T1: MAR ← IR (ADR) T2: DR ← (MAR) T3: AC← DR T4: AC←AC +1
Step 2: Store the AC to ADR … Transfer the content of AC to DR. … Store DRinto ADR using MAR and If the content of AC is zero then the next instruction is not to be executed or skipped, to do so increment the value of PC. The control unit checks the zero flag and increments PC if condition is fulfilled.	T5: DR ← R1 T6: (MAR) ←DR T6: If AC = 0 then PC ←PC + 1

Figure 10.7: Execution cycle of increment and skip instruction

(3) This example shows the sequence of micro-operations required for a branching operation, using a subroutine call instruction. A subroutine call instruction is required to store the return address and then start execution of the subroutine, which will require the change in the value of the PC register. The return address, in general, is stored on a stack, however, for this example, it is assumed that the return address is stored in the subroutine address (ADR) specified in the callinstruction. Assume the following is a subroutine call instruction:

SUBCALL ADR

The sequence of micro-operations to execute this instructionare given in Figure 10.8.

… Transfer ADR to MAR and the return address, which is in PC is put in the DR.	T1: MAR ← IR(ADR) T1: DR ← PC
… To branch to the subroutine, the ADR should be moved to PC. Further, at this address (ADR), the return address is to be put. Please note that ADR is already in MARat time T1.	T2: PC ← IR (ADR) T2: (MAR) ←DR
… The first instruction of the subroutine starts at the ADR plus one, thus, increment PC.	T3: PC ←PC + 1

Figure 10.8: Execution cycle of subroutine call instruction

It may be noted that the sequence of micro-operations required to perform instruction execution, viz. FI, DI, FoA and ExI are machine dependent. In this section, we have presented a very simple example for the same.

10.4.3 Interrupt Cycle

Interrupt Processing: In addition to execution of the instruction, the processor should also respond to the interrupt, which may have occurred while instructions of a program are getting executed. It may be noted that programs may have priority and may allow only some of the interrupts, while they are executing. Thus, every computer has provision of enabling interrupts as per their priority.Now, the question is how the processor will check if an enabled interrupt has occurred? One of the solutions to this problem is to keep a control line for interrupt signal and check it after each instruction cycle. Next question is how to acknowledge an interrupt? To acknowledge an interrupt, processor may perform an interrupt cycle, as given in Figure 10.9. The interrupt cycle has been explained assuming that a stack is used to store the address of the instruction, from where the program will be restarted once the interrupt is processed.

… The return address is in the PC register, which is to be stored on a stack, named STACK, with stack top pointed by stack pointer register (SP). It is being represented as STACK[SP]. DR register is to be used for this operation.	T1: DR ← PC T2: STACK[SP]←DR
… Increment the stack pointer to next location and start executing the interrupt servicing program by transferring the address of its first instruction, say ISRAddress, to PC.	T3: SP ←SP+1 T3: PC ←ISRAddress

Figure 10.9: Interrupt cycle using a stack

It may be noted that even interrupt servicing programs are a sequence of instructions. Each of these instructions are executed as per instruction cycle. You may refer to the further readings for more details on instruction cycle.

10.5 INSTRUCTION PIPELINING

In the previous section, you have gone through various steps of instruction execution. Can these steps be performed in parallel to execute an instruction? The answer is NO, as to execute an instruction these steps are to be performed in sequence. However, can the steps of different instruction be performed in parallel to each other or can be executed in an overlapped manner. Execution of several instructions in parallel will require several processing elements, rather breaking the execution of instruction into steps and executing instruction in an overlapped manner may be useful. This is the the principle of instruction pipelining. Thus, a simple instruction pipeline would require to execute instructions in an overlappedway, which will facilitate and reduce the time of

overall instruction execution. This would require that instruction cycle should be divided into equal parts which can be executed in parallel for different stages of instructions. One such decomposition of instruction cycle stages, also called pipeline stages are -fetch the instruction (FI), decode the instruction (DI), fetch operand address(FoA) and execute the instruction (ExI). A new stagehas been added here that allows storage of result back to the memory location, let us call it StR.
Figure 10.10 shows the overlapped execution of seven instructions using a five stage instruction pipelining.

Time Slot	1	2	3	4	5	6	7	8	9	10	11
Instruction 1	FI	DI	FoA	ExI	StR						
Instruction 2		FI	DI	FoA	ExI	StR					
Instruction 3			FI	DI	FoA	ExI	StR				
Instruction 4				FI	DI	FoA	ExI	StR			
Instruction 5					FI	DI	FoA	ExI	StR		
Instruction 6						FI	DI	FoA	ExI	StR	
Instruction 7							FI	DI	FoA	ExI	StR

Figure 10.10: Instruction Pipeline

In Figure 10.10, you may notice that at time slot time slot 5 the pipeline is executing 5 instructions simultaneously, though in different stages. At the end of time slot 5, execution of the first instruction is completed, thereafter, at the end of each time slot a new instruction would be completed or in other words the first instruction gets completed at the end of time slot 5, the second instruction gets completed at the end of time slot 6, the third instruction gets completed at the end of time slot 7 and so on.Thus, the execution of instruction in an overlapped fashion has resulted in almost one instruction execution in one time slot.However, the instruction pipelines suffer from the problem of resource conflicts. For example, in the 5th time slotThe first instruction is storing the result, therefore, would require memory reference; at the same time slot the second instruction is in the execution stage, thus, it will fetch the operand using memory reference and use the ALU to execute this instruction; at the same time the third instruction is in operand fetch stage, which also requires memory reference; the fourth instruction is in the decode stage; and the fifth instruction is in the fetch instruction stage, which also requires memory reference. Thus, the pipelined processor should allow each of these instructions to reference memory simultaneously through different paths so that there is no conflict, otherwise the instructions cannot be executed in the pipeline fashion.

The Problems relating to Pipelined execution:

- Pipelined execution may suffer from resource conflict as explained above.
- The pipelined execution seems good for execution of sequence of instruction, but instructions that require transfer of control, like conditional branch instruction, may cause disruption of pipeline sequence, as the decision whether a branch will be taken or not, can occur only at the execution stage. In case a branch is to be taken then all the subsequent instructions, which were already fetched in the pipelineare to be removed from the pipeline.

Some solutions for problems related to pipelined execution:

The branch penalty can be minimized using any of the following schemes:
- Predicting, if branch will be taken or not and accordingly fetching the next instruction.
- Making provision of pre-fetching those instructions, which may be executed because of a branch.
- Not allowing fetching of the next instruction to pipeline till the branch decision is made.

Check Your Progress 3

1) What is the need of the indirect cycle? Will indirect cycle be needed even if an instruction use register addressing schemes? Justify your answer.

…………………………………………………………………………………….

……………………………………………………………………………………..

…………………………………………………………………………………….

2) What is fetch cycle? Do the present-day machines also have this cycle?

…………………………………………………………………………………….

……………………………………………………………………………………..

…………………………………………………………………………………….

3. What is the role of Interrupt cycle?

…………………………………………………………………………………….

…………………………………………………………………………………….

……………………………………………………………………………………..

10.6 ALU ORGANISATION

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. A simple ALUs can be constructed for performing computation on fixed-point numbers. An ALU for floating-point arithmetic implementation requires more complex control logic and data processing capabilities. 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 processors. Let us discuss all these issues in greater detail in this section.

10.6.1 A Simple ALU Organisation

An ALU consists of circuits that perform data processing micro-operations. Figure 10.11 shows the organisationof a fixed point ALU was suggested by John von Neumann in his IAS computer design.

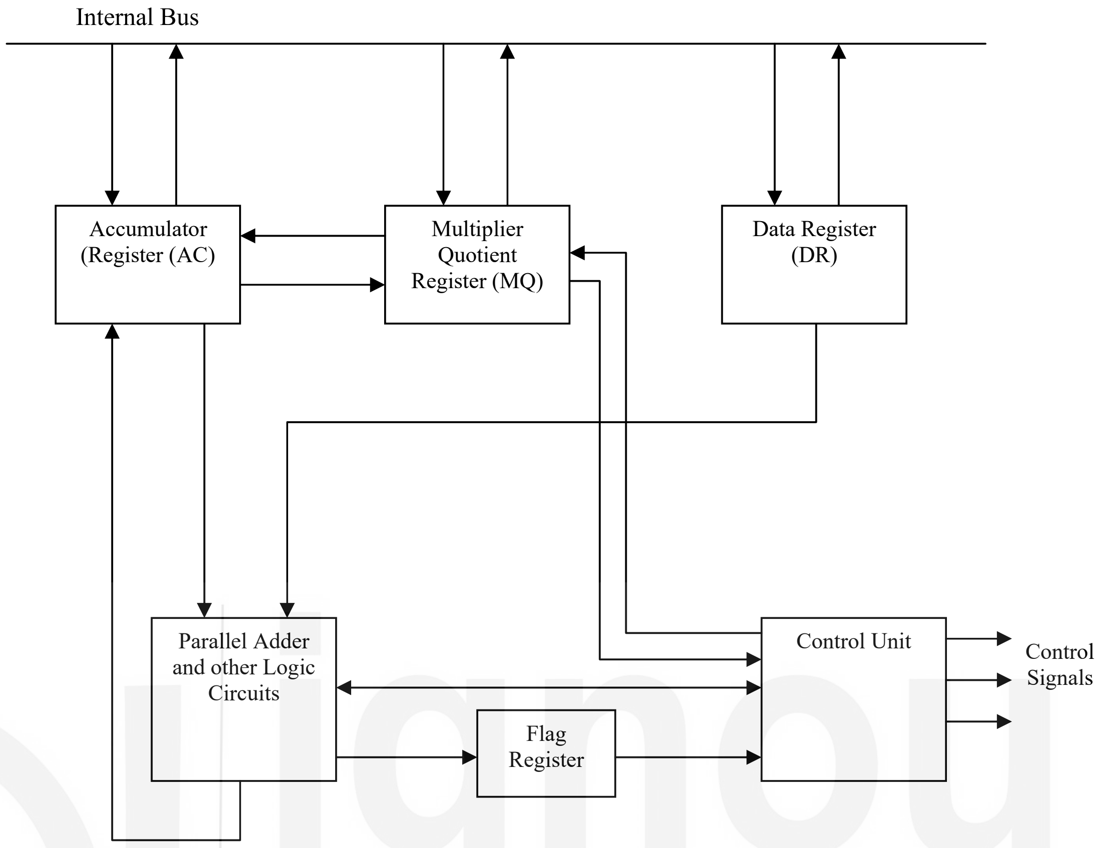

Internal Bus
Accumulator (Register (AC)
Multiplier Quotient Register (MQ)
Data Register (DR)
Parallel Adder and other Logic Circuits
Flag Register
Control Unit
Control Signals

Figure 10.11: Structure of a Fixed-point Arithmetic logic unit

The above structure consists of three registers AC, MQ and DR, with the assumed size of one word each. Please note that the Parallel adders and other logic circuits (these are the arithmetic, logic circuits) this von Neumann machine can have at most two inputs and one output. In other words, 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 Figure 10.11, 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	: $AC \leftarrow AC + DR$
Subtraction	: $AC \leftarrow AC - DR$
AND	: $AC \leftarrow AC \wedge DR$
OR	: $AC \leftarrow AC \vee DR$
Exclusive OR	: $AC \leftarrow AC \oplus DR$
NOT	: $AC \leftarrow AC'$

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 instructions. Please note that in the ALU shown in Figure 10.11, the multiplication and division instructions are not implemented directly using the logic circuits. 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. Please note that these are not micro-operations for the given ALU organization, as execution of these two instructions would require a series of shift-add operations.

DR is another important register, which is used for storing second operand. In fact, it acts as a buffer register and stores the data brought from the memory. In machines where we have general purpose registers any of the registers can be utilized as AC, MQ and DR. For more details on ALUs, you can go through the further readings.

10.6.2 A Sample ALU Design

ALU consists of circuits that executes the micro-operations. The data is input to ALU through registers and the output of ALU is also stored in an output register. For performing the input/output of data to ALU, a BUS is used. So, let us first explain how an internal bus can be used for Data transfer.

A computer processor has large number of registers. These registers are used to store data that is required to be processed by the ALU.But, how is the datacommunicated among these registers? One possibility is to create separate data paths from every register to all other registers, however, this connection structure would waste large amount of processor resources. Therefore a shared media call internal BUS. A BUS consists of shared data lines, which are then connected to every register. Using these shared lines any two registers can communicate with each other, at a time. The number of lines in the shared BUS is kept same as the size of registers.

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 lines, are made using the multiplexercircuit.Figure 10.12 shows an example of 2-bit data bus using 2×1 multiplexers. Please note the size of the registers is also of two bits.
The construction of a bus system for two 2-bit registers using two 2×1 multiplexers is shown in the Figure 10.12. Each register has two bits, viz. Bit 1 and Bit 0. Each multiplexer has 2-bit data input, numbered 0 and 1, and one control or selection lines, C_0. The circuit assumes no enable bits. The 0thdata input of MUX 0is connected to the corresponding Bit 0 of Register A and the 1st data input of MUX 0 is connected to Bit 0 of Register B. Similarly, the Bit 1 of Register A and Bit 1 of Register B are connected to 0th data input and 1st data input of MUX 1 respectively. There is just 1 selection line S, when S is 0, then 0th input values for MUX 0 and MUX 1 are transferred to the output, that is the content of Register A is transferred on the bus, whereas, when S is 1 bits of Register B are selected for transmission on the bus.

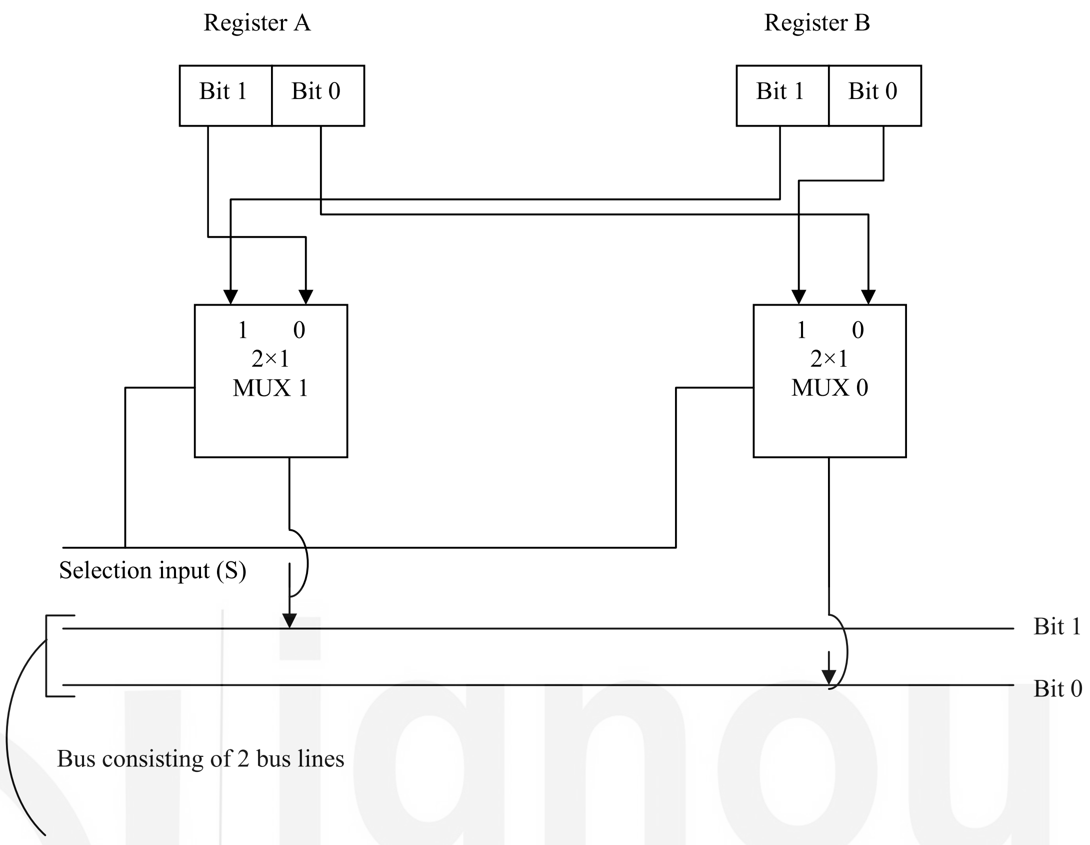


Figure 10.12: Implementation of a 2-bit BUS

The Figure 10.13lists the selection of data based on the selection input to the multiplexers.

S	MUX 1	MUX 0	Comments
0	Bit 1 of Register A	Bit 0 of Register A	Content of Register A is put on the bus lines.
1	Bit 1 of Register B	Bit 0 of Register B	Content of Register B is put on the bus lines.

Figure 10.13: Bus Line Selection

Thus, to construct a bus for 2 registers of 2-bits each, you would require two 2×1 multiplexers. Similarly, to construct a bus for 8 registers of sixteen bits each, you would require sixteen 8×1 multiplexers, which will have 3 selection input. Please note one multiplexer is needed for transfer of one bit. Since sixteen bits are to be transferred, therefore, sixteen multiplexers would be needed. Further, one of the 8 registers would be selected to transfer data on the BUS, therefore, 3 selection input would be needed, as $2^{3} = 8$.

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, one such circuit is shown in Unit 3 of Block 1. Figure 10.14 shows a simple circuit for a 2-bit arithmetic circuit. The circuit is constructed by using 2 full adders and two 2×1 multiplexers, which requires just one selection input. Please recollect that afull adder circuit adds two input bits and one carry-in bit to produce one sum-bit and one carry-out-bit.

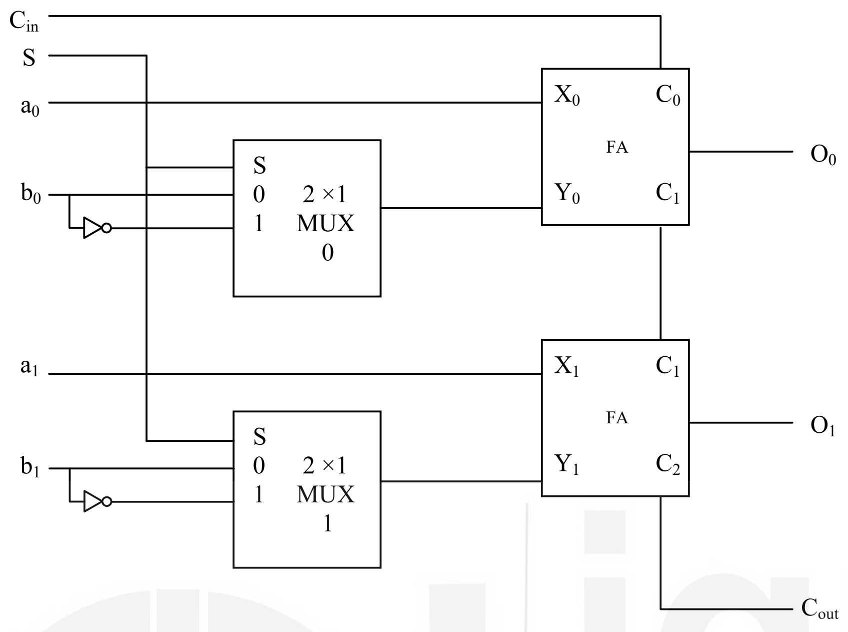


Figure 10.14: A two-bit arithmetic circuit (adder-subtractor)

The multiplexer controls one of the input to the circuit resulting in a set of micro-operations. Let us find out how the multiplexer control lines will change one of the Inputs for Adder circuit. Figure 10.15 shows the two inputs that are possible in the Figure 10.14. (Please note the convention used in this table, viz. uppercase alphabet indicates a 2-bit data word, whereas the lowercase alphabet indicates a bit.)

Control Input	**Output of 4 × 1 Multiplexers**		**Y input to Adder**	**Comments**
S	**MUX 0**	**MUX 1**		
0	b_0	b_1	B	The data word B is input to Full Adders
1	Complement of b_0	Complement of b_1	B′	1's complement of B is input to Full Adders

Figure 10.15: Input to full adders using the multiplexers in Figure 10.14

Now let us discuss how by using the carry-in-bit (C_{in}) and these input values, you can obtain various micro-operations.

Input to Circuits

- Register A bits as $a_0$and a_1, are input to X_0 and X_1 bits of the Full Adders (FA).
- Register B bits are input as given in the Figure 10.15to form the Y input to FA.
- Please note that each bit of register A and register B is fed to different full adder unit.
- Please also note that the A input directly goes to adder but B input can be manipulated through the Multiplexers to create different input values, as shown in Figure 10.15. The B input is controlled by the selection line S.
- The input carry C_{in}, which can be equal to 0 or 1, is input to the full adder that adds the least significant bits. The carry out of this full adder is then fed to the

full adder of the next higher bit and so on. The carry out of the most significant bit full adder is the output of the circuit. Logically it is the same as that of addition operation performed by us. We do pass the carry of lower digits addition to higher digits. The following Boolean function represents the output of this adder circuit:

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

Please note that in Figure 10.15, the value of X is a direct input, but the value of Y is input through the multiplexer using the selection input S. In addition, the value of C_{in} is another input. The arithmetic micro-operations that can be implemented using Figure 10.15 are given in Figure 10.16.

S	$\mathbf{C_{in}}$	**Y**	$\mathbf{O = X+Y+C_{in}}$	**Equivalent Micro-Operation**	**Micro-Operation Name**
0	0	B	O = A + B	R ← R1 + R2	Add
0	1	B	O = A + B + 1	R ← R1 + R2 + 1	Add with carry
1	0	B′	O = A+B′	R ← R1 + R2′	Subtract with borrow
1	1	B′	O = A+ B′ + 1	R ← R1 + 2's complement of R2	Subtract

Figure 10.16: Arithmetic Micro-operations implemented using Figure 10.15

Let us refer to some of the cases of the Figure 10.16.

When S= 0, input line B is applied directly to the Y inputs of the full adder. Now,

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

If you choose S= 1, then B′ forms the Y input to the full adder. So,
If C_{in}= 1, then output D = A + B′ + 1. This is called subtract micro-operation. (Why?)

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

```
  0111        (Value of A)
+ 1001       ( Complement of B)
------------------------------------------
1 0000 + (ignore the carry out bit and Add Carry in = 1)
= 0001
```
Thus, it is a subtract micro-operation.

If C_{in}= 0, then D = A + 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, is equivalent to:

O = A +B′
O = (A – 1) + (B′ + 1)
=> O = (A – 1) + 2's complement of B
=> O = A – (B+1)   Thus, is the name subtract with borrow

Two special cases:
When S= 0, C_{in}=1 and Y input is ZERO.

The output O = A + 0 + C_{in} => O = A + 1

This micro-operation is an increment micro-operation.
When S= 1,C_{in}=0 and input word Y has all 1's.

Output O = A + All (1s) + C_{in} => D = A – 1 (How? Let us explain with the help of the following example).
This is a decrement micro-operation.

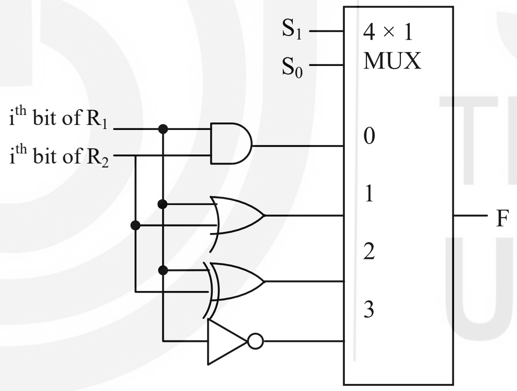
Example: Let us assume that the Register A is of 4 bits and contains the value 0101 and it is added to an all (1) value as:

```
  0101
+    1111
1   0100
```

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

Implementation of Logic Micro-operations

In many computers only four logic micro-operations, viz. AND, OR, XOR and NOT logicmicro-operations, are implemented. The other logic micro-operations can be derived from these four micro-operations. Figure 10.17 shows one bit, which is the i^th bit stage of the four logic operations. Please note that the circuit consists of 4 gates and a 4 × 1 MUX. The i^th 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	$F = R_1'$	Complement of Register R_1

(b) Functional representation

Figure 10.17: Logic diagram of one stage of logic circuit

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 shift registers but sometimes it can be made as a part of overall ALU. More details on ALU can be studied from the further readings.

10.7 ARITHMETIC PROCESSORS

Arithmetic processorswere needed in the older computer processors to perform arithmetic processing, especially the floating-point arithmetic, as those processorsdid not had the required logic circuits to directly handle such processing. They were also called co-processor, as they were used as an additional processor of some main processor. However, in this era of ultra-large-scaleintegration and beyond, all the fixed point and floating-point computational capabilities are built in the processor. The concept relating to arithmetic processor is explained below.

A typical processor 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 processor. 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 processor 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 processor 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?

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. This type of connection is called loosely coupled.

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 were termed as co-processors.

These days floating point units are implemented as a part of the processor itself. More details on these can be found in further readings.

☞Check Your Progress 4

1. Explain the implementation of ALU.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

Instruction fetch: fetching the

2. What is an Arithmetic Processor?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

10.8 SUMMARY

This unit discusses the concept of instruction execution for a hypothetical machine with the help of micro-operations. It also describes very simplified view of implementation of micro-operations using combinational and sequential circuits. The idea is to give you a basic information about the implementation of a computer system based on its instruction set. The unit also discusses the concept of register transfer language for representing the micro-operations. The unit also defined the concept of Instruction Pipeline. The unit also discussed the hardware implementation of micro-operations. The unit shows a simple implementation of bus, which is the backbone for any register transfer operation. This is followed by a discussion on arithmetic circuit and micro-operation there on using full adder circuits. The logic micro-operation implementation has also been discussed. Finally, the unit also discussed the arithmetic processors.
You may refer to the further readings for more details on micro-operation concept and instruction cycle.

10.9 SOLUTIONS / ANSWERS

Check Your Progress 1

1. An address register is used to store memory address or can be used to compute memory address of instructions or operands.
2. To address 20 registers, you may require address field of length 5 bits, as $2^5 = 32$, thus, about 12 addresses are unused.
3. Independent set of conditional codes can help in parallel checking of conditions.
4. Yes. Several operating system allocate memory space for storing such information for later use.

Check Your Progress 2

1. Memory read operation requiresthe address of the location to be read. This address is first applied on the address BUS and the control unit enables memory read. Thus, the content of the addressed location is put in the data BUS. At the same time a data register is enabled to store the data on the data BUS. These operations can be represented as: Address BUS ←MAR ;DR ← Data BUS. The overall operation can be represented as: DR ← (MAR)
   In memory write operationthe memory address, where data is to be written, is applied on address BUS and the data that is to be stored/written is put on the data BUS. At the same time memory write operation is enabled by the control unit. This operation can be represented as: (MAR) ← DR

2. No, as multiplication operation will be implemented using addition and shift micro-operations.
3. (i) AND with R2 containing 0000 0000
   (ii) Initially XOR of R1 with R1 will clear R1, then perform OR with R2 having value 11001000

Check Your Progress 3

1. Indirect cycle is needed, when indirect memory addressing is used by an instruction. It converts an indirect address to a direct address. If an instruction is using register addressing scheme, then indirect cycle may be required only if instruction is using register indirect addressing. The indirect cycle in this case would require simple register transfer micro-operation.
2. Fetch cycle is primarily used for fetching instruction that is to be executed from the memory. The present-day machines may use instruction cache, instruction prefetch buffer etc., yet this instruction is needed to be moved to the processor, which may execute it. Thus, the nature of the fetch cycle may change but it may be required.
3. Interrupt cycle is responsible for acknowledging an interrupt. When an interrupt occurs in a computer, it is acknowledged, when the instruction execution gets completed. The processor checks, if an interrupt has occurred, if it is then an interrupt cycle is performed.

Check Your Progress 4

1. The implementation of ALU can be done with the help of combinational and sequential circuits. The combinational circuits perform the computations. The results of these computations is stored in sequential circuits. The internal register bus is used for input and output to ALU. Arithmetic circuit like adders may be used to perform arithmetic micro-operations, logic gates may be used to perform logic micro-operations and shift registers may be used to perform shift operations.
2. Arithmetic processor performs arithmetic computation. These may be support processors to a computer.