UNIT 2 REGISTERS, MICRO-OPERATIONS AND INSTRUCTION EXECUTION

Structure **Page No.**

2.0 INTRODUCTION

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 into two simpler questions. These are:

What are the steps required for the execution of an instruction? How are these steps performed by the CPU?

The answer to the first question lies in the fact that each instruction execution consists of several steps. Together they constitute an instruction cycle. A micro-operation is the smallest operation performed by the CPU. These operations put together execute an instruction.

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

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 and their implementation. The discussion on micro-operations will gradually lead us towards the discussion of a very simple ALU structure. The detail of ALU structure is the topic of the next unit.

2.1 OBJECTIVES

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

- describe the register organisation of the CPU;

- define what is a micro-operation;
- differentiate among various micro-operations;
- discuss an instruction execution using the micro-operations; and
- define the concepts of instruction pipelining.

2.2 BASIC CPU STRUCTURE

A computer manipulates data according to the instructions of a stored program. **Stored program** means the program and data are stored in the same memory unit. The central processing unit, also referred to as CPU, performs the bulk of the data processing operations. It has three main components:

1. A set of registers for holding binary information.
2. An arithmetic and logic unit (ALU) for performing data manipulation, and
3. A control unit that coordinates and controls the various operations and initiates the appropriate sequence of micro-operations for each task.

Computer instructions are normally stored in consecutive memory locations and are executed in sequence one by one. The control unit allows reading of an instruction from a specific address in memory and executes it with the help of ALU and Register.

Instruction Execution and Registers

The basic process of instruction execution is:

1. Instruction is fetched from memory to the CPU registers (called instruction fetch) under the control unit.
2. It is decoded by the control unit and converted into a set of lower level control signals, which cause the functions specified by that instruction to be executed.
3. After the completion of execution of the current instruction, the next instruction fetched is the next instruction in sequence.

This process is repeated for every instruction except for program control instructions, like branch, jump or exception instructions. In this case the next instruction to be fetched from memory is taken from the part of memory specified by the instruction, rather than being the next instruction in sequence.

But why do we need Registers?

If t_{cpu} is the cycle time of CPU that is the time taken by the CPU to execute a well-defined micro-operation using registers, and t_{mem} is the memory cycle time, that is the speed at which the memory can be accessed by the CPU, then (t_{cpu}/t_{mem}) is in the range of 2 to 10, that is CPU is 2 – 10 times faster than memory. Thus, CPU registers are the fastest temporary storage areas. 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. Each instruction must designate the registers it will address. Thus, a machine requires a large number of registers.

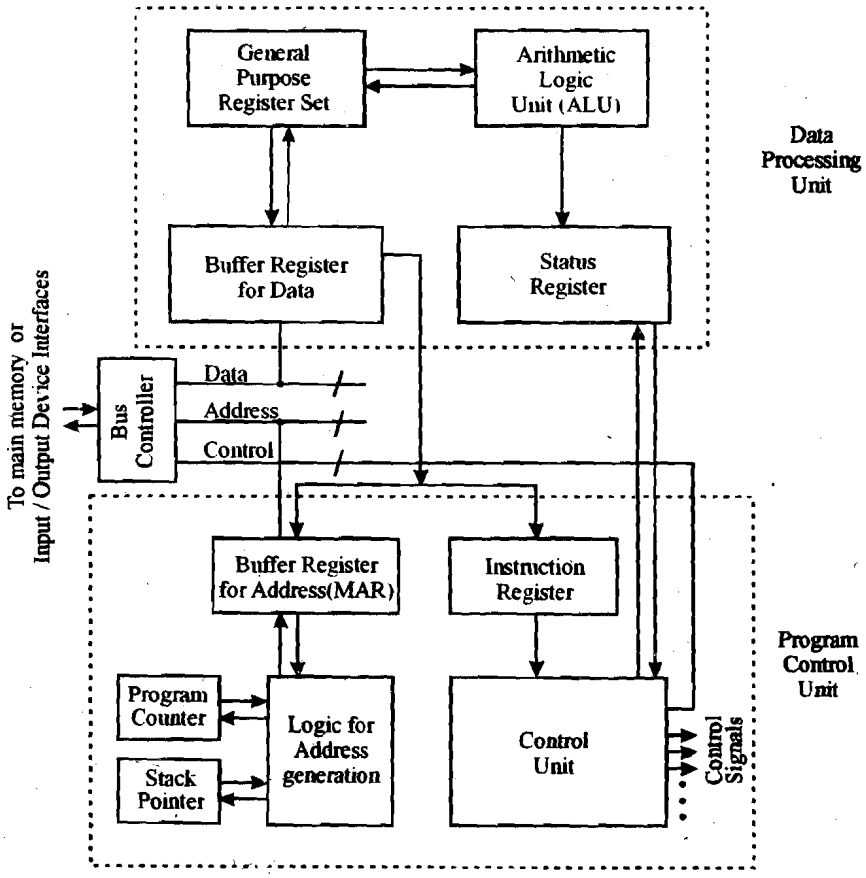

Figure 1: CPU with general register organisation

But how do the registers help in instruction execution? We will discuss this with the help of Figure 1.

Step 1:

The first step of instruction execution is to fetch the instruction that is to be executed. To do so we require:

- Address of the "instruction to be fetched". Normally Program counter (PC) register stores this information.
- Now this address is converted to physical machine address and put on address bus with the help of a buffer register sometimes called Memory Address Register (MAR).
- This, coupled with a request from control unit for reading, fetches the instruction on the data bus, and transfers the instruction to Instruction Register (IR).
- On completion of fetch PC is incremented to point to the next instruction.

In Step 2:

- The IR is decoded; let us assume that Instruction Register contains an instruction. ADD Memory location B with general purpose register R1 and store result in R1, then control unit will first instruct to:
    - Get the data of memory location B to buffer register for data (DR) using buffer address register (MAR) by issuing Memory read operation.
    - This data may be stored in a general purpose register, if so needed let us say R2

- Now, ALU will perform addition of R1 & R2 under the command of control unit and the result will be put back in R1. The status of ALU operation for example result in zero/non zero, overflow/no overflow etc. is recorded in the status register.

- Similarly, the other instructions are fetched and executed using ALU and register under the control of the Control Unit.

Thus, for describing instruction execution, we must describe the registers layout, micro-operations, ALU design and finally the control unit organization. We will discuss registers and micro- operation in this unit. ALU and Control Unit are described in Unit 3 and Unit 4 of this Block.

2.3 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 next instruction to be executed.
- 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. With these three kinds of registers, a computer would be able to execute programs.
- Other types of registers:
    - Memory-address register (MAR) holds the address of next memory operation (load or store).
    - Memory-buffer register (MBR) 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).

A few factors to consider when choosing the number of registers in a CPU are:

- CPU can access registers faster then it can access main memory.
- For addressing a register, depending on the number of addressable registers a few bit addresses is needed in an instruction. These address bits are definetly quite less in comparison to a memory address. For example, for addressing 256 registers you just need 8 bits, whereas, the common memory size of 1MB requires 20 address bits, a difference of 60%.
- Compilers tend to use a small number of registers because large numbers of registers are very 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 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 cannot be used by the programmers but are used to control the CPU or the execution of a program.

Different vendors have used 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 machine. Therefore, let us discuss more about these categories.

2.3.1 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.

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, in order to simplify the task of programmers and computers dedicated registers can be used. For example, registers may be dedicated to floating point operations. One such common dedication may be the data and address registers.

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

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, it has been found that 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 have to 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 later in this 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. In certain cases two consecutive registers may be used to hold data whose length is double of the register length.

2.3.2 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. One of the control registers for a von-Neumann machine is the Program Counter (PC).

Almost all the CPUs, as discussed earlier, 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 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.

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 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, stack pointer if a stack is used to implement subroutine calls, 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?

.....

.....

.....

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

.....

.....

.....

Let us now look into an example register set of MIPS processor.

2.4 GENERAL REGISTERS IN A PROCESSOR

In Block 4 Unit 1, you would be exposed to 8086 registers. In this section we will provide very brief details of registers of a RISC system called MIPS.

MIPS is a register-to-register or load/store architecture and uses three address instructions for data manipulation. It is because of register-register operands that you can have more operands in an instruction of 32 bits, as register address are smaller. The MIPS have 32 addressable registers = $2^5 \Rightarrow$ 5 bits register address. The table given below displays the MIPS general purpose registers.

MIPS register names begin with a $. There are two naming conventions:

- By number:

  $0 $1 $2 ... $31

- By (mostly) two-letter names, such as:

  $a0 - $a3 $t0 - $t7 $s0 - $s7 $gp $fp $sp $ra

Not all of these are general-purpose registers. The following table describes how each general register is treated, and the actions you can take with each register.

Name	Register number	Description	Specify in Expression
ZERO	0	Always has the value 0.	$zero
AT	1	Reserved for the assembler to handle large constants.	$at
V0 - V1	2-3	Function value registers. Values for results and expression evaluation.	$v0 - $v1
A0 - A3	4-7	Argument registers.	$a0 - $a3

T0 - T7	8-15	Temporary registers	$t0 - $t7
S0 - S7	16-23	Saved registers	$s0 - $s7
T8 - T9	24-25	Temporary registers	$t8 - $t9
K0 - K1	26-27	Reserved for the operating system	$k1 - $k2
GP	28	Global pointer register	$gp
SP	29	Stack pointer register	$sp
FP	30	Frame pointer register	$fp
RA	31	Return address register	$ra

You will also study another 8086 based register organization in Block 4 of this course. So, all the computers have a number of registers. But, how exactly is the instruction execution related to registers? To explore this concept, let us first discuss the concept of Micro-operations.

2.5 MICRO-OPERATION CONCEPTS

We have discussed the general architecture and register set of MIPS microprocessor. Our next task is to look at the functionality of ALU, the control unit and how an instruction is executed. In this section, we will define a micro-operation concept, which is the key concept to describe instruction execution.

A micro-operation is an elementary operation performed normally during one clock pulse. On the information stored in one or more registers. The result of the operation may replace the previous content of a register or is transferred to a new register or a memory location.

A digital system performs a sequence of micro-operations on data stored in registers or memory. The specific sequence of micro-operations performed is predetermined for an instruction. Thus, an instruction is a binary code specifying a definite sequence of micro-operations to perform a specific function.

For example, a C program instruction sum = sum + 7, will first be converted to equivalent assembly program:

- Move data from memory location "sum" to register R1 (LOAD R1, sum)
- Add an immediate operand to register (R1) and store the results in R1 (ADD R1, 7)
- Store data from register R1 to memory location "sum" (STORE sum, R1).

Thus, several machine instructions may be needed (this will vary from machine to machine) to execute a simple C statement. But, how will each of these machine statements be executed with the help of micro-operations? Let us try to elaborate the execution steps:

- Fetch the instructions.
    - Pass the address of Program Counter (PC) to Memory Address Register (MAR).
    - Issue the memory read operation to fetch instruction in the Buffer Register for data, such as M(BR).

- Increment Program Counter to refer to next instruction in sequence and bring instruction to Instruction Register (IR).

- Execute the instruction
  - Decode the instruction to ascertain operation.
  - As one of the operands is already available in R1 register and the second operand is an immediate operand so fetch operand step is not required. The immediate operand is available in the address part of the instruction.
  - Perform the ALU based addition with R1 and buffer register, store the result in R1.

Thus, we may have to execute the instruction in several steps. For the subsequent discussion, for simplicity, let us assume that each micro-operation can be completed in one clock period, although some micro-operations require memory read/write that may take more time.

Let us first discuss the type of micro-operations. The most common micro-operations performed in a digital computer can be classified into four categories:

1) Register transfer micro-operations: simply transfer binary information from one register to another.
2) Arithmetic micro-operations: perform simple arithmetic operations on numeric data stored in registers.
3) Logic micro-operations: perform bit manipulation (logic) operations on non-numeric data stored in registers.
4) Shift micro-operations registers: perform shift operations on data stored in registers.

2.5.1 Register Transfer Micro-operations

These micro-operations, as the name suggests transfer information from one register to another. The information does not change during these micro-operations. A register transfer micro-operation may be designed as: R1← R2. The ← symbol implies that the contents of register R2 are transferred to register R1. R2 here is a source register while R1 is a destination register. We will use this notation throughout this section. Please note the following important points about register transfer micro-operations.

- 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 predetermined control condition. We will discuss more about the control unit in Unit 4 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.

The convention used to represent the micro-operations is:

1. Computer register names are designated by capital letters (sometimes followed by numerals) to denote its function. For example, R1, R2 (General Purpose Registers), AR (Address Register), IR (Instruction Register) etc.

2. The individual bits within a register are numbered from 0 (rightmost bit) to n-1 (leftmost bit) as shown in Figure 2b). Common ways of drawing the block diagram of a computer register are shown below. The name of the 16-bit register is IR (Instruction Register) which is partitioned into two subfields in Figure 2d). Bits 0 through 7 are assigned the symbol L (for Low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The symbol IR (L) refers to the low-order byte and IR (H) refers to high-order byte.

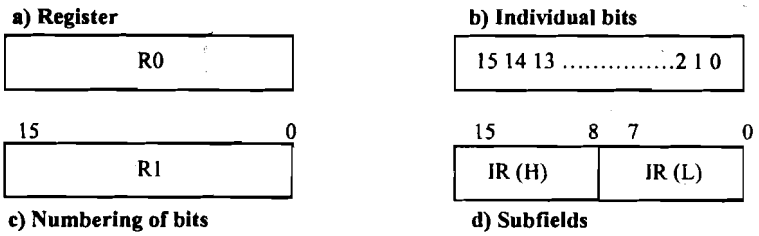

Figure 2: Register Formats

3. Information transfer from one register to another is designated in symbolic notation by a replacement operator. For example, the statement R2 ← R1 denotes a transfer of all bits from the source register R1 to the destination register R2 during one clock pulse and the destination register has a parallel load capacity. However, the contents of register R1 remain unchanged after the register transfer micro-operation. More than one transfer can be shown using a comma operator.

4. If the transfer is to occur only under a predetermined control condition, then this condition can be specified as a control function. For example, if P is a control function then P is a Boolean variable that can have a value of 0 or 1. It is terminated by a colon (:) and placed in front of the actual transfer statement. The operation specified in the statement takes place only when P = 1. Consider the statements:

   If (P =1) then (R2 ← R1)
   or,
   P: R2 ← R1,

   Where P is a control function that can be either 0 or 1.

5. All micro-operations written on a single line are to be executed at the same time provided the statements or a group of statements to be implemented together are free of conflict. A conflict occurs if two different contents are being transferred to a single register at the same time. For example, the statement: new line X: R1← R2, R1← R3 represents a conflict because both R2 and R3 are trying to transfer their contents to R1 at the same time.

6. A clock is not included explicitly in any statements discussed above. However, it is assumed that all transfers occur during the clock edge transition immediately following the period when the control function is 1. All statements imply a hardware construction for implementing the micro-operation statement as shown below:

   Implementation of controlled data transfer from R2 to R1 only when T = 1
   T : R1 ← R2

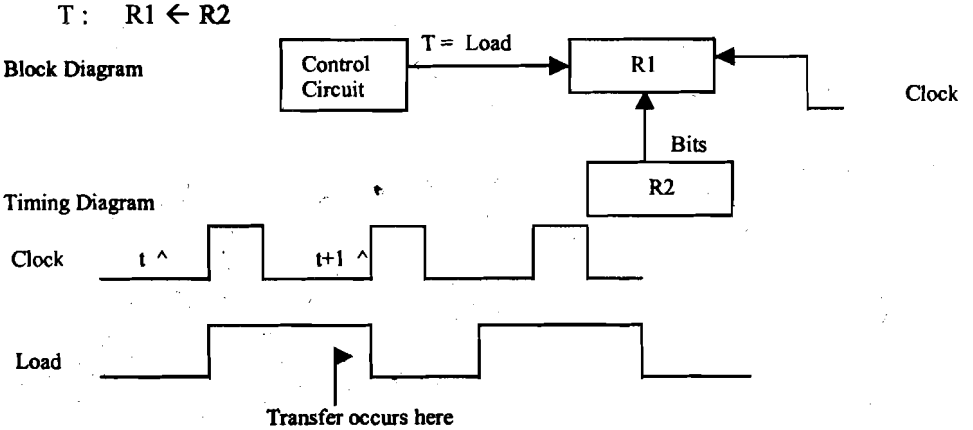

Figure 3: The Register Transfer Time

It is assumed that the control variable is synchronized with the same clock as the one applied to the register. The control function T is activated by the rising edge of the clock pulse at time t. Even though the control variable T becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t+1. At time t+1, load input is again active and the data inputs of R2 are then loaded into the register R1 in parallel. The transfer occurs with every clock pulse transition while T remains active.

Bus and Memory Transfers

A digital computer has many registers, and rather than connecting wires between all registers to transfer information between them, a common bus is used. Bus is a path (consists of a group of wires) one for each bit of a register, over which information is transferred, from any of several sources to any of several destinations.

From a register to Bus: BUS ← R. The implementation of bus is explained in Unit 3 of this block.

The transfer from bus to register can be expressed symbolically as:

R1 ← BUS,

The content of the selected register is placed on the BUS, and the content of the bus is loaded into register R1 by activating its load control input.

Memory Transfer

The transfer of information from memory to outside world i.e., I/O Interface is called a *read* operation. The transfer of new information to be stored in memory is called a *write* operation. These kinds of transfers are achieved via a system bus. It is necessary to supply the address of the memory location for memory transfer operations.

Memory Read

The memory unit receives the address from a register, called the memory address register designated by MAR. The data is transferred to another register, called the data register designated by DR. The read operation can be stated as:

Read: DR ← [MAR]

Memory Write

The memory write operation transfers the content of a data register to a memory word M selected by the address. Assume that the data of register R1 is to be written to the memory at the address provided in MAR. The write operation can be stated as:

Write: [MAR] ← R1

Please note, it means that the location pointed by MAR will be written and not MAR.

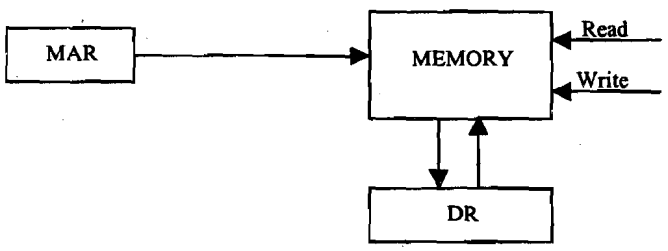

Figure 4: Memory Transfer

2.5.2 Arithmetic Micro-operations

These micro-operations perform simple arithmetic operations on numeric data **stored in registers**. The basic arithmetic micro-operations are addition, subtraction, increment, decrement, and shift.

Addition micro-operation is specified as:

$$R3 \leftarrow R1 + R2$$

It means that the contents of register R1 are added to the contents of register R2 and the sum is transferred to register R3. This operation requires three registers to hold data along with the Binary Adder circuit in the ALU. Binary adder is a digital circuit that generates the arithmetic sum of two binary numbers of any lengths and is constructed with full-adder circuits connected in cascade. An n-bit binary adder requires n full-adders. Add micro-operation, in accumulator machine, can be performed as:

$$AC \leftarrow AC + DR$$

Subtraction is most often implemented in machines through complement and adds operations. It is specified as:

$R3 \leftarrow R1 - R2$
$R3 \leftarrow R1 + (2\text{'s complement of } R2)$
$R3 \leftarrow R1 + (1\text{'s complement of } R2 + 1)$
$R3 \leftarrow R1 + \overline{R2} + 1$ (The bar on top of R2 implies 1's complement of R2 which is bitwise complement)

Adding 1 to the 1's complement produces the 2's complement. Adding the contents of R1 to the 2's complement of R2 is equivalent to subtracting the contents of R2 from R1 and storing the result in R3. We will describe the basic circuit required for these micro-operations in the next unit.

The increment micro-operation adds one to a number in a register. This operation is designated as:

$$R1 \leftarrow R1 + 1$$

This can be implemented in hardware by using a binary-up counter.

The decrement micro-operation subtracts one from a number in a register. This operation is designated as:

$$R1 \leftarrow R1 - 1$$

This can be implemented using binary-down counter.

What about the multiply and division operations? Are not they micro-operations? In most of the older computers multiply and divisions were 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.

2.5.3 Logic Micro-operations

Logic operations are basically 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 micro-operation:

R1 ← 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, and NAND. In many computers only four: AND, OR, XOR (exclusive OR) and complement micro-operations are implemented.

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 R1 and R2 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 R1, and 1100 data for register R2. Why? Because if you see the bit combinations of R2, and R1, they represent the truth table entries (read from right to left and bottom to top) 00, 01, 10 and 11. Thus, the resultant of the logical operation on them will indicate which logic micro-operation is needed to be performed for that data manipulation. The following table gives details on some of these operations:

R1	1	0	1	0
R2	1	1	0	0

Operation name	What is the operation?	Example and Explanation
Selective Set	Sets those bits in Register R1 for which the corresponding R2 bit is 1.	R1 = 1010 R2 = 1100 1110 The value 1110 suggests that selective set can be done using logic OR micro-operation. Please note that all those bits of R1, for which we have 0 bit in R2, have remained unchanged. The bits in R1 which need to be set selectively must have the corresponding R2 bits as 1.
Selective Clear	Clear those bits in register R1 for which corresponding R2 bits are 1.	R1 = 1010 R2 = 1100 0010 The R1 value after the operation is 0010 which suggests that Corresponding micro-operation is R1 AND $\overline{R2}$
Selective Complement	Complement those bits in register R1 for which the corresponding R2 bits are 1.	R1 = 1010 R2 = 1100 0110 The R1, value 0110 after the operation suggests that the selective complement can be done using exclusive - OR micro-operation. The bits in R1 which need to be complemented selectively must have the corresponding R2 bits as 1.
Mask Operations	Clears those bits in Register R1 for which the corresponding R2	R1 = 1010 R2 = 1100 1000 The R1 value after the operation is 1000

	bits are 0.	which suggests that the mask operation can be performed using AND micro-operation. However, the bits in R1 which are cleared or masked correspond to the bits on R2 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-operation with the bits which are to be inserted.	This is a two-step process. *Example:* Say contents of R1 = 0011 1011 Suppose, we want to insert 0110 in place of left most 0011 then: 0011 1011 (R1 before) 0000 1111 (R2 for masking) ——— Perform AND operation (mask) 0000 1011 (R1 after) Now insert: 01100000 (R2 for insertion) ——— Perform OR operation 0110 1011 R1 after insert
Clear	Clear all the bits	R1 = 1101 R2 = 1101 0000 Implemented by taking exclusive OR with the same number. 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 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.

Let us summarize micro-operations using the following table:

Sl. No.	Micro-operations	Examples
1.	Register transfer	R1 ← R2 (register transfer) [MAR] ← R1 (Register to memory)
2.	Arithmetic micro-operations	ADD R1 ← R1 + R2 SUBTRACT R1 ← R1 + ($\overline{R2}$ +1) INCREMENT R1 ← R1 +1 DECREMENT R1← R1 – 1
3.	Logical micro operations	AND OR COMPLEMENT XOR
4.	Shift	Left or right shift • Logical • Arithmetic • Circular

Check Your Progress 2

1. How does the memory read / operation carried out using system bus?

   .....

   .....

   .....

3. Are multiplication and division arithmetic operations micro-operations?

   .....

   .....

   .....

3. What will be the value for R2 operand if:

   (i) Mask operation clears register R1
   (ii) Bits 1011 0001 is to be inserted in an 8 bit R1 register.

4. What are the differences between circular and logical shift micro-operations?

   .....

   .....

   .....

2.6 INSTRUCTION EXECUTIONS AND MICRO - OPERATIONS

Let us now discuss instruction execution using the micro-operations. A simple instruction may require:

- Instruction fetch: fetching the instruction from the memory.
- Instruction decode: decode the instruction.
- Operand address calculation: find out the effective address of the operands.
- Execution: execute the instruction.
- Interrupt Acknowledge: perform an interrupt acknowledge cycle if an interrupt request is pending.

Let us explain how these steps of instruction execution can be broken down to micro-operations. For simplifying the discussion, let us assume that the machine has the structure as shown in Figure 1. In addition, let us also assume that the instruction set of the machine has only two addressing modes direct and indirect memory addresses and a memory access take same time as that of a register access that is one clock cycle.

Instruction fetch: In this phase the instruction is brought from the address pointed by PC to instruction register. The steps required are:

Transfer the address of PC to MAR. (Register Transfer)	MAR ← PC
MAR puts its contents on the address bus for main memory location selection, the control unit instructs the MAR to do so and also uses a memory read signal. The word so read is placed on the data bus where it is accepted by the Data register (Memory-read using bus. It may take more than one clock pulses depending on the t_{cpu} and t_{mem}) The PC is incremented by one memory word length to point to the next instruction in sequence. This micro-operation can be carried out in parallel to the micro-operation above.	DR ← (MAR), PC← PC +1
The instruction so obtained is transferred from data register to the Instruction register for further processing. (Register Transfer)	IR ← DR

Instruction Decode: This phase is performed under the control of the Control Unit of the computer. The Control Unit determines the operation that is to be performed and the addressing mode of the data. In our example, the addressing modes can be direct or indirect.

Operand Address Calculation: In actual machines the effective address may be a memory address, register or I/O port address. The register reference instructions such as complement R1, clear R2 etc. normally do not 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 is calculated. The calculation of effective address may require more memory fetches (for example in the case of indirect addressing), thus in this step we may calculate the effective address as:

For Direct Address: • Transfer the address portion of instruction is the direct address so no further calculation is needed.	IR (Address) and DR (Address) contain the Effective address.
For Indirect Address: • Transfer the address bits of instruction to the MAR. This transfer can be achieved using DR, as DR and IR at this point of time contain the same value. (Register Transfer) • Perform a memory read operation as done in fetch cycle and the desired address of the operand is obtained in the DR. (Memory Read) • Transfer the address part so obtained in DR as the address part of instruction. (Register Transfer) Thus, the indirect address is now converted to direct address or effective address.	MAR ← DR (Address) DR← (MAR) IR (Address) ← DR (Address)

Thus, the address portion of IR now contains the effective address, which is the direct address of the operand.

Execution: Now the instruction is ready for execution. A different opcode will require different sequence of steps for the execution. Therefore, let us discuss a few examples of execution of some simple instructions for the purpose of identifying some of the steps needed during instruction execution. Let us start the discussions with a simple case of addition instruction. Suppose, we have an instruction: Add R1, A which adds the content of memory location A to R1 register storing the result in R1. This instruction will be executed in the following steps:

Transfer the address portion of the instruction to the MAR. (Register transfer)	MAR ← IR (Address)
Read the memory location A and bring the operand in the DR. (Memory read)	DR ← (MAR)
Add the DR with R1 using ALU and bring the results back to R1. (Add micro-operations)	R1 ← R1 + DR

Now, let us try a complex instruction - a conditional jump instruction. Suppose an instruction:

INCSKIP A

increments A and skips the next instruction if the content of A has become zero. This is a complex instruction and requires intermediate decision-making. The micro operations required for this instruction execution are:

Transfer the address portion of IR to the MAR. (Register transfer)	MAR ← IR (Address)
Read memory. DR on reading will contain the operand A. (Memory read)	DR ← (MAR)
Transfer the contents of DR to R1. We are assuming that DR, although it can be used in computation, it cannot be used as destination of an ALU operation. Thus, we need to transfer its content to a general purpose register R1 where the operation can be performed. (Register transfer)	R1 ← DR
Increment the R1. (Increment micro-operation)	R1 ← R1 +1
Transfer the content of R1 to DR. (Register transfer)	DR ← R1
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. (Memory write)	(MAR) ←DR
If the content of R1 is zero then increment PC by one, thus skipping the next instruction. 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. (Increment on a condition)	If R1 = 0 then PC ← PC + 1

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 subroutine call (CALL A) can be:

Transfer the contents of address portion of IR to MAR. (Register Transfer) Transfer the **return address**, that is, the contents of PC to DR. This micro-operation can be performed in parallel to the previous micro-operation. (Register transfer)	MAR ← IR (Address), DR ← PC
Transfer the **branch address** that is stored in Address part of the instruction to program counter. (Register transfer)	PC ← IR (Address)
Store the DR using MAR. Thus, the return address is stored at the first location of the subroutine. (This operation normally is done in stack, but in this example we are storing the return address in the first location of the subroutine). This micro-operation can be performed in parallel to previous micro-operation. (Memory write)	(MAR) ←DR
Increment the PC as it contains the first location of subroutine, which is used to store the return address. The first instruction of subroutine starts from the next location. (Increment)	PC ← PC + 1

Thus, the number of steps required in execution may differ from instruction to instruction.

Interrupt Processing: On completion of the execution of an instruction, the machine checks whether there is any pending interrupt request for the interrupts that are enabled. If an enabled interrupt has occurred then that Interrupt may be processed. The nature of interrupt varies from machine to machine. However, let us discuss one simple illustration of interrupt processing events. A simple sequence of steps followed in interrupt phase is:

Transfer the contents of PC to DR, as this is the return address after the interrupt service program has been executed. This address must be saved.	DR ← PC
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. (Memory write) Transfer the address of the first instruction of interrupt servicing routine to the PC. This micro-operation can be performed in parallel to the above micro-operation.	(MAR) ←DR PC ← address of the first instruction interrupt service programs

After completing the above interrupt processing, CPU will fetch the next instruction that may be interrupt service program instruction. Thus, during this time CPU might be doing the interrupt processing or executing the user program. Please note each instruction of interrupt service program is executed as an instruction in an instruction cycle.

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.

2.7 INSTRUCTION PIPELINING

After discussing instruction execution, let us now define a concept that is very popular in any CPU implementation. This concept is instruction pipeline.

To extract better performance, as defined earlier, instruction execution can be done through instruction pipeline. The instruction pipelining involves decomposing of an instruction execution to a number of pipeline stages. Some of the common pipeline stages can be instruction fetch (IF), instruction decode (ID), operand fetch (OF), execute (EX), store results (SR). An instruction pipe may involve any combination of such stages. A major design decision here is that the instruction stages should be of equal execution time. Why?

A pipeline allows overlapped execution of instructions. Thus, during the course of execution of an instruction the following may be a scenario of execution.

Time Slot ->	1	2	3	4	5	6	7	8	9	10	11
Instruction 1	IF	ID	OF	EX	SR						
Instruction 2		IF	ID	OF	EX	SR					
Instruction 3			IF	ID	OF	EX	SR				
Instruction 4				IF	ID	OF	EX	SR			
Instruction 5					IF	ID	OF	EX	SR		
Instruction 6						IF	ID	OF	EX	SR	
Instruction 7							IF	ID	OF	EX	SR

Figure 5: Instruction Pipeline

Please note the following observations about the above figure:

- The pipeline stages are like steps. Thus, a step of the pipeline is to be complete in a time slot. The size of the time slot will be governed by the stage taking maximum time. Thus, if the time taken in various stages is almost similar, we get the best results.
- The first instruction execution is completed on completion of 5th time slot, but afterwards, in each time slot the next instruction gets executed. So, in ideal conditions one instruction is executed in the pipeline in each time slot.
- Please note that after the 5th time slot and afterwards the pipe is full. In the 5th time slot the stages of execution of five instructions are:

SR	(instruction 1)	(Requires memory reference)
EX	(instruction 2)	(No memory reference)
OF	(instruction 3)	(Requires memory reference)
ID	(instruction 4)	(No memory reference)
IF	(instruction 5)	(Requires memory reference)

The Pipelining Problems:

- On the 5th time slot and later, there may be a register or memory conflict in the instructions that are performing memory and register references that is various stages may refer to same registers/memory location. This will result in slower execution instruction pipeline that is one of the higher number instruction has to wait till the lower number instructions completed, effectively pushing the whole pipelining by one time slot.

- Another important situation in Instruction Pipeline may be the branch instruction. Suppose that instruction 2 is a conditional branch instruction, then by the time the decision to take the branch is taken (at time interval 5) three more instructions have already been fetched. Thus, if the branch is to be taken then the whole pipeline is to be emptied first. Thus, in such cases, pipeline cannot run at full load.

How can we minimize the problems occurring due to the branch instructions?

We can use many mechanisms that may minimize the effect of branch penalty.

- To keep multiple streams in pipeline in case of branch
- Pre-fetching the next as well as instruction to which branch is to take place
- A loop buffer may be used to store the instructions of a loop instruction
- Predicting whether the branch will take place or not and acting accordingly
- Delaying the pipeline fill up till the branch decision is taken.

Check Your Progress 3

State True or False T F

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

2) Register transfer micro-operations are not needed for instruction execution. ☐

3) 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. ☐

2.8 SUMMARY

In this unit, we have discussed in detail the register organisation and a simple structure of the CPU. After this we have discussed in details the micro-operations and their implementation in hardware using simple logical circuits. While discussing micro-operations our main emphasis was on simple arithmetic, logic and shift micro-operations, in addition to register transfer and memory transfer. The knowledge you have acquired about register sets and conditional codes, helps us in giving us an idea that conditional micro-operations can be implemented by simply checking flags and conditional codes. This idea will be clearer after we go through 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 have also defined the concepts of Instruction Pipeline. We will be using this approach for discussing control unit details in Unit 3 and Unit 4. The following table gives the details of various terms used in this unit.

General purpose registers	These registers are used for any address or data computation / storage
Status and control register	Stores the various condition codes
Programmer visible registers	Used by programmers during programming
Micro-operations	Involves register transfer micro operations arithmetic micro-operations like add, subtract, logic micro-operations like AND, OR, NOT, XOR and shift micro-operations left or right shift
Micro-operations and instruction execution	An instruction is executed through a sequence of micro-operations. Thus, a program is executed as a sequence of instruction is executed when a sequence of microinstructions are executed.
Instruction pipeline	Allows overlapped execution of instructions. A good pipe can produce one instruction per clock cycle.

You will also get the details on 8086 microprocessor register sets, conditional codes, instructions etc. in Unit 1 of Block 4.

You can refer to further readings for more register organisation examples and for more details on micro-operations and instruction execution.

2.9 SOLUTIONS /ANSWERS

Check Your Progress 1

1. Registers, which are 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 2

1. Read operation involves reading of location pointed to by MAR. The address bus is loaded with the contents of MAR
   address BUS ←MAR
   In addition a read signal is issued by control unit, and data is stored to MBR register or data register.
   DR ← data BUS
   The combined operation can be shown as
   DR ← [MAR]

2. Yes, if implemented through circuits.
   No, if implemented through algorithms involving add/ subtract and shift micro-operations.

3. (i) 0000 0000
   (ii) Initially AND with 0000 0000 followed by OR with 1011 0001

4. The bits circulate and after a complete cycle the data is still intact in circular shift. Not so in logical shift.

Check Your Progress 3

1. True
2. False
3. True