UNIT – I
REGISTER TRANSFER LANGUAGE AND MICRO OPERATIONS

Objective:

- To familiarize with organizational aspects of memory, processor and I/O.

Syllabus:

Introduction- Functional units, Computer Registres, Register Transfer language, Register Transfer Bus and memory transfers, Arithmetic, logic and shift micro-operations, Arithmetic logic shift unit.
Basic Computer Organization and Design: Instruction codes, Instruction cycle, register reference instructions, Memory – reference instructions, Input – Output and Interrupt.

Learning Outcomes:

At the end of the unit student will be able to:
1. Understand different types of instructions.
2. Describe about instruction cycle.

Learning Material

Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.

Compute design is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system. Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as **computer implementation**.

Computer architecture is concerned with the structure and behaviour of the computer as seen by the user.

1.1 FUNCTIONAL UNIT

- A computer in its simplest form comprises of five functional units namely input unit, output unit, memory unit, arithmetic & logic unit and control unit. Below figure depicts the functional units of a computer system.

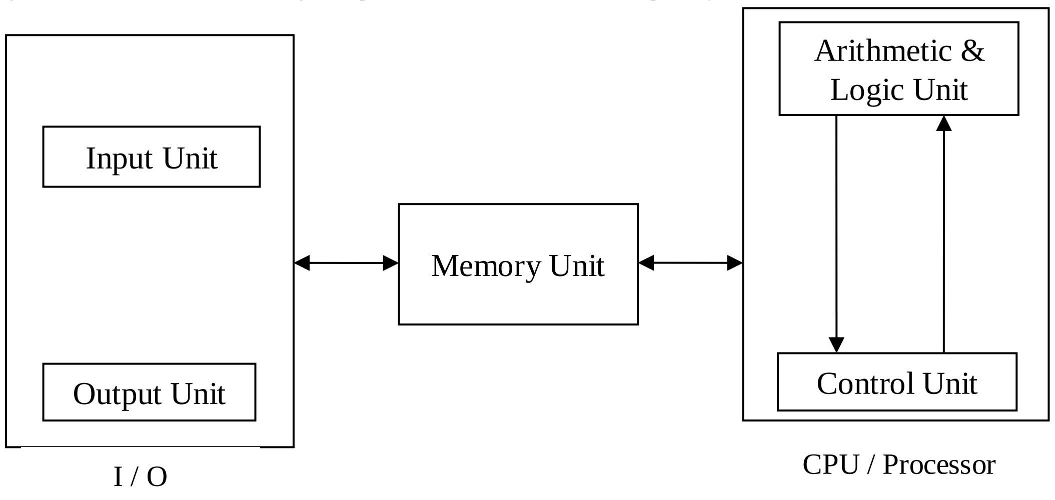


Figure 1: Basic functional units of a computer

1. ***Input Unit:*** Computer accepts encoded information through input unit. The standard input device is a keyboard. Whenever a key is pressed, keyboard controller sends the code to CPU/Memory.
Examples include Mouse, Joystick, Tracker ball, Light pen, Digitizer, Scanner etc.
*2. **Output Unit:*** Computer after computation returns the computed results, error messages, etc. via output unit. The standard output device is a video monitor, LCD/TFT monitor. Other output devices are printers, plotters etc.
*3. **Memory Unit:*** Memory unit stores the program instructions (Code), data and results of computations etc.
Memory unit is classified as:
  - Primary /Main Memory
  - Secondary /Auxiliary Memory

- The Memory unit can be referred to as the storage area in which programs are kept which are running, and that contains data needed by the running programs.
- The Memory unit can be categorized in two ways namely, primary memory and secondary memory.
- It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.
- Primary storage is the fastest memory that operates at electronic speeds. Primary memory contains a large number of semiconductor storage cells, capable of storing a bit of information. The word length of a computer is between 16-64 bits.
- It is also known as the volatile form of memory, means when the computer is shut down, anything contained in RAM is lost.
- Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor.
- The most common examples of primary memory are RAM and ROM.
- Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.
- It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.

- The most common examples of secondary memory are magnetic disks, magnetic tapes, and optical disks.

4. ***Arithmetic and logic unit:*** ALU consist of necessary logic circuits like adder, comparator etc., to perform operations of addition, multiplication, comparison of two numbers etc.
5. ***Control Unit:*** Control unit co-ordinates activities of all units by issuing control signals. Control signals issued by control unit direct the data transfers and then appropriate operations take place. Control unit interprets or decides the operation/action to be performed.

1.2 COMPUTER REGISTERS

- Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers. A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).
- The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.
- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.
- For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR. Other designations for registers are PC (for program counter), IR (for instruction register), and R1 (for processor register).
- The individual flip-flops in an n-bit register are numbered in sequence from 0 through n - 1, starting from 0 in the rightmost position and increasing the numbers toward the left.
- The following diagram shows the representation of registers.

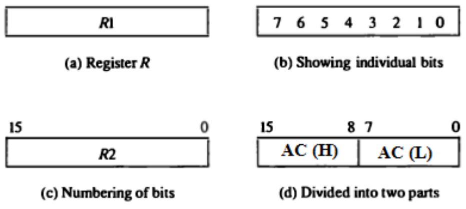


Figure 2: Block diagram of register

Table 1: List of Registers for the Basic Computer

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

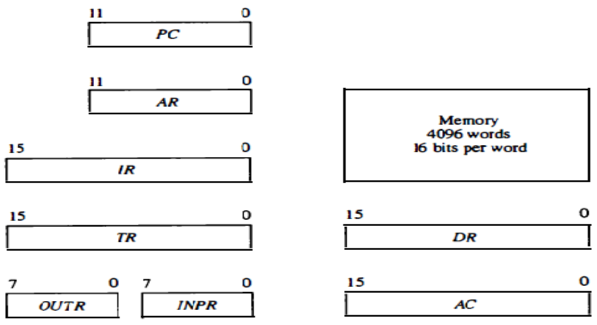


Figure 3: Basic computer registers and memory

- The memory unit has a capacity of 4096 words and each word contains 16 bits.12 bits of an instruction word are needed to specify the address of an operand.
- The data register (DR) holds the operand read from memory.The accumulator (AC) register is general purpose processing register.
- The instruction read from memory is placed in the instruction register(IR). The temporary register (TR) is used as temporary data during the processing.
- The memory address register (AR) has 12 bits since this is the width of a memory address.
- The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed.
- Two registers are used for input and output .the input register (INPR) receives an 8 bit character from an input device. The output register (OUTR) holds an 8 bit character for an output device.

COMMON BUS SYSTEM :-

- The basic computer has 8 registers, a memory unit and a control unit. paths must be provided to transfer information from one register to another and between memory and registers.
- A more efficient scheme for transferring information in a system with many registers is to use a common bus system.
- The outputs of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.
- The number along each output shows the decimal equivalent of the required binary selection. For example, the number along the output of DR is 3.

- The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0$ = 011 since this is the binary value of decimal 3. The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse. The memory receives the contents of the bus when its write pin is enabled.
- The memory places its 16-bit output onto the bus when the read pin is enabled and $S_2S_1S_0$=111.
- Four registers, DR, AC, IR, and TR, have 16 bits each. Two registers, AR and PC, have 12 bits each since they hold a memory address.
- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0's. When AR or PC receives information from the bus, only the 12 least significant bits are transferred into the register.
- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus.
- INPR receives a character from an input device to provide information onto the bus which in turn is then transferred to AC.
- OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers.
- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear). This type of register is equivalent to a binary counter with parallel load and synchronous clear.

- The increment operation is achieved by enabling the count input of the counter .Two registers have only a LD input.
- The input data and output data of the memory are connected to the common bus but the memory address is connected to AR.
- The 16 inputs of AC come from an adder and logic circuit. This circuit has three sets of inputs. One set of 16-bit inputs come from the outputs of AC . They are used to implement register microoperations such as complement AC and shift AC .

- Another set of 16-bit inputs come from the data register DR. The inputs from DR and AC are used for arithmetic and logic rnlcrooperations, such as add DR to AC or AND DR to AC.

- The result of an addition is transferred to AC and the end carry-out of the addition is transferred to flip-flop E (extended AC bit). A third set of 8-bit inputs come from the input register INPR.

- A third set of 8 bit inputs come from the input register INPR.
- **Note that the content of any register** can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle. The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

- **For example**, the two microoperations

- DR ← AC and AC ← DR

- **can be executed at the same time**. This can be done by placing the content of AC on the bus (with $S_2S_1S_0$ = 100), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC, and enabling the LD (load) input of AC, all during the same clock cycle.

- **The two transfers** occur upon the arrival of the clock pulse transition at the end of the clock cycle.

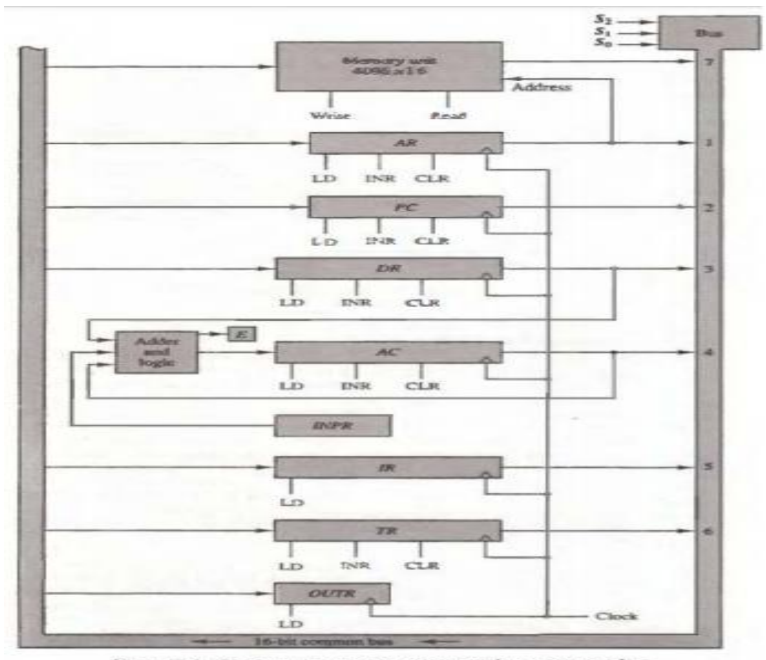


Figure 5-4 Basic computer registers connected to a common bus.

Figure 4: Basic Computer Register Connected to a Common BUS

Micro operation: The operations executed on data stored in registers are called microoperations. A microoperation is an elementary operation performed on information stored in one or more registers. The result of the operation may replace the previous binary information of registers or may be transferred to another register.

Eg: Shift, Count, Clear and Load.

1.3 REGISTER TRANSFER LANGUAGE

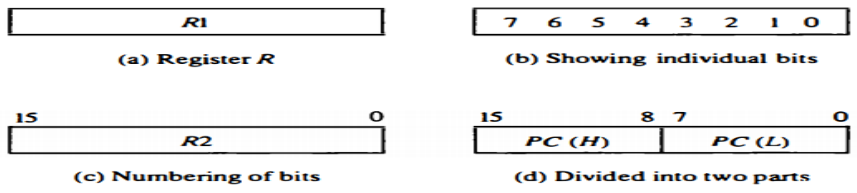
- The symbolic notation used to describe the Micro operation transfers among registers is called a **Register Transfer Language**.

- The Information transferred from one register to another register is designed in symbolic form by means of replacement operator (🡐).

The internal hardware organization of a digital system is best defined by specifying:

- The set of registers and the flow of data between them.
- The sequence of micro-operations performed on the data which are stored in the registers.
- The control paths that initiates the sequence of micro-operation
- An n-bit register are numbered in sequence from 0 through n-1.The most common way to represent a register
- Is rectangular box with the name of the register inside as shown in below figure.
- A 16 –bit register is partitioned into two parts. Bits 0 through 7 are assigned the symbol L(LOWER BYTE) and bits 8 through 15 are assigned the symbol H(HIGHER BYTE). The name of the 16 bit register is PC the symbol PC(0-7) OR PC(L) refers to the lower order byte and PC(8-15)or PC(H) to the higher order byte.

Figure Block diagram of register.



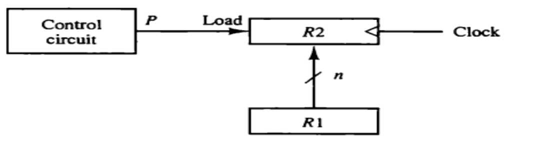
- The statement R2 🡐 R1 denotes a transfer of the contents of register R1 into register R2.
- If we want to transfer only under a predefined condition, this can be shown by means of if-then statement.

  if(p = 1) then R2 🡐 R1

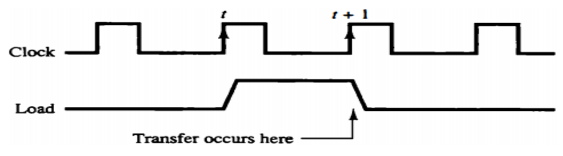
  where p is a control signal generated in the control section.
- A Control function is a Boolean variable that is equal to 0 or 1.
- The control function included in the statement is represented as follows.

  p: R2 🡐 R1
- Here the transfer operation is performed by hardware only if p = 1. The transfer operation is not performed by hardware only if p=0
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.
- The n outputs of register R1 are connected to the n inputs of register R2.The letter n will be used to indicate any number of bits for the register. p: R2 🡐 R1, R1 🡐 R2 denotes an operation that exchanges the contents of two registers during one common clock pulse provided that P=1.

Figure Transfer from R1 to R2 when P = 1.



(a) Block diagram



(b) Timing diagram

The basic symbols of the register transfer notation are listed in the following table.

TABLE Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R*2
Parentheses ( )	Denotes a part of a register	*R*2(0–7), *R*2(*L*)
Arrow ←	Denotes transfer of information	*R*2 ← *R*1
Comma ,	Separates two microoperations	*R*2 ← *R*1, *R*1 ← *R*2

* A comma is used to separate two or more operations that are executed at the same time.

Eg: R2 🡐 R1, R1 🡐 R2

- The above statement denotes an operation that exchanges the content of two registers during one common clock pulse.

1.3.1 Bus Transfer

- Digital computers have many registers, and paths must be provided to transfer information from one register to another register.
- The no. of wires will be excessive if separate lines are used between each registers and all other registers in the system.
- A Bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
- Control signal determines which register is selected by the bus during each particular register transfer.
- The construction of a bus system for 4 registers and multiplexers is shown in the following figure.
- The multiplexers select the source register whose binary information is then placed on the bus.
- Here each register has 4 bits, numbered 0 through 3.
- The bus consists of four 4×1 multiplexers, each having four data inputs, 0 through 3, and two selection inputs S_1 and S_0.
- The selection lines choose the four bits of one register and transfer them into the 4-line common bus.

- The two selection lines S1 and S0 are connected to the selection inputs of all four multiplexers.
- The selection lines choose the four bits of one register and transfer them into the four line common bus.
- When S_1S_0 = 00, then 0 data input of all four multiplexers are selected and applied to output that form the bus.
- **Similarly, register B** is selected if S1S0 = 01, and so on. Table below shows the register that is selected by the bus for each of the four possible binary value of the selection lines.The bus system will multiplex K registers of n bits each to produce an n line common bus.

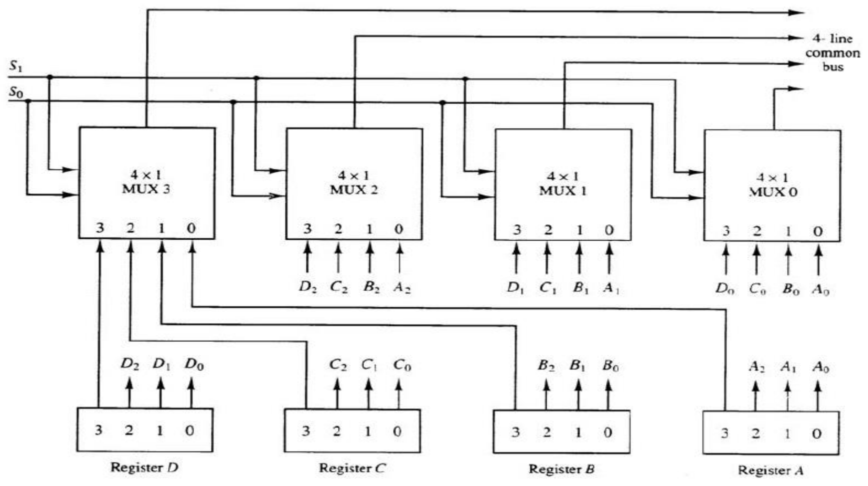


Figure 6: Bus system for four registers

- The following table shows the register that is selected by the bus for each of the four possible binary values of selection lines.

Table 3: Selection of Registers

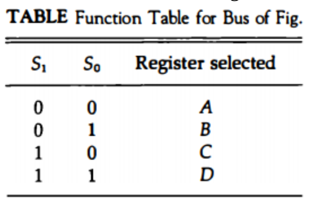


TABLE Function Table for Bus of Fig.

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

- The transfer of information from a bus into one of many registers can be accomplished by connecting the bus lines to inputs of all destination registers and activating the load control of particular destination register.

  Eg: BUS 🡐 C
- In the above statement the content of register C is placed onto BUS

  Eg: R1 🡐 BUS
- In the above statement the content of register C, placed onto BUS is loaded into Register R1by activating the load pin.

1.3.1.1 Three (Tri) state Bus Buffer

- A Bus system can be constructed with three-state gates instead of multiplexers.
- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 or 0 as a conventional gate.
- The third state is a high-impedance state. The high-impedance behaves like an open circuit.
- The graphical symbol of a three state buffer is shown in the following figure.

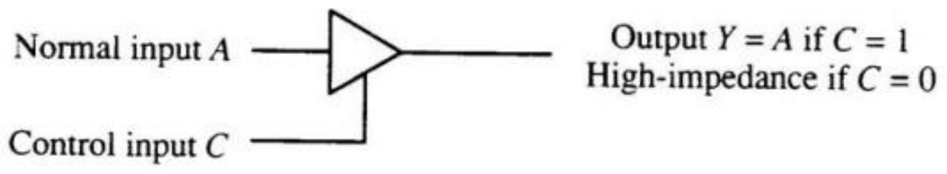


Figure 7: Graphic symbol of a three state buffer

- In the above diagram, control input determines the output state.
- When the control input is equal to 1, the output is enabled like as any conventional buffer.
- When the control input is equal to 0, the output is disabled and the gate goes to a high-impedance state.
- The following figure shows construction of a bus system with Three state Bus Buffer.
- Here the output of 4 buffers are connected together to form a single bus line.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- No more than one buffer may be in the active state at any given time.
- Only one three-state buffer has access to the bus line while all other buffers are maintained in high impedance state.
- With the help of the decoder, three state buffers are controlled.
- When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, i.e. 1, one of the three-state buffers will be active, depending on the select lines of the decoder.
- To construct a common bus for four registers of n bits each using three-state buffers, we need n circuits with four buffers in each circuit.
- Each group of four buffers receives one significant bit from the four registers.
- Each common output produces one of the lines for the common bus for a total of n lines.

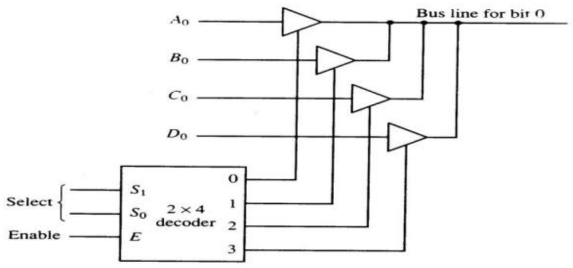


Figure 8: Bus Line with three state buffer

Table 4: Active Three State Buffer with respect to enable input

E	S_1	S_0	Active Three State Buffer
0	X	X	High impedance state
1	0	0	A
1	0	1	B
1	1	0	C
1	1	1	D

1.3.2 Memory Transfer

- The transfer of information from a memory word symbolized by M, to the outside environment is called a ***read*** operation.
- The transfer of new information to be stored into the memory is called a ***write*** operation.
- The particular memory word among the many available is selected by the memory address during the transfer.
- Consider a memory unit that receives the address from a register, called the address register, symbolized by AR.
- The data is transferred to another register, called the data register, symbolized by DR.
- The memory read operation can be stated as follows

  Read: DR $\leftarrow$M [AR]
- Here data is transferred into DR from the memory location specified by AR.
- The memory write operation can be stated as follows

  Write: M [AR]$\leftarrow$ DR
- Here write operation transfers the content of a data register to a memory word M specified by the address in AR.

The micro operations most often encountered in digital computers are classified into four categories:

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

1.4 ARITHMETIC MICRO OPERATIONS

- The basic arithmetic microoperations are addition, subtraction, increment and decrement.

<u>*Add Microoperation:*</u>

$$R3 \leftarrow R1 + R2$$

- The above statement states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.

<u>*Subtract Microoperation:*</u>

$$R3 \leftarrow R1 + \overline{R2} + 1$$

- In the above statement $\overline{R2}$ is the symbol for the 1's complement of R2. 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 R1 – R2.
- The increment and decrement microoperations are symbolized by plus-one and minus-one operations, respectively.

TABLE Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

1.4.1 Binary Adder

- To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the arithmetic addition.
- The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.
- The augend bits (A) and the addend bits (B) are designated by subscript numbers from right to left, with subscript '0' denoting the low-order bit.
- The carry inputs starts from C0 to C3 connected in a chain through the full-adders. C4 is the resultant output carry generated by the last full-adder circuit.
- The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.
- The sum outputs (S0 to S3) generates the required arithmetic sum of augend and addend bits.

- The ***n*** data bits for the **A** and **B** inputs come from different source registers. For instance, data bits for **A** input comes from source register R1 and data bits for **B** input comes from source register R2.
- The arithmetic sum of the data inputs of A and B can be transferred to a third register or to one of the source registers (R1 or R2).

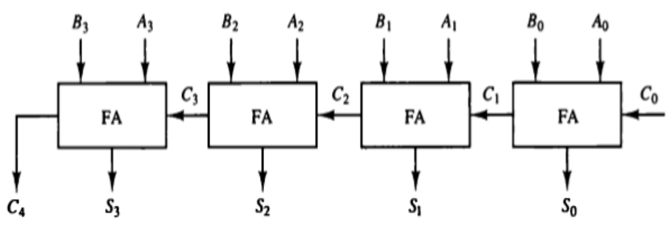

B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
FA FA FA FA
C_3 C_2 C_1 C_0
C_4 S_3 S_2 S_1 S_0

Figure 9: 4-bit binary adder

- The S outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders. The output carry from each full-adder is connected to the input carry of the next-higher-order full-adder.

1.4.2 Binary Adder-Subtractor

- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.
- The subtraction of binary numbers can be completed effectively by creating the 2's complement of addend bits and inserting it to the augend bits. The 2's complement can be acquired by taking the 1's complement and inserting one to the least significant pair of bits.
- A 4-bit adder-subtractor circuit is shown in the following figure.

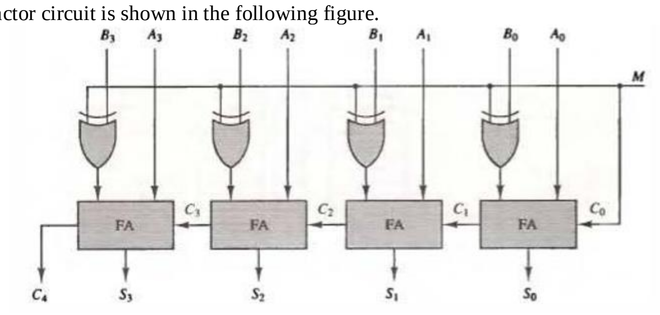

B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
M
FA FA FA FA
C_3 C_2 C_1 C_0
C_4 S_3 S_2 S_1 S_0

Figure 10: Adder-subtractor

- In the above figure, mode input M controls the operation.
- When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor.
- Each exclusive-OR gate receives input M and one of the inputs B. When M = 0, we have $B \oplus 0 = B$. The full-adders receive the value of B, the input carry is 0, and the circuit performs A+B.
- When M = 1, we have $B \oplus 1 = B'$ and $C_0 = 1$. The B inputs are all complemented and 1 is added through the input carry. The circuit performs the operation A+2's complement of B.
- The circuit operates A plus the2's complement of B. For unsigned numbers, this provides A - B if A \geq B or the 2's complement of (B - A) if A < B. For signed numbers, the result is A - B supported that there is no overflow.

1.4.3 Binary lncrementer

- The increment microoperation adds one to a number in the register.
- The diagram of a 4-bit incrementer circuit is shown below. One of the inputs of the least significant half-adder (HA) circuit is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.

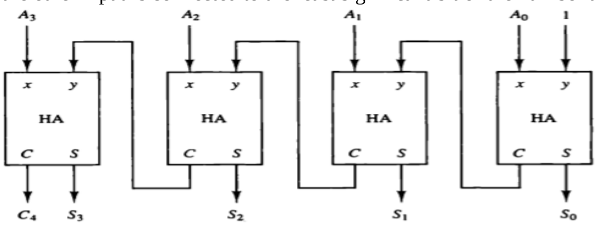
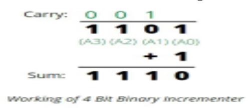


Figure 11: 4-Bit Binary Incrementer

- The half adders are connected one after the other , as it has 2 inputs and 2 outputs , so for the LSB (least significant bit) half adder or the right most half adder is given 1 as direct input(first input) and A0 which is the first bit of the register (second input) , so we get the two output : sum (S0) and carry (C).
- The carry(C) from previous half adder is propagated to the next half adder, so the carry output of the previous half adder becomes the input of the next higher order half adder.
- So considering the case for 4 half adders the circuit gets in total 4 bits (A0, A1, A2, A3), 1 is added and we get an incremented output.

The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder. The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output S_0 through S_3.



1.4.4 Arithmetic Circuit

The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

- The diagram of a 4-bit arithmetic circuit is as shown in the following figure. It has four full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations.
- There are two 4-bit inputs A and B and a 4-bit output D. The four inputs from A go directly to the X inputs of the full adder. Each of the four inputs from B are connected to the data inputs of the multiplexers.
- The multiplexers data inputs also receive the complement of B the other two data inputs are connected to logic 0 and logic 1 is a fixed value.
- The four multiplexers are controlled by two selection inputs, S_1 and S_0.

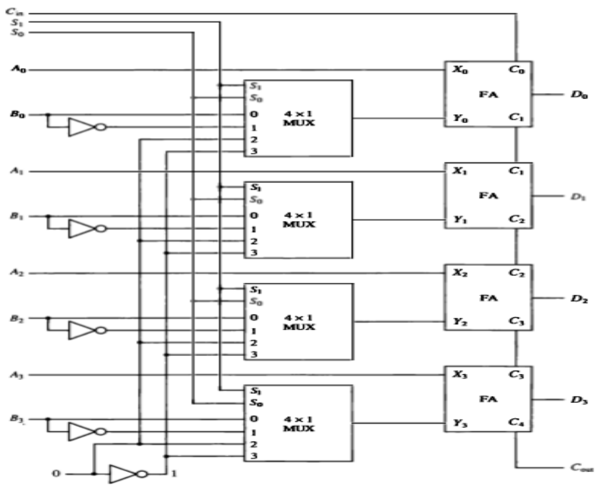


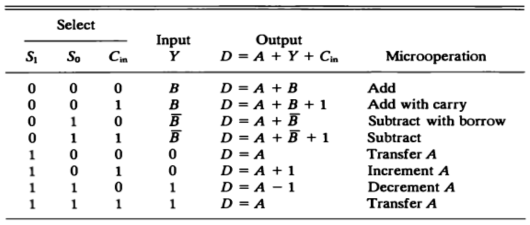
Figure 12: 4-bit arithmetic circuit

- The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- The output of the binary adder is calculated from the following arithmetic sum:

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

- where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. C_{in} is the input carry, which can be equal to 0 or 1.
- By controlling the value of Y with the two selection inputs S_1 and S_0 and making C_{in} equal to 0 or 1, it is possible to generate the eight arithmetic microoperations listed in the following table.

Table 6: Arithmetic Circuit Function Table



Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

- **When S_1S_0 = 00**, the value of B is applied to the Y inputs of the adder. If C_{in} = 0, the output D = A + B . If C_{in} = 1, output D = A + B + l. Both cases perform the add microoperation with or without adding the input carry.

- **When S_1S_0 = 01**, the complement of B is applied to the Y inputs of the adder. If C_{in} = 1, then D = A + B + 1. This produces A plus the 2's complement of B, which is equivalent to a subtraction of A - B.

  When C_{in} = 0, then D = A + B. This is equivalent to a subtract with borrow, that is, A - B - 1. When S_1S_0 = 10, the inputs from B are neglected, and instead, all O's are inserted into the Y inputs. The output becomes D = A + 0 + Cm路 This gives D = A when C_{in} = 0 and D = A + 1 when C_{in} = 1. In the first case we have a direct transfer from input A to output D.

  In the second case, the value of A is incremented by 1. When S_1S_0 = 11, all 1' s are inserted into the Y inputs of the adder to produce the decrement operation D = A - 1 when C_{in} = 0.
  This is because a number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces F = A + 2's complement of 1 = A - 1. When C_{in} = 1, then D = A - 1 + 1 = A, which causes a direct transfer from input A to output D.

1.5 LOGIC MICROOPERATIONS

- Logic micro operations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variables.
- For example, the exclusive-OR microoperation on the contents of two registers R1 and R2 is symbolized by the statement.

$$P: R1 \leftarrow R1 \oplus R2$$

- The above specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1.
- As a numerical example, assume that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100. The exclusive-OR microoperation stated above symbolizes the following logic computation:

1010	Content of R1
$\underline{1100}$	Content of R2
0110	Content of R1 if P = 1

- The symbol **V** will be used to denote an **OR** microoperation and the symbol **Λ** to denote an **AND** microoperation. The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.

- There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible truth tables obtained with two binary variables.

Table 7: Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

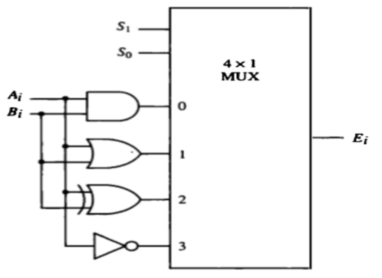
- In the following table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x and y. Note that the functions are determined from the 16 binary combinations that can be assigned to F.

Table 8: Truth Tables for 16 Functions of Two Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

1.5.1 Hardware Implementation for Logic Microoperations

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
- The following figure shows one stage of a circuit that generates the four basic logic microoperations. It consists of four gates and a multiplexer.
- The outputs of the gates are applied to the data inputs of the multiplexer. The two selection inputs S_1 and S_0 choose one of the data inputs of the multiplexer and direct its value to the output.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

Figure 13: One stage of logic circuit

1.5.2 Applications of logic microoperations

1.5.2.1. selective-set

- The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B. The following numerical example clarifies this operation:

1010	A before
<u>1100</u>	B (logic operand)
1110	A after

1.5.2.2. selective-complement

- The selective-complement operation complements bits in A where there are corresponding l's in B. It does not affect bit positions that have 0's in B.
  For example:

1010	A before
<u>1100</u>	B (logic operand)
0110	A after

1.5.2.3. Selective-clear

- The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B.
  For example:

1010	A before
<u>1100</u>	B (logic operand)
0010	A after

1.5.2.4. mask

- The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0'sin B. The mask operation is an AND micro operation as seen from the following numerical example:

1010	A before

1100	B (logic operand)
1000	A after masking

1.5.2.5. Insert

- The insert operation inserts a new value into a group of bits.
- This is done by first masking the bits and then ORing them with the required value.
- For example, an A register contains eight bits, 0110 1010. To replace the four leftmost bits by the value 1001.
  First mask the four unwanted bits.

0110 1010	A before masking
0000 1111	B (mask)
0000 1010	A after masking

Now insert the new value.

0000 1010	A before insert
1001 0000	B (insert)
1001 1010	A after insertion

** The mask operation is an AND microoperation and the insert operation is an OR microoperation.

1.5.2.6. clear

- The clear operation compares the words in A and B and produces an all 0's result, if the two numbers are equal.
- This operation is achieved by an exclusive-OR microoperation as shown by the following example.

1010	A
1010	B
0000	$A \leftarrow A \oplus B$

1.6 SHIFT MICROOPERATIONS

- Shift rnicrooperations are used for serial transfer of data.
- They are also used in conjunction with arithmetic, logic, and other data-processing operations.
- The contents of a register can be shifted to the left or the right.
- There are three types of shifts:
  1. Logical Shift
  2. Circular Shift
  3. Arithmetic Shift

1.6.1. Logical Shift

- A logical shift is one that transfers 0 through the serial input to fill the vacancy created by shift operation.
- The symbols *shl* is used for logical shift-left rnicrooperation.
  Shr is used for shift-right rnicrooperation.

shl: Example: R1 ←shl R1
the above statement left shifts the content of register R1 by 1-bit.
Example: R1 = 0110
After performing shift left, R1 has the content 1100

shr: Example: R1 ←shr R1
the above statement right shifts the content of register R1 by 1-bit.
Example: R1 = 1100
After performing shift left, R1 has the content 0110

1.6.2. Circular Shift

- The circular shift also known as a rotate operation.
- It circulates the bits of the register around the two ends without loss of information.
- The symbols *cil* used for circular shift-left rnicrooperation.
  cir used for circular shift-right rnicrooperation.

cil: Example: R1 ←cil R1
the above statement specifies circular shift left of the content of register R1.
Here all the bits are shifted one bit position to LEFT, the Left most bit (MSB) was circulated to Right most bit (LSB).
Example: R1 = 1011
After performing circular shift left, R1 has the content 0111

cir: Example: R1 ←cir R1
the above statement specifies circular shift right of the content of register R1.
Here all the bits are shifted one bit position to RIGHT, the Right most bit (LSB) was circulated to Left most bit (MSB).
Example: R1 = 1011
After performing circular shift right, R1 has the content 1101

1.6.3. Arithmetic Shift

- An arithmetic shift is a microoperation that shifts a signed binary number to the left or right.
- After the shift microoperation the sign of the number is to be restored.
- The symbols ash*l* used for arithmetic shift-left rnicrooperation.
  ashr used for arithmetic shift-right rnicrooperation

ashl: Example: R1 ←ashl R1
the above statement specifies arithmetic shift left of the content of register R1
Here all the bits except the MSB, shift one bit position to LEFT. The second Left most bit was discarded and Right most bit (LSB) was loaded by 0.
Example: R1 = 1011
After performing arithmetic shift left, R1 has the content 1110

ashr: Example: R1 ←ashr R1
the above statement specifies arithmetic shift right of the content of register R1
Here all the bits shifted one bit position to RIGHT. The Left most bit (MSB) remains same.
Example: R1 = 1011
After performing arithmetic shift right, R1 has the content 1101

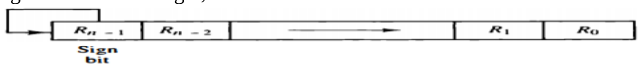


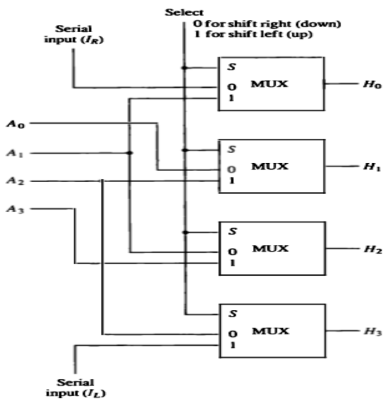
Figure Arithmetic shift right.

TABLE Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

1.6.4 Hardware Implementation for Shift Microoperations

- A combinational circuit shifter can be constructed with multiplexers as shown in figure.



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 14: 4-bit combinational circuit shifter

** Here H_0 bit is considered as MSB and H_3 bit is considered as LSB.

- The 4-bit combinational circuit shifter uses four multiplexers, each of 2X1.
- The 4-bit shifter has four data inputs, A_0 through A_3, and four data outputs H_0 through H_3.
- There are two serial inputs, one for shift left (I_L) and the other for shift right (I_R).
- When the selection input S = 0, the input data are shifted right (down in the diagram).
- When S = 1, the input data are shifted left (up in the diagram).
- The above function table shows which input goes to each output after the shift.
- A shifter with n data inputs and n data outputs requires n multiplexers each of 2 X 1.

1.6.5 Arithmetic Logic Shift Unit

- Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.
- To perform a microoperation, the contents of specified registers are placed in the inputs of the ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register.
- The ALU is a combinational circuit, so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.
- The arithmetic, logic, and shift circuits can be combined into one ALU with common selection variables.
- One stage of an arithmetic logic shift unit is shown in the following figure.
- Inputs A_i and B_i are applied to both the arithmetic and logic units.
- A particular microoperation is selected with inputs S_1 and S_0.
- A 4 x 1 multiplexer at the output chooses between an arithmetic output in D_i and a logic output in E_i.
- The data in the multiplexer are selected with inputs S_3 and S_2.The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.
- The circuit whose one stage is specified in the following figure provides eight arithmetic microoperation, four logic microoperations, and two shift microoperations.
- Each operation is selected with the five variables S_3, S_2, S_1, S_0 and C_{in} The input carry C_{in} is used for arithmetic operations only.
- The first eight are arithmetic microoperations, which are selected with S_3S_2 = 00.
- The next four are logic microoperations, which are selected with S_3S_2 = 01.
- The input carry has no effect during the logic microoperations and is marked with don't-care x.
- The last two operations are shift microoperations and are selected with S_3S_2 = 10 for shift right microoperation and S_3S_2= 11 for shift left microoperation. The other three inputs have no effect on the shift.

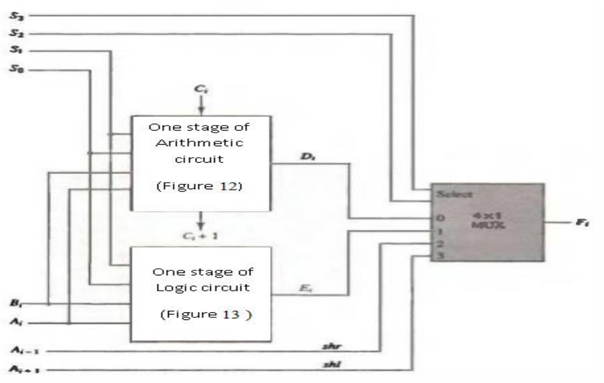


Figure 15: One stage of Arithmetic Logic Shift Unit

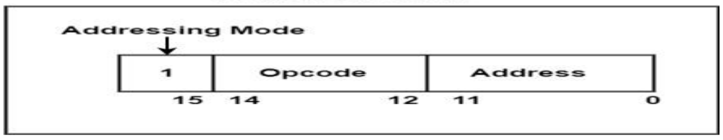
TABLE Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F

1.7 INSTRUCTION CODES

- An instruction code is a group of bits which instructs the computer to perform certain operation.
- Instructions are encoded as binary *instruction codes*. Each instruction code contains a *operation code*, or *opcode*, which designates the overall purpose of the instruction (e.g. add, subtract, move, input, etc.).
- The number of bits allocated for the opcode determines how many different instructions the architecture supports.
- In addition to the opcode, many instructions also contain one or more operands, which indicate where in registers or memory the data required for the operation is located.
- For example, an *add* instruction requires two operands, and a *not* instruction requires one.
- Suppose all instruction codes of a hypothetical accumulator-based CPU are exactly 16 bits. A simple instruction code format could consist of a 4-bit operation code (opcode) and a 12-bit memory address.

Instruction Format



Opcodes

An opcode is a collection of bits that represents the basic operations including add, subtract, multiply, complement, and shift. The total number of operations provided through the computer determines the number of bits needed for the opcode. The minimum bits accessible to the opcode should be n for 2n operations. These operations are implemented on information that is saved in processor registers or memory.

Address

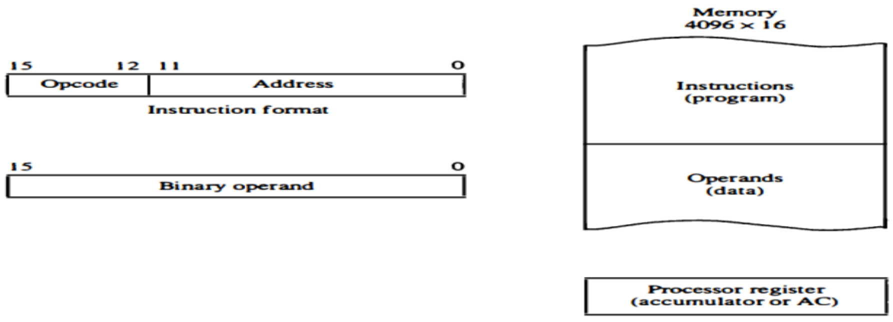
The address is represented as the location where a specific instruction is constructed in the memory. The address bits of an instruction code is used as an operand and not as an address. In such methods, the instruction has an immediate operand. If the second part has an address, the instruction is referred to have a direct address.

Stored Program Organization

- The simplest way to organize a computer is to have one processor register and an instruction code format with two parts.
- The first part specifies the operation to be performed and the second specifies an address.
- The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register. **Figure below** depicts this type of organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$. If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.

  The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.
  It then executes the operation specified by the operation code.

Figure Stored program organization.



- **Computers** that have a single-processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and the content of AC**.** If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes. For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register.They do not need an operand from memory. For these types of operations, the second part of the instruction code (bits 0 through 11) is not needed for specifying a memory address and can be used to specify other operations for the computer.

Indirect Address:-

- It is sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand.
- When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand.
- When the second part specifies the address of an operand, the instruction is said to have a direct address.
- This is in contrast to a third possibility called indirect address, where the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found. One bit of the instruction code can be used to distinguish between a direct and an indirect address. **As an illustration** of this configuration, consider the instruction code format shown in Fig. below part (a). It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.The mode bit is 0 for a direct address and 1 for an indirect address. A direct address instruction is shown in Fig. below part (b). It is placed in address 22 in memory.

  The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
  The control finds the operand in memory at address 457 and adds it to the content of AC. The instruction in address 35 shown in Fig. below part (c) has a mode bit I = 1.
- Therefore, it is recognized as an indirect address instruction. The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand.
  The address of the operand in this case is 1350. The operand found in address 1350 is then added to the content of AC. The indirect address instruction needs two references to memory to fetch an operand. The first reference is needed to read the address of the operand; the second is for the operand itself.

  The effective address to be the address of the operand in a computation-type instruction or the target address in a branch-type instruction. Thus the effective address in the instruction of Fig. below part (b) is 457 and in the instruction of Fig below part (c) is 1350.
  The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data. The pointer could be placed in a processor register instead of memory as done in commercial computers.

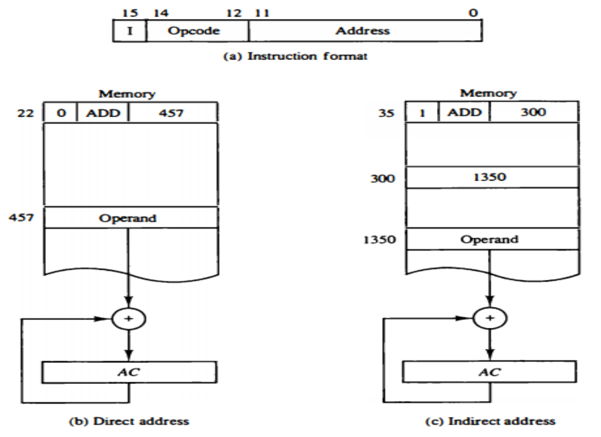


Figure Demonstration of direct and indirect address.

1.8 INSTRUCTION CYCLE

The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.
In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

After the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered or no further instructions to be executed.

1.8.1 Fetch and Decode

- Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T_0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on.
- The Microoperations for the fetch and decode phases can be specified by the following register transfer statements.

  T_0: AR $\leftarrow$ PC
  T_1: IR $\leftarrow$M[AR], PC $\leftarrow$ PC + 1
  T2: D0, • • • , D7 $\leftarrow$ Decode IR(12-14), AR $\leftarrow$ IR(0-11), I $\leftarrow$ IR(l5)
- Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T_0.
- The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T_1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.
- At time T_2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR.The first two register transfer statements are implemented in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection:

1. Place the content of PC onto the bus by making the bus selection inputs S_2 S_1 S_0 equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR since T_0 is 1.

- To implement the second statement

  T_1: IR 🡐 M[AR], PC 🡐 PC + 1

  it is necessary to use timing signal T_1 to provide the following connections in the bus system.
  1. Enable the read input of memory.
  2. Place the content of memory onto the bus by making $S_2S_1S_0$ = 111.
  3. Transfer the content of the bus to IR by enabling the LD input of IR.
  4. Increment PC by enabling the INR input of PC.

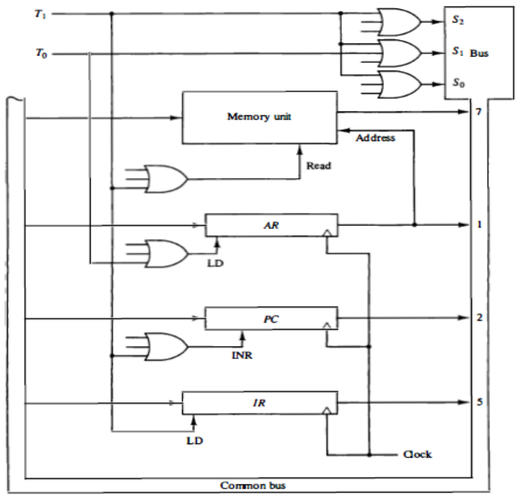


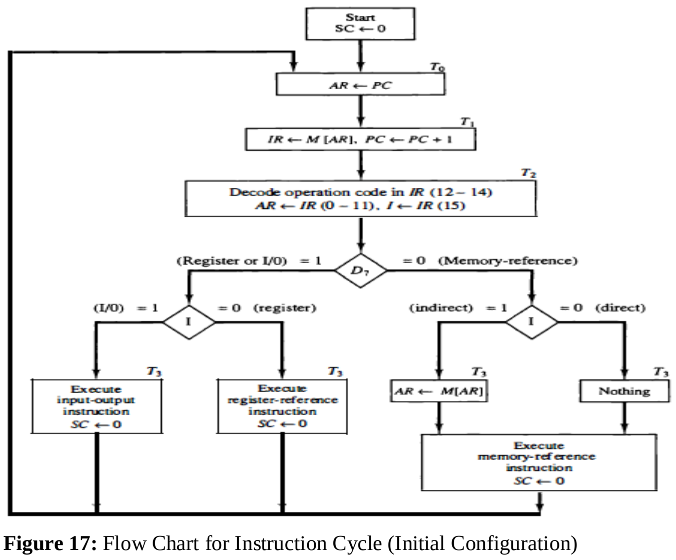
Figure 16: Register Transfers for the Fetch Phase

1.8.2 Determine the Type of Instruction

- Decoder output D7= 1 and I = 1 then it is called I/O reference instruction.
- If D7 = 1, and I = 0 the instruction must be a register-reference.
- If D7 = 0, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.
- If D7 = 0 and I = 1, we have a memory reference instruction with an indirect address.
- If D7 = 0 and I = 0, we have a memory reference instruction with a direct address.
  - The micro operation for the indirect address condition can be symbolized by the register transfer statement.

    AR 🡐 M [AR]
  - The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows:

D_7' IT_3	:	AR 🡐 M [AR]
D_7' I' T_3	:	Nothing
D_7 I' T_3	:	Execute a register-reference instruction
D_7IT_3	:	Execute an input-output instruction



Start
SC ← 0
T_0
AR ← PC
T_1
IR ← M [AR], PC ← PC + 1
T_2
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/0) = 1
D_7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T_3
Execute input-output instruction SC ← 0
T_3
Execute register-reference instruction SC ← 0
T_3
AR ← M[AR]
T_3
Nothing
Execute memory-reference instruction SC ← 0

Figure 17: Flow Chart for Instruction Cycle (Initial Configuration)

1.9 COMPUTER INSTRUCTIONS

The basic computer has three instruction code formats:

1. Memory Reference Instructions
2. Register Reference Instructions
3. Input / Output Instructions

- **The operation code (opcode)** part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

  A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.

- **The register reference instructions** are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.
- **A register-reference** instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
- **Similarly**, an input-output instruction does not need a reference to memory and is recognized by the operation code Ill with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.
- **The type of instruction** is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12- 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I.

Figure Basic computer instruction formats.

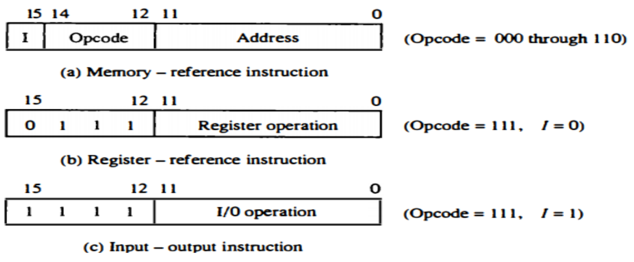


TABLE Basic Computer Instructions

	Hexadecimal code		
Symbol	$I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Register-Reference Instructions

- Register-reference instructions are recognized by the control when $D_7 = 1$ and I = 0.
- These instructions use bits 0 through 111 of the instruction code to specify one of 12 instructions.
- These 12 bits are available in IR(0-11). They were also transferred to AR during time T_2.
- The control functions and microoperations for the register-reference instructions are. listed in Table below. These instructions are executed with the clock transition associated with timing variable T_3.
- Each control function needs the Boolean relation $D_7I'T_3$, which we designate for convenience by the symbol **r**.
- The control function is distinguished by one of the bits in IR(0-11). By assigning the symbol B_i to bit i of IR, all control functions can be simply denoted by rB_i. The next three bits constitute the operation code and are recognized from decoder output D_7. Bit 11 in IR is I and is recognized from B_{11}. The control function that initiates the microoperation for this instruction is $D_7I'T_3B_{11} = rB_{11}$.
- The execution of a register-reference instruction is completed at time T_3.The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal T_0.
- The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers. The next four instructions cause a skip of the next instruction in sequence when a stated condition is satisfied. The skipping of the instruction is achieved by incrementing PC once again (in addition, it is being incremented during the fetch phase at time T_1).
  The condition control statements must be recognized as part of the control conditions .
- The AC is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of AC is zero (AC = 0) if all the flip-flops of the register are zero. The HLT instruction clears a start-stop flip-flop S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

TABLE Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0-11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear *SC*
CLA	rB_{11}:	$AC \leftarrow 0$	Clear *AC*
CLE	rB_{10}:	$E \leftarrow 0$	Clear *E*
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement *AC*
CME	rB_8:	$E \leftarrow \overline{E}$	Complement *E*
CIR	rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment *AC*
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if *AC* zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if *E* zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Memory-Reference Instructions

- In order to specify the rnicro operations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely.
- Table below lists the seven memory-reference instructions. The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table.
- The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1.
- The execution of the memory-reference instructions starts with timing signal T_4. The symbolic description of each instruction is specified in the table in terms of register transfer notation.
- The actual execution of the instruction in the bus system will require a sequence of microoperations. This is because data stored in memory cannot be processed directly.
- The data must be read from memory to a register where they can be operated on with logic circuits.

TABLE Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR],\quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC,\quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC

- This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address.
- The result of the operation is transferred to AC. The microoperations that execute this instruction are:
  D_0T_4:DR←M[AR]
  D_0T_5: AC ← AC ∧ DR, SC ← 0
- The control function for this instruction uses the operation decoder D_0 since this output of the decoder is active when the instruction has an AND operation whose binary code value is 000. Two timing signals are needed to execute the instruction.

ADD to AC

- This instruction adds the content of the memory word specified by the effective address to the value of AC.
- The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop.
- D_1T_4:DR←M[AR]
  D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0

LDA: Load to AC

This instruction transfers the memory word specified by the effective address to AC.

Themicrooperations neededtoexecutethisinstructionare
D_2T_4:DR←M[AR]
D_2T_5: AC ← DR, SC ← 0
The adder and logic circuit receive information from DR which can be transferred into AC. Therefore, it is necessary to read the memory word into DR first and then transfer the content of DR into AC. The reason for not connecting the bus to the inputs of AC is the delay encountered in the adder and logic circuit.

STA: Store AC

- This instruction stores the content of AC into the memory word specified by the effective address**.** Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:
  D_3T_4: M [AR] ← AC, SC ← 0

BUN: Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address.Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle.
PC is incremented at time T_1 to prepare it for the address of the next instruction in the program sequence.

- **The BUN instruction** allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.
- **Theinstruction** isexecutedwithonemicrooperation:
  D_4T_4: PC ← AR, SC ← 0

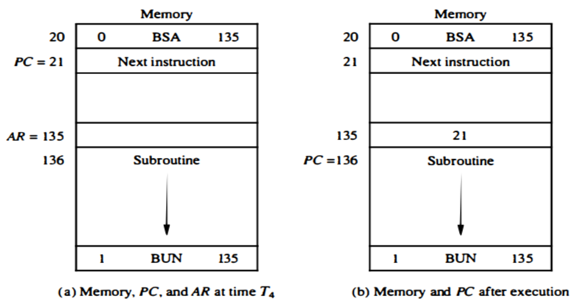
BSA: Branch and Save Return Address

This instruction is useful for branching to a portion of the program called a subroutine or procedure.

- When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.
- The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine.
- This operation was specified in Table above (see Memory-Reference Instructions) with the following register transfer:
  M[AR] ← PC, PC ← AR + 1
  The BSA instruction is assumed to be in memory at address 20. The I bit is 0 and the address part of the instruction has the binary equivalent of 135.
- After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address). AR holds the effective address 135.
- This is shown in part (a) of the figure. The BSA instruction performs the following numerical operation:
  M[135] ← 21, PC ← 135 + 1 = 136

- The result of this operation is shown in part (b) of the figure. The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136.
- The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.
- When this instruction is executed**,** control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21.

Figure Example of BSA instruction execution.



(a) Memory, *PC*, and *AR* at time T_4 (b) Memory and *PC* after execution

- To use the memory and the bus properly, the BSA instruction must be executed With a sequence of two microoperations:
  D_5T_4:M[AR]←PC,AR←AR+1
  D_5T_5: PC ← AR, SC ← 0
- Timing signal T_4 initiates a memory write operation, places the content of PC onto the bus, and enables the INR input of AR .
- The memory write operation is completed and AR is incremented by the time the next clock transition occurs. The bus is used at T_5 to transfer the content of AR to PC .

ISZ: Increment and Skip if Zero

- This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.
- Thefollowingsequenceofmicrooperations:
  D_6T_4:DR←M[AR]
  D_6T_5:DR←DR+1
  D_6T_6: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

Control Flowchart

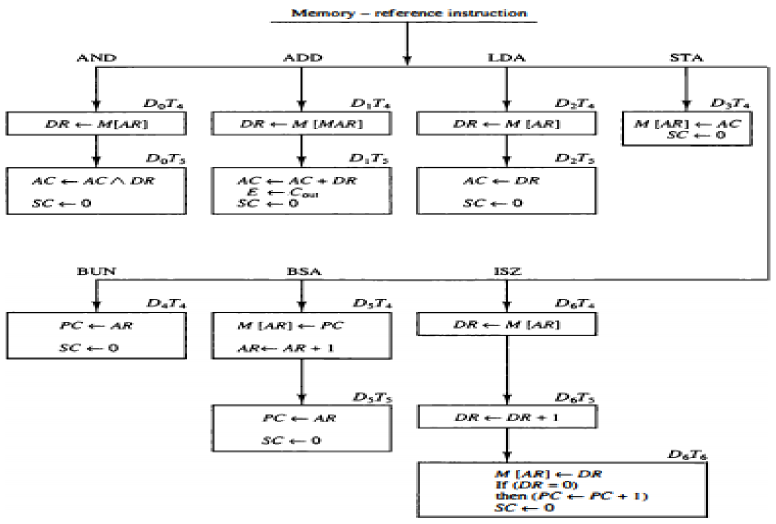


Figure Flowchart for memory-reference instructions.

1.10 INPUT-OUTPUT AND INTERRUPT

- Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device.The serial information from the keyboard is shifted into the input register INPR.
- The serial information for the printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel.The input-output configuration is shown in the following figure 24.

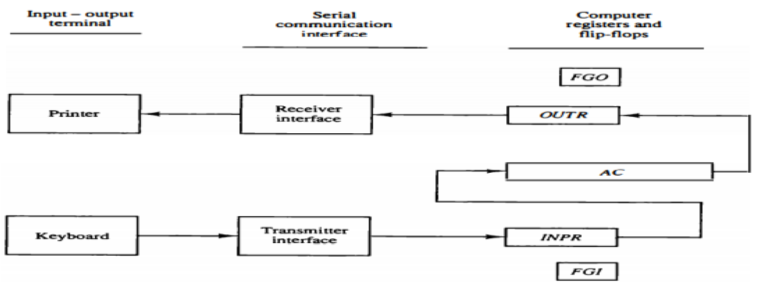


Figure 21: Input-output configuration

- The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially.
- The input register INPR consists of eight bits and holds alphanumeric input information. The 1-bit input flag FGI is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- The output register OUTR works similarly but the direction of information flow is reversed. Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the information, prints the corresponding character, and when the operation is completed, it sets FGO to 1. The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.
- The computer is wasting time while checking the flag instead of doing some other useful processing task. An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.
- In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility. While the computer is running a program, it does not check the flags. However, when a flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that a flag has been set.
- The interrupt enable flip-flop IEN can be set and cleared with two instructions. When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer. When IEN is set to 1 (with the ION instruction), the computer can be
- interrupted.

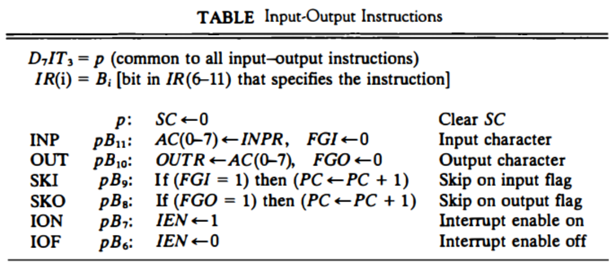


TABLE Input-Output Instructions

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

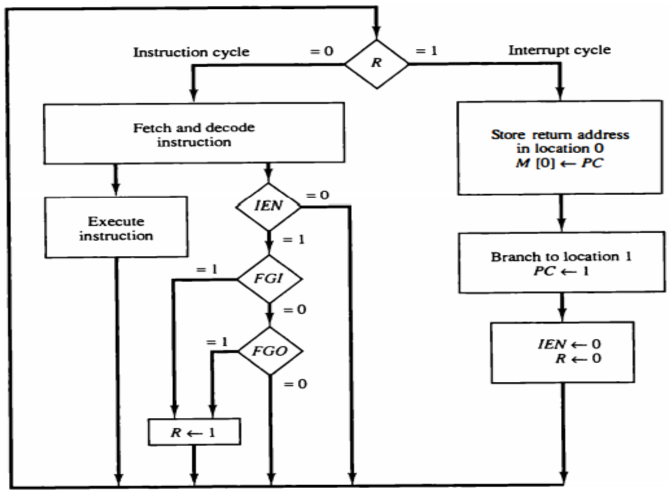


Figure 22: Flowchart for interrupt cycle

Interrupt Cycle

- **The interrupt cycle** is initiated after the last execute phase if the interrupt flip-flop R is equal to 1. This flip-flop is set to 1 if IE N = 1 and either FGI or FGO are equal to 1. With any clock transition except when timing signals T_0, T_1 or T_2 are active. The con dition for setting flip-flop R to 1 can be expressed with the following register transfer statement: T′0T′1T′2(IEN)(FGI+FGO):R← 1 The symbol + between FGI and FGO in the control function designates a logic OR operation. This is ANDed with IEN and T′0 T′1T′2. Instead of using only timing signals T_0, T_1 or T_2 (as shown in Figure in Section - Determine the Type of Instruction) we will AND the three timing signals with R' so that the fetch and decode phases will be recognized from the three control functions R'T_0, R'T_1 and R'T_2.**The reason for this** is that after the instruction is executed and SC is cleared to 0, the control will go through a fetch phase only if R = 0. Otherwise, if R = 1, the control will go through an interrupt cycle. **The interrupt cycle** stores the retu rn address (available in PC) into memory location 0, branches to memory location 1, and clears IEN, R, and SC to 0. This can be done with the following sequence of microoperations:
- RT_0: AR ← 0, TR ← PC
- RT_1: M[AR] ← TR, PC ← 0
- RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

- **During the first timing signal AR** is cleared to 0, and the content of PC is transferred to the temporary register TR.With the **second timing signal**, the return address is stored in memory at location 0 and PC is cleared to 0. The third timing signal increments PC to 1, clears IEN and R, and control goes back to T_0 by clearing SC to 0.**The beginning** of the next instruction cycle has the condition R'T_0 and the content of PC is equal to 1.**The control** then goes through an instruction cycle that fetches and executes the BUN instruction in location 1

CO UNIT -1 QUESTIONS

1. Explain the life cycle of an instruction.
2. How is the Register reference instruction different from memory referenced instruction.
3. Explain Arithmetic Logic Shift unit.
4. Discuss Functional Units.
5. What is effective address? Briefly describe memory reference instructions.
6. Define Register Transfer Language. Explain the instructions with suitable examples.
7. What is a micro-operation? Explain different micro-operations.
8. Explain about Register reference instructions.
9. What is an Interrupt? Explain the Flowchart of interrupts.
10. Illustrate how various registers and memory are connected using a common bus system.
11. Design a bus system for connecting 4 registers each of size 8 bits.
12. Construct a 4-bit Adder/Subtractor circuit.
13. Describe about Logical Microoperations.
14. Explain about Arithmetic Microoperations.
15. What is effective address? Briefly describe memory reference instructions.
16. Design a bus system for connecting 6 registers each of size 4 bits.
17. Explain various shift micro operations with example.
18. Illustrate the flowchart for interrupt life cycle for an instruction.
19. What are the different types of computer instructions? Explain how they are differentiated.
20. Outline Instruction Codes. Explain the instruction cycle.
21. Describe basic computer registers and draw block diagram of register, basic computer registers and memory.
22. Explain the tristate buffer.