UNIT – I

Register transfer language and Micro operations

Objective:

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

Syllabus:

Functional units, Computer Registres, Register Transfer language, Register Transfer Bus and memory transfers, Arithmetic, logic and shift micro-operations, Arithmetic logic shift unit.
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

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.

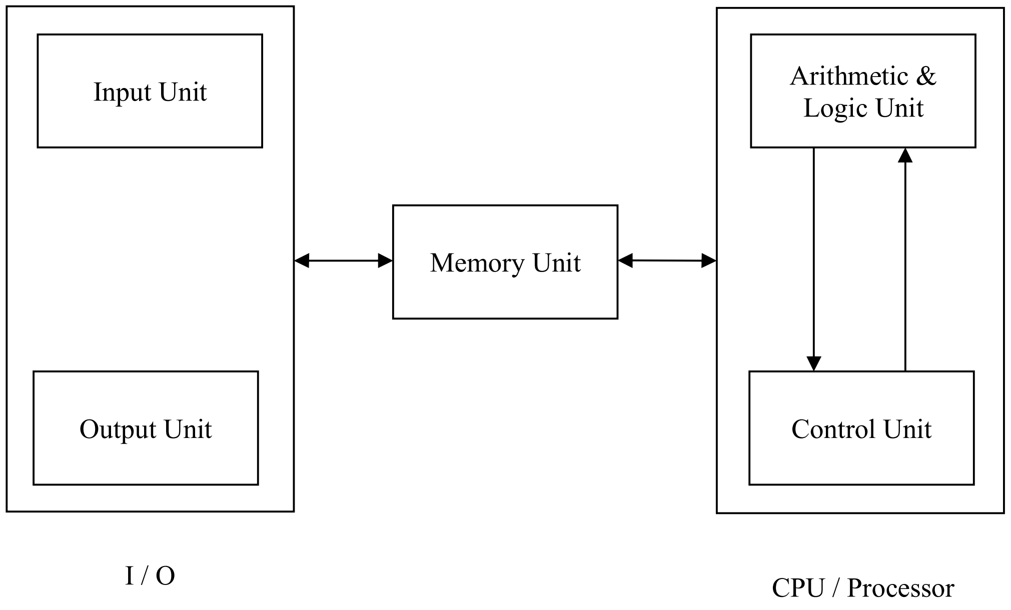


Figure 1: Basic functional units of a computer

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

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 govern the data transfers and then appropriate operations take place.
Control unit interprets or decides the operation/action to be performed.

1.2 COMPUTER REGISTERS

- 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.

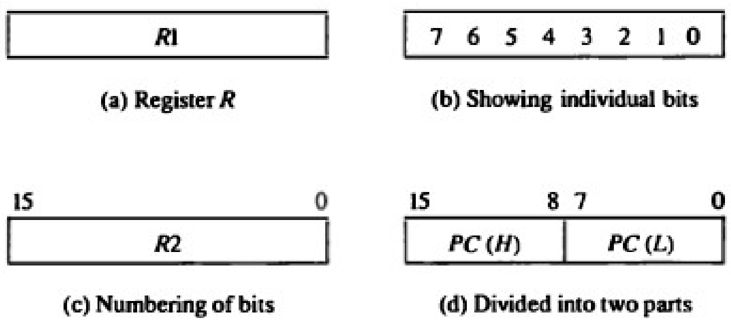


Figure 2: Block diagram of register

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

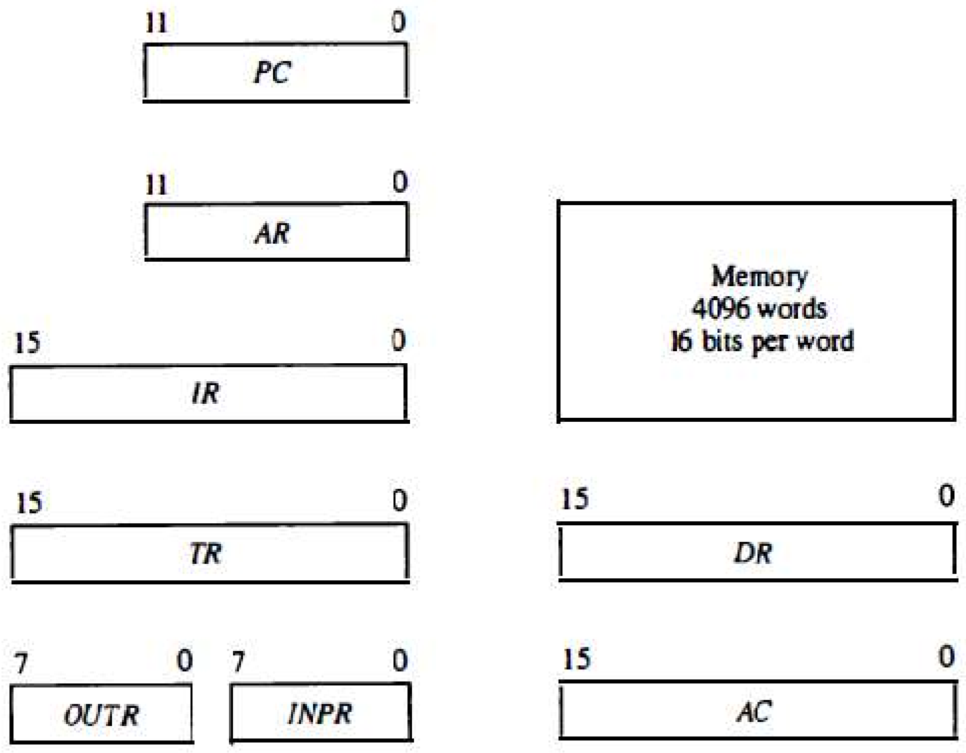


Figure 3: Basic computer registers and memory

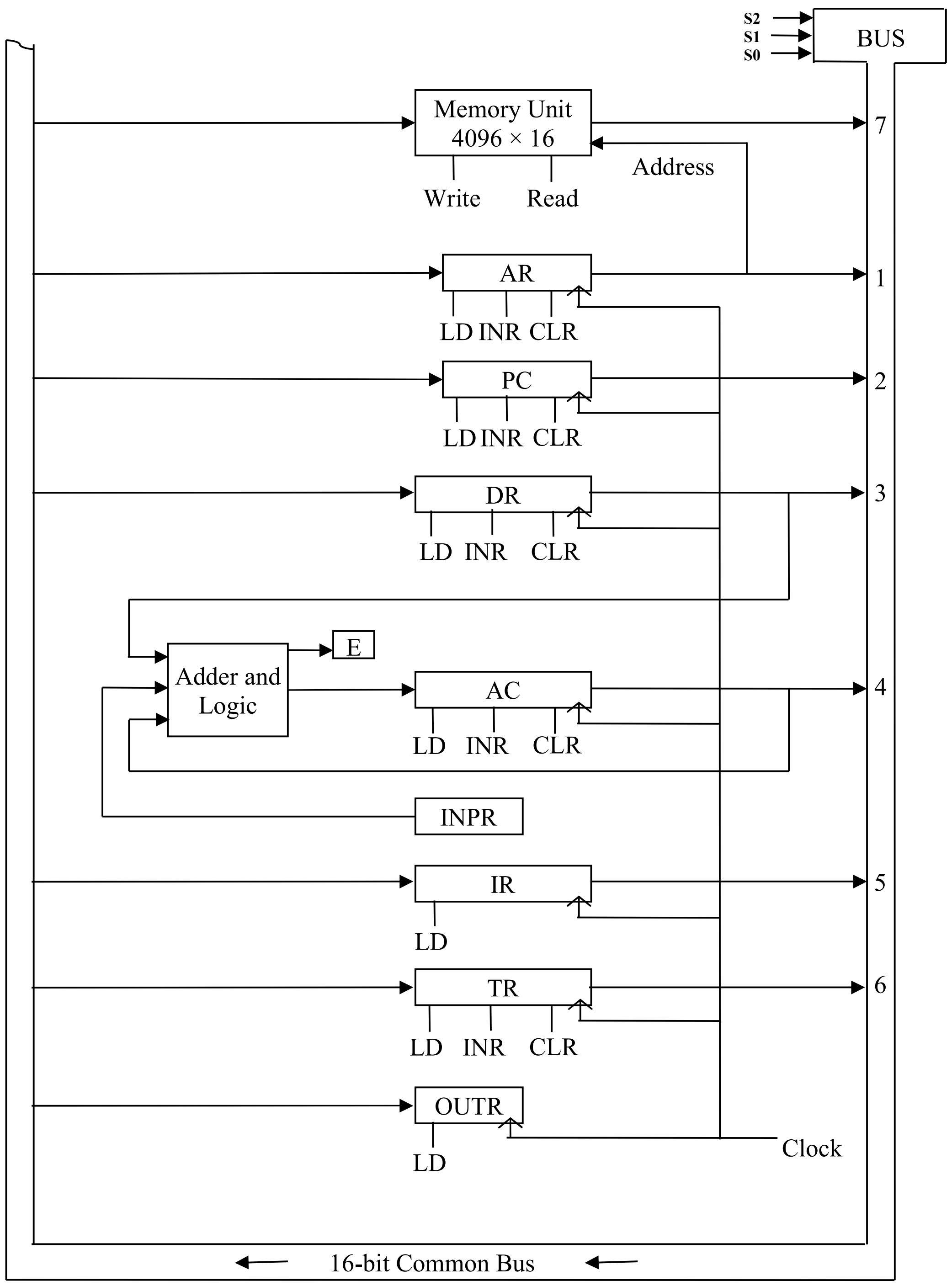


Figure 4: Basic Computer Register Connected to a Common BUS

- 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 $S_2S_1S_0$=11.
- Four registers, D R, 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.

Micro operation: is an elementary operation performed on information stored in one or more registers.
Eg: Shift, Count, Clear and Load.

1.3 REGISTER TRANSFER LANGUAGE

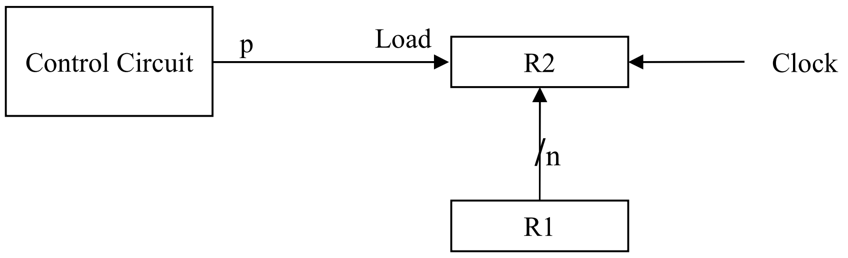
- The symbolic notation used to describe the Micro operation transfers among registers is called a Register Transfer Language.
- Information transferred from one register to another register is designed in symbolic form by means of replacement operator($\leftarrow$).
- The statement R2 $\leftarrow$ R1 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 $\leftarrow$ R1

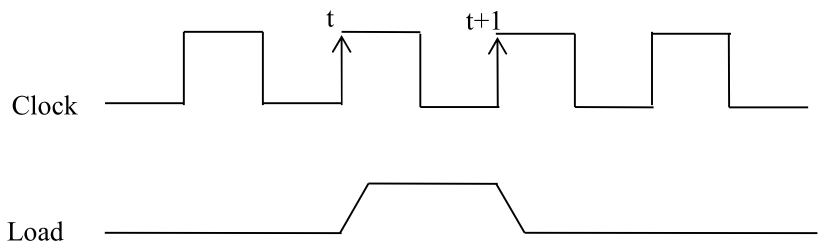
  where p is a control signal generated in the control section.
- A Control function is a Boolean variable that is equal to 0 or 1.
- The control function included in the statement is represented as follows.

  p: R2 $\leftarrow$ R1

- Here the transfer operation is performed by hardware only if p = 1.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.



(a) Block Diagram



(b) Timing Diagram

Figure 5: Transfer from R1 to R2 when p =1

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

Table 2: Basic Symbols for Register Transfer

S.No	Symbol	Description	Example
1	Letters (and Numbers)	Denotes a register	MAR, R2
2	Parenthesis ( )	Denotes a part of register	R2(0-7), R2(L)
3	Arrow $\leftarrow$	Denotes transfer of information	R2 $\leftarrow$ R1
4	Comma ,	Separates two microoperations	R2 $\leftarrow$ R1, R1 $\leftarrow$ R2

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

Eg: R2 $\leftarrow$ R1, R1 $\leftarrow$ 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 is shown in the following figure.

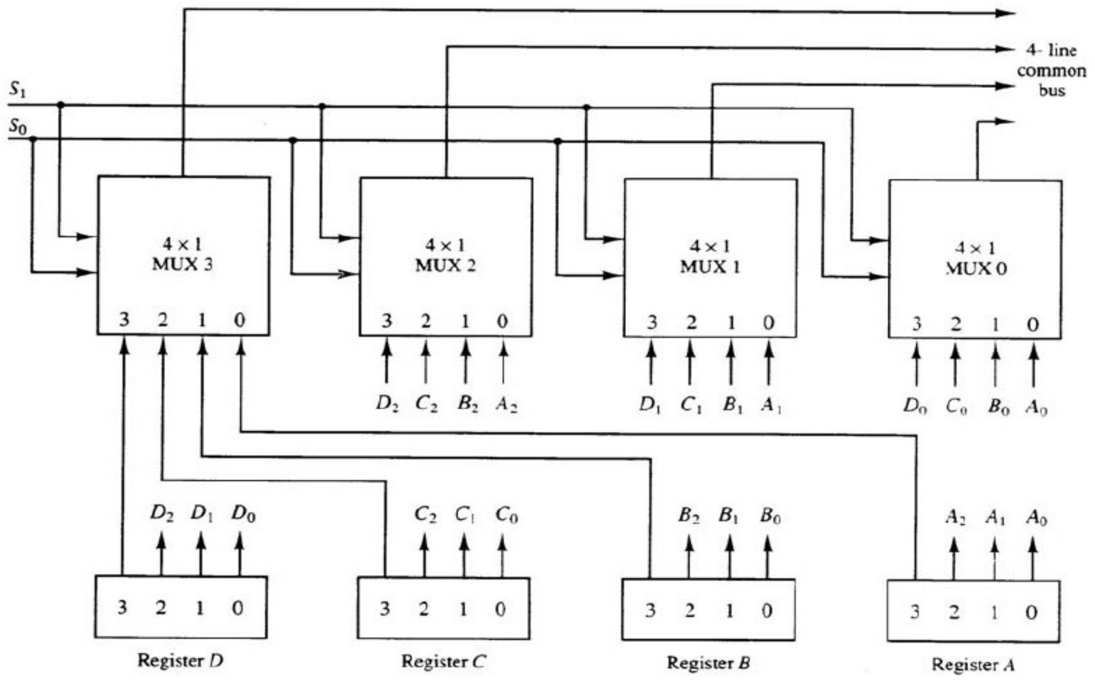


Figure 6: Bus system for four registers

- 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.
- When $S_1S_0 = 00$, then 0 data input of all four multiplexers are selected and applied to output that form the bus.
- 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

S_1	S_2	**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 $\leftarrow$ C
- In the above statement the content of register C is placed onto BUS

  Eg: R1 $\leftarrow$ 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.

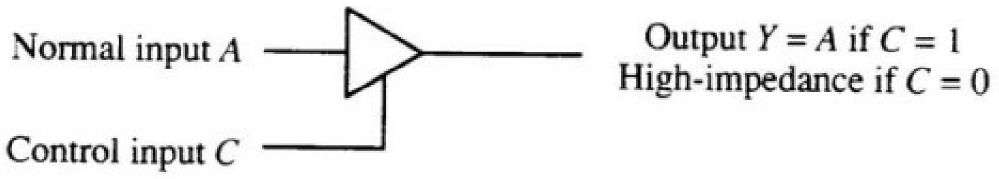


Figure 7: Graphic symbol of a three state buffer

- 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.

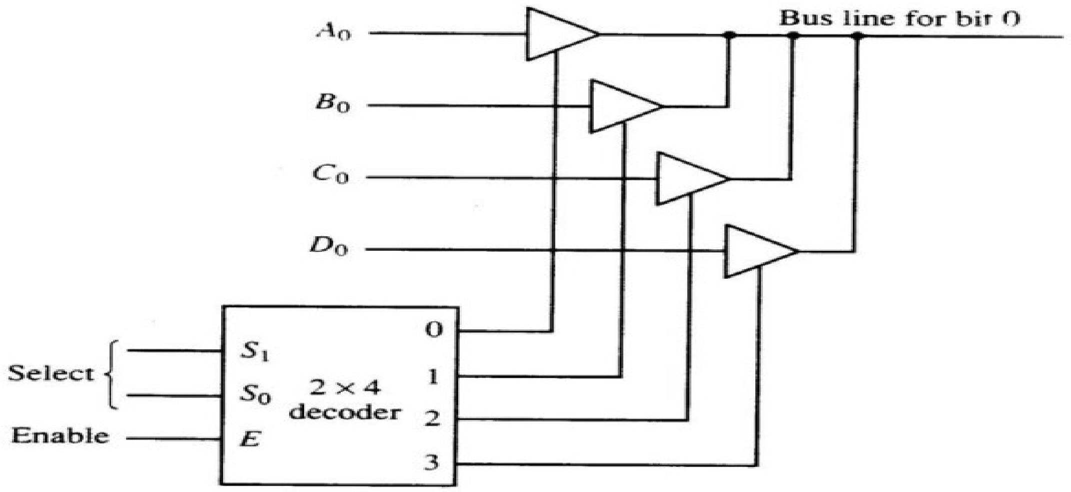


Figure 8: Bus Line with three state buffer

- 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.

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.

Add Microoperation:

$$R3 \leftarrow R1 + R2$$

- 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.

Subtract Microoperation:

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

- 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 5: 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.

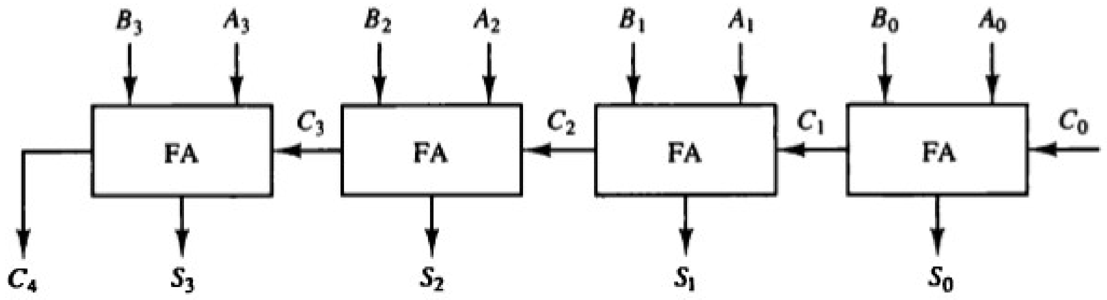


Figure 9: 4-bit binary adder

- 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.
- A 4-bit adder-subtractor circuit is shown in the following figure.

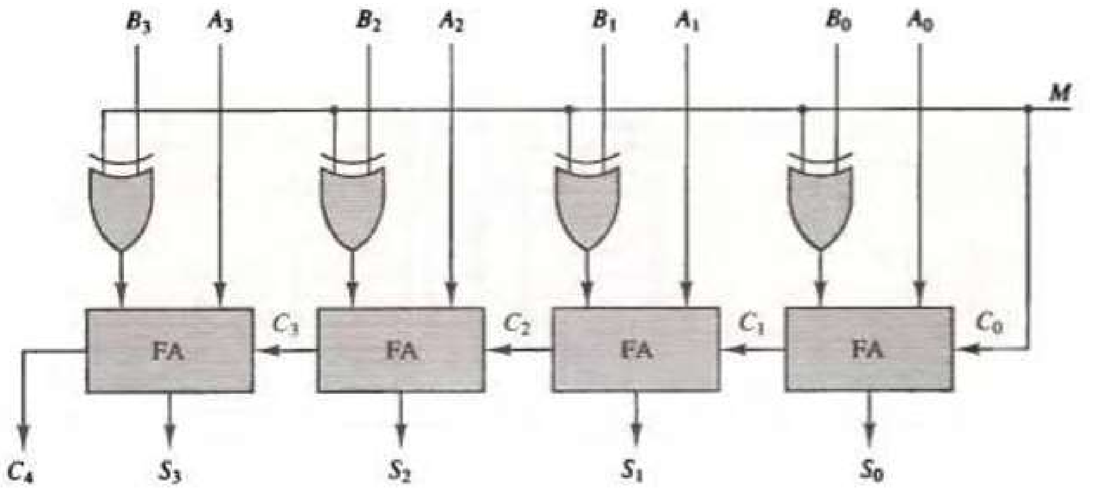


Figure 10: Adder-subtractor

- 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⊕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⊕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.

1.4.3 Binary Incrementer

- 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.

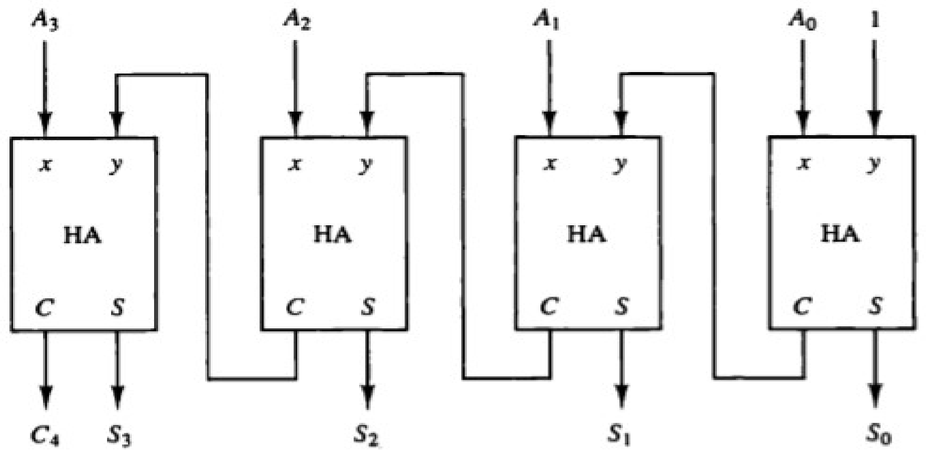


Figure 11: 4-Bit Binary Incrementer

- The 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 four multiplexers are controlled by two selection inputs, S_1 and S_0.

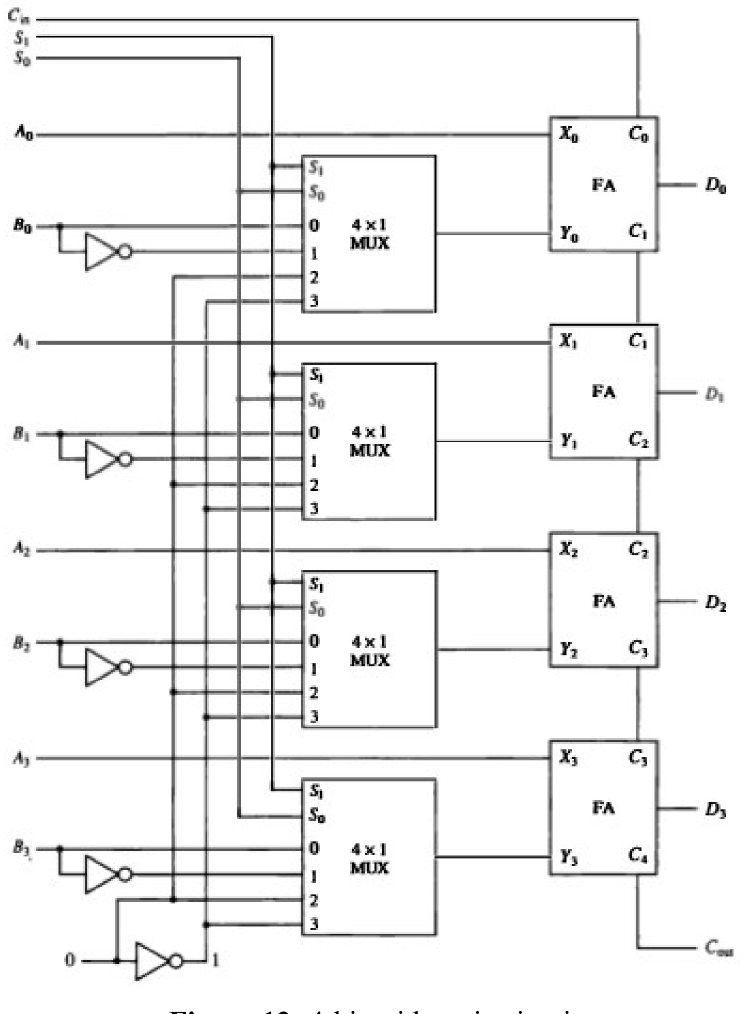


Figure 12: 4-bit arithmetic circuit

- 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

1.5 LOGIC MICROOPERATIONS

- Logic microoperations 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 R 1 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.

List of Logic Microoperations

- 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

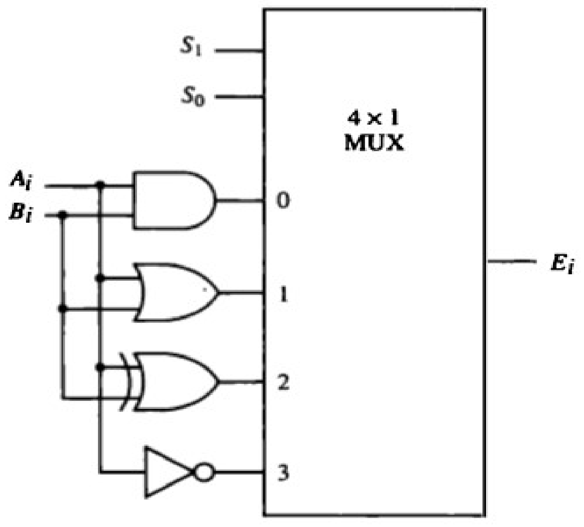
- In the following table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x and y. Note that the functions are determined from the 16 binary combinations that can be assigned to F.

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 mnicrooperations. 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	$E = \bar{A}$	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
$\underline{1100}$	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
1100	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
1100	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
<u>1010</u>	B
0000	A ←A⊕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 logical 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 logical arithmetic shift-left rnicrooperation.

  ashr used forarithmetic 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

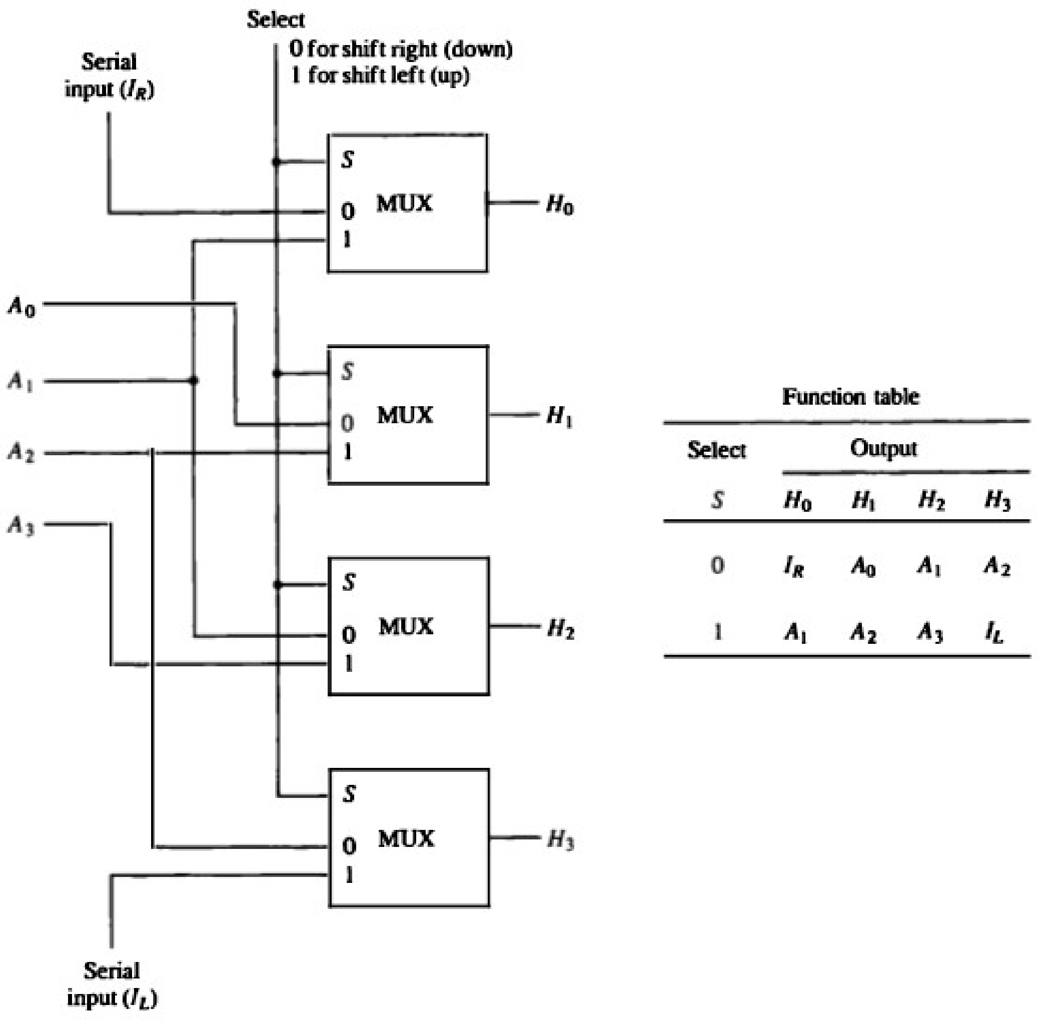
After performing arithmetic shift right, R1 has the content 1101

Table 9: Shift Microoperations

Symbolic designation	**Description**
R ← shl R	Shift-left register R
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left register R
R ← cir R	Circular shift-right register R
R ← ashl R	Arithmetic shift-left R
R ← 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.



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.

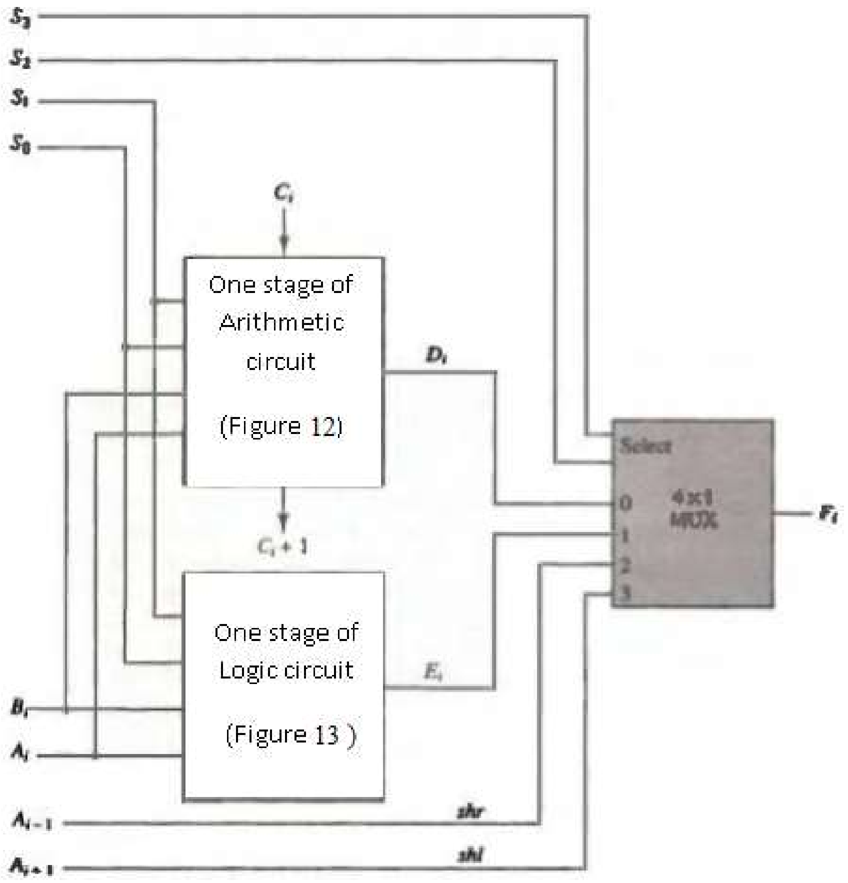
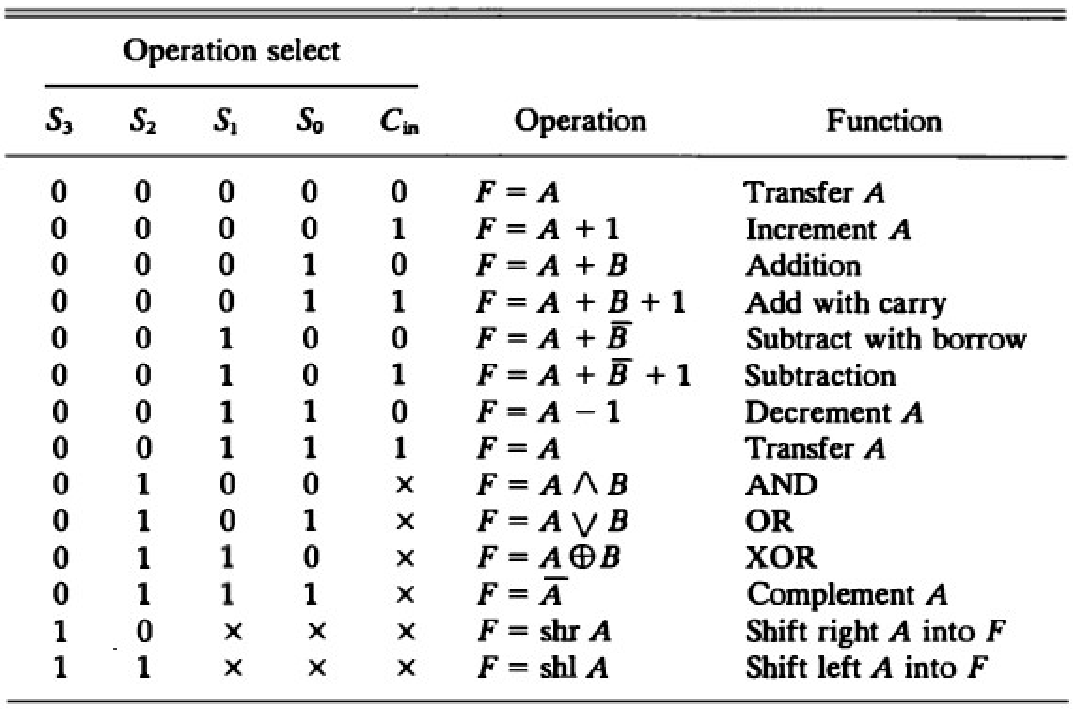


Figure 15: One stage of Arithmetic Logic Shift Unit

- 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 above 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=and 11 for shift left microoperation. The other three inputs have no effect on the shift.

Table 10: 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	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	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.

```
 15     12 11         6 5         0
+-----------------------------------+
| Opcode |  Operand |Operand   |
+-----------------------------------+
```

- 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.

```
  15    12 11          0
 +-----------------------+
| Opcode | Address      |
 +-----------------------+
```

1.7.1 Stored Program Organization

- The simplest way to organize a computer is to have one process register and an instruction code formats with two parts.
- The first part specifies the operation to be performed and the second specifies the address.
- The instructions are stored in one section of the memory and the data is stored in the another section.
- The figure considers the memory of size 4096 x 16. The number of Address lines required to represent the 4096 words are 12 because $4096=2^{12}$. The number of data lines required are 16.

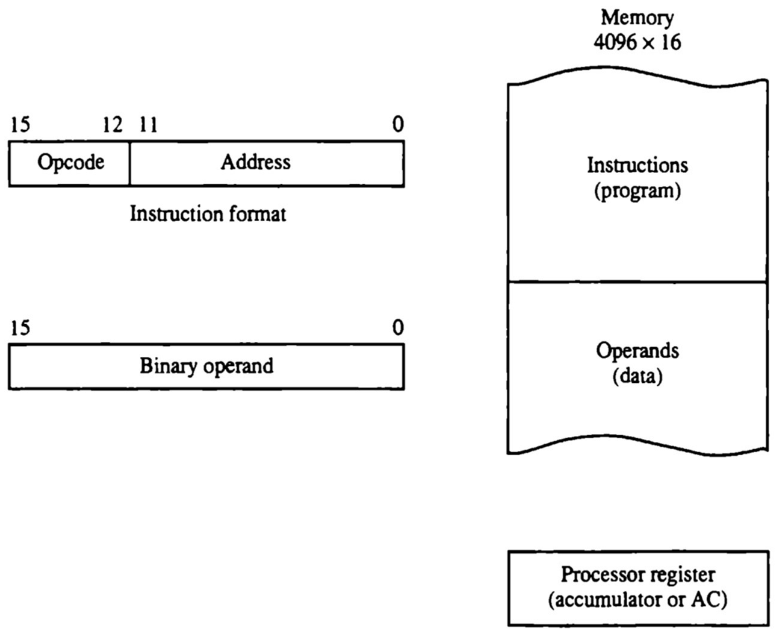


Figure 16: Stored program organization

1.8 INSTRUCTION FORMAT

- A 16 bit instruction code format could consist of a 4-bit operation code (opcode) and a 12-bit memory address, where the 4 bits are divided into 2 parts as shown in the figure below.
- The bits 12-14 represent the operation code and the 15th bit which is represented with I states whether the given address is Direct or Indirect.

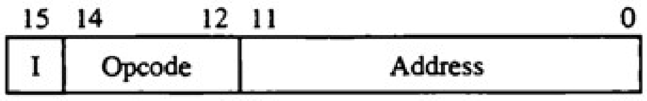


Figure 17: Instruction format

- If the value of I = 0 then the given address is Direct address and if the value of I = 1 then the given address is Indirect address.
- Direct : Instruction code contains address of operand.
- Immediate : Instruction code contains operand
- Indirect : Instruction code contains address of address of operand.
- Effective address = actual address of the data in memory.

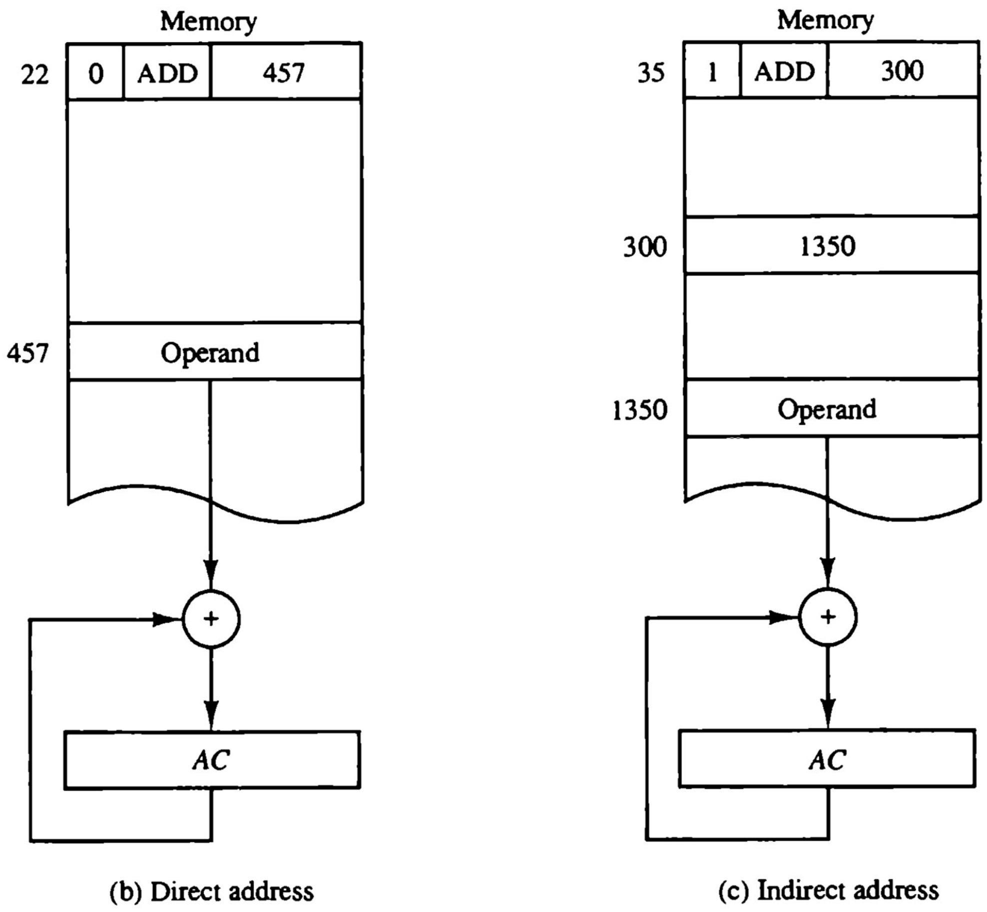


Figure 18: Demonstration of direct and indirect address

Example: The following examples use the following piece of computer memory.

1.9 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.

1.9.1 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), 1 $\leftarrow$ 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. To implement this it is necessary to use timing signal T_1 to provide the following connections in the bus system.

     T_1: IR $\leftarrow$ M[AR], PC $\leftarrow$ PC + 1

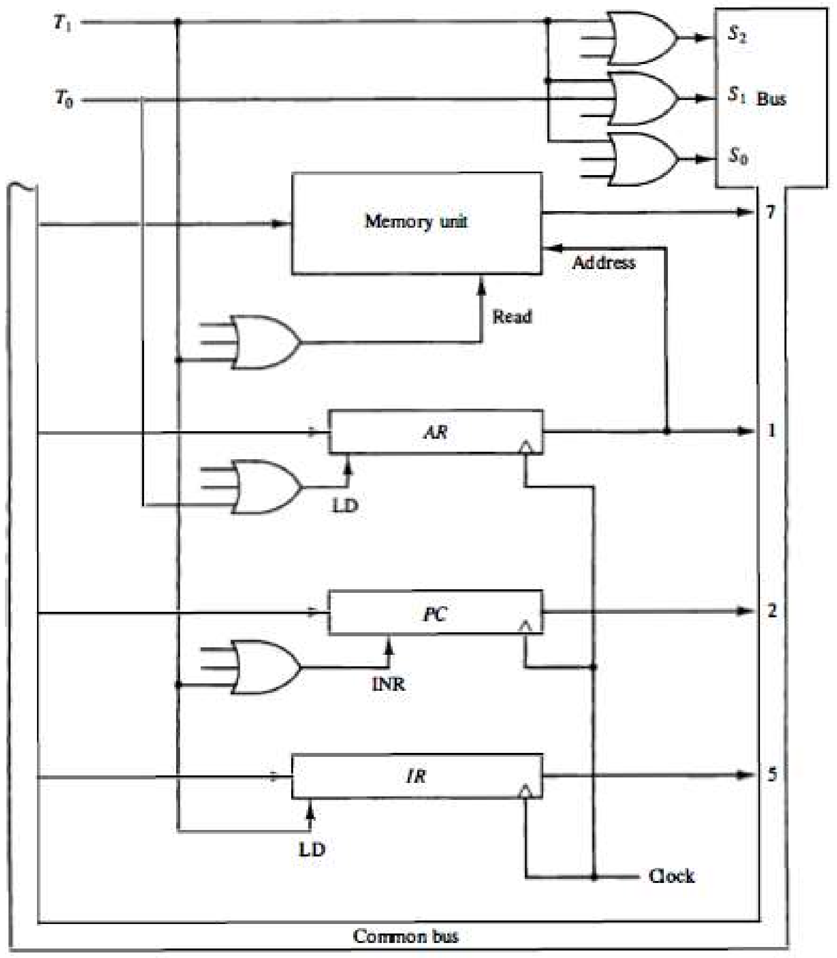


Figure 19: Register Transfers for the Fetch Phase

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.

1.9.2 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.

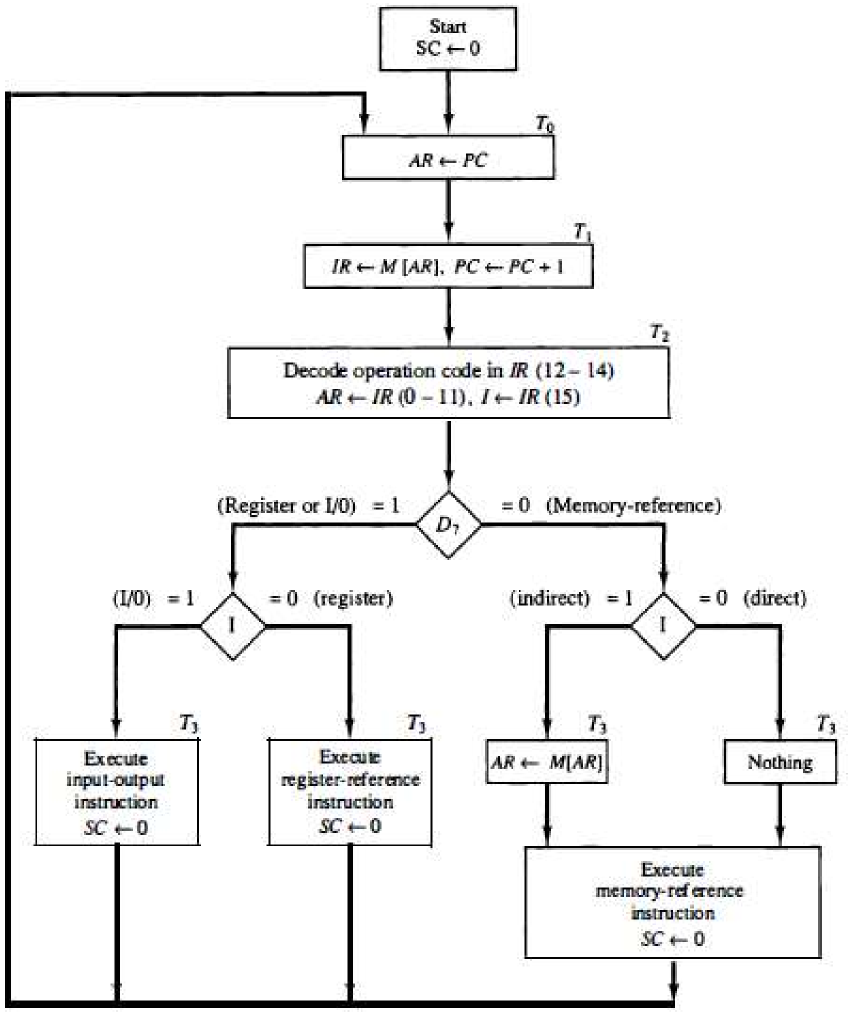

Start
SC ← 0
T_0
AR ← PC
T_1
IR ← M [AR], PC ← PC + 1
T_2
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/0) = 1
D_7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T_3
Execute input-output instruction SC ← 0
T_3
Execute register-reference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

Figure 20: Flow Chart for Instruction Cycle (Initial Configuration)

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

1.10 COMPUTER INSTRUCTIONS

The basic computer has three instruction code formats:

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

1.10.1. Memory Reference Instructions

In Memory reference instruction:

- First 12 bits(0-11) specify an address.
- Next 3 bits specify operation code (opcode) and can range from 000 to 110.
- Left most bit specify the addressing mode I
- I = 0 for direct address
- I = 1 for indirect address
- The address field is denoted by three x's (in hexadecimal notation) and is equivalent to 12-bit address.
- When I = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is zero (0).
- When I = 1 the last four bits of an instruction have a hexadecimal digit equivalent from 8 to E since the last bit is one (1).

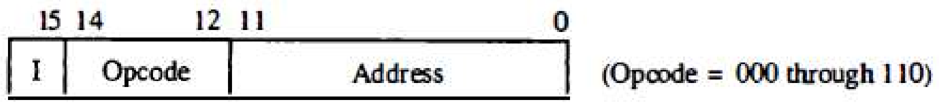


Figure 21: Memory - reference instruction format

Table 11: Basic Computer Instructions for Memory Reference Instructions

Symbol	Hexadecimal code		Description
	I = 0	**I = 1**	
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

1.10.2. Register Reference Instructions

In Register Reference Instruction:

- First 12 bits (0-11) specify the register operation.
- The next three bits equals to 111 specify opcode.
- The last mode bit of the instruction is 0.

- Therefore, left most 4 bits are always 0111 which is equal to hexadecimal 7.

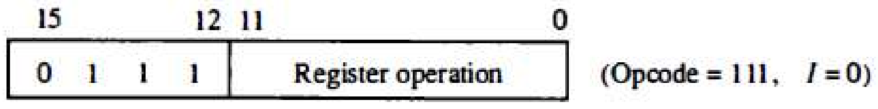


Figure 22: Register - reference instruction format

Table 12: Basic Computer Instructions for Register - reference instructions

Symbol	Hexadecimal code	Description
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 is AC is negative
SZA	7004	Skip next instruction is AC is 0
SZE	7002	Skip next instruction is E is 0
HLT	7001	Halt computer

1.10.3. Input / Output Instructions

In I/O Reference Instruction:

- First 12 bits (0-11) specify the I/O operation.
- The next three bits equals to 111 specify opcode.
- The last mode bit of the instruction is 1.
- Therefore, left most 4 bits are always 1111 which is equal to hexadecimal F.

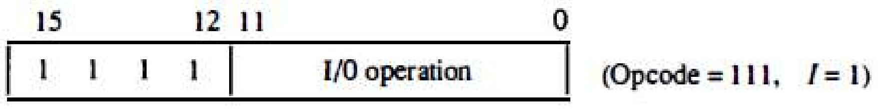


Figure 23: Input - output instruction format

Table 13: Basic Computer Instructions for Input - output instructions

Symbol	Hexadecimal code	Description
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

1.11 MEMORY-REFERENCE INSTRUCTIONS

- In order to specify the micro operations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely.
- The function of the memory-reference instructions can be defined precisely by means of register transfer notation. Table given 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 data must be read from memory to a register where they can be operated on. We will see the operation of each instruction and list the control functions and micro operations needed for their execution.

Table 14: Memory Reference Instructions

Symbol operation	Decoder	Symbolic description
AND	D_o	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[A\ R]$, $E \leftarrow C_{OUT}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$,if $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

- After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of micro operations that the control follows during the execution of a memory-reference instruction.

AND to AC

- This is an instruction that performs the logic AND 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 micro operations that execute this instruction are:

D_0T_4: $DR \leftarrow M\,[AR]$

D_0T_5: $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

- 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 bit of accumulator). The micro operations needed to execute this instruction are

D_1T_4: $DR \leftarrow M\,[AR]$

D_1T_5: $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$

LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC. The micro operations needed to execute this instruction are

  D_2T_4: DR ← M [AR]

  D_2T_5: AC ← DR, SC ← 0

- As we know that there is no direct path from the bus into AC. 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.

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 complete the execution of this instruction in 1 timing signal.

  D_3T_4: M [AR] ←AC, SC ←0

BUN: Branch Unconditionally

- It allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally to the location specified by effective address. The instruction is executed with one micro operation:

  D_4T_4: PC ←AR, SC ←0

Branch and Save Return Address (BSA)

- 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.

  M [AR] ←PC, PC ←AR + 1

- 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.
- The BSA instruction performs the function usually referred to as a subroutine call. The indirect BUN instruction at the end of the subroutine performs the function referred to as a subroutine return.

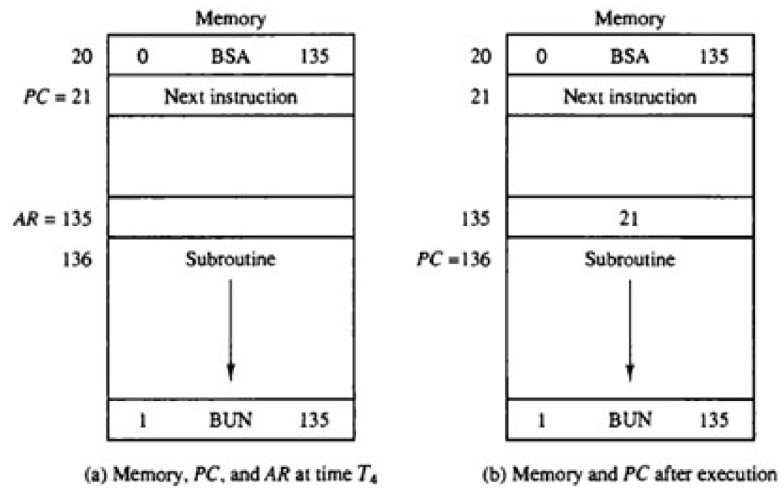


Figure 24: Example of BSA Instruction Execution

- It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer.
- To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two micro operations:

  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 T5 to transfer the content of AR to PC.

Increment and Skip if Zero (ISZ)

- This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.
- The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by 1 in order to skip the next instruction in the program.
- Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of micro operations:

  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

1.11.1 Control Flowchart

- A flowchart showing all micro operations for the execution of the seven memory- reference instructions is shown in Fig. given below.
- The control functions are indicated on top of each box. The micro operations that are performed during time T_4, T_5, or T_6, depending on the opcode value. This is indicated in the flowchart by six different paths, one of which the control takes after the instruction is decoded.
- The sequence counter SC is cleared to 0 during the last timing signal in each case. This causes a transfer of control to timing signal T_0 to start the next instruction cycle.
- Note that we need only seven timing signals to execute the longest instruction (ISZ) requiring the 3-bit sequence counter.

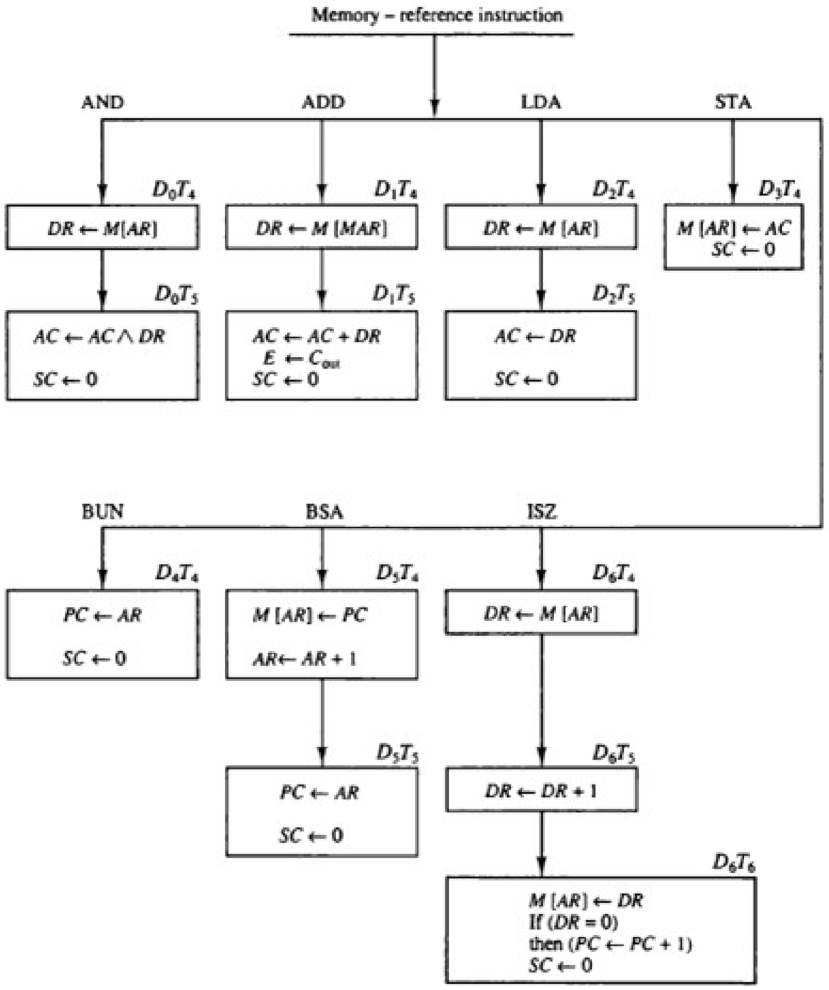


Figure 25: Flowchart for Memory Reference Instructions