UNIT-I

Digital Computers: Introduction, Block diagram of Digital Computer, Definition of Computer Organization, Computer Design and Computer Architecture.

Register Transfer Language and Micro operations: Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, logic micro operations, shift micro operations, Arithmetic logic shift unit.

Basic Computer Organization and Design: Instruction codes, Computer Registers Computer instructions, Timing and Control, Instruction cycle, Memory Reference Instructions, Input – Output and Interrupt.

The digital computer:

- The digital computer is a digital system that performs various computational tasks.
- The word digital implies that the information in the computer is represented by variables that take a limited number of discrete values.
- The first electronic digital computers, developed in the late 1940s, were used primarily for numerical computations.
- In this case the discrete elements are the digits. From this application the term digital computer has emerged.
- In practice, digital computers function more reliably if only two states are used. Digital computers use the binary number system, which has two digits: **0 and 1**. A binary digit is called a bit.
- Information is represented in digital computers in groups of bits. By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols.
- In contrast to the common decimal numbers that employ the base 10 system, binary numbers use a base 2 system with two digits: 0 and I. The decimal equivalent of a binary number can be found by expanding it into a power series with a base of 2.
- **For example,** the binary number 1001011 represents a quantity that can be converted to a decimal number by multiplying each bit by the base 2 raised to an integer power as follows:

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 75$$

- The seven bits 1001011 represent a binary number whose decimal equivalent is 75. However, this same group of seven bits represents the letter K when used in conjunction with a binary code for the letters of the alphabet.

- A computer system is sometimes subdivided into two functional entities: hardware and software.

The hardware of the computer is usually divided into three major parts, as shown in Fig. 1-A. The central processing unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and control circuits for fetching and executing instructions.

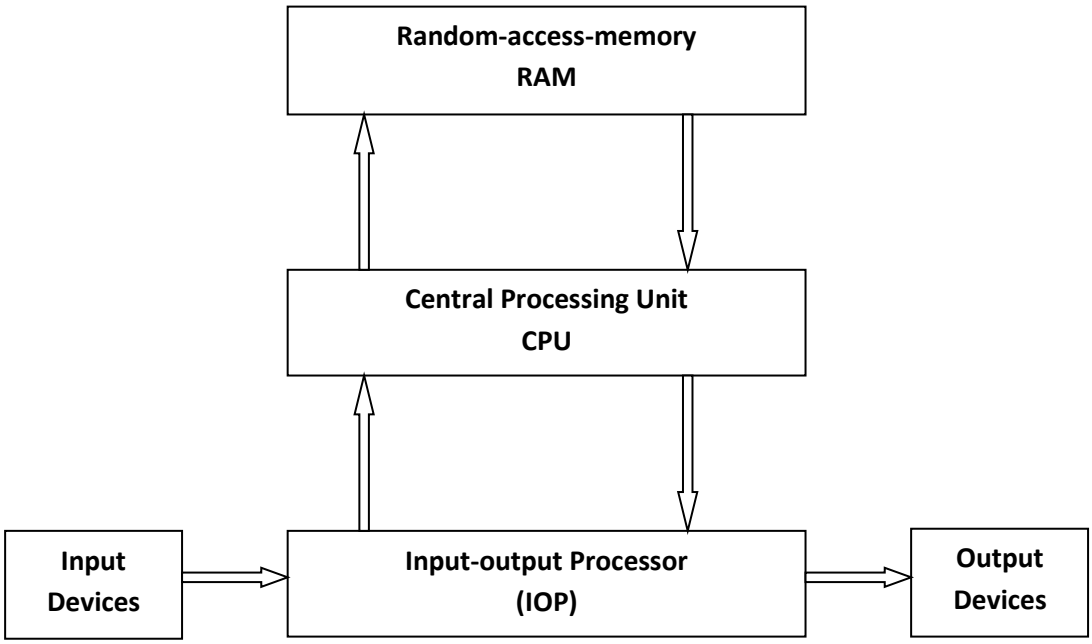


Figure 1-A: Block diagram of a digital computer.

- The memory of a computer contains storage for instructions and data. It is called a random access memory (RAM) because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time.
- The input and output processor (lOP) contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world.
- The input and output devices connected to the computer include keyboards, printers, terminals, magnetic disk drives, and other communication devices.

- **Computer organization** is concerned with the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be

in place and the task is to investigate the organizational structure to verify that the computer parts operate as intended.

- **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 behavior of the computer as seen by the user. It includes the information formats, the instruction set, and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system.

Register Transfer language:

- **Register**: A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters).
- 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.
- The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig. I-(a). The individual bits can be distinguished as in (b). The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c). A 16-bit register is partitioned into two parts in (d).

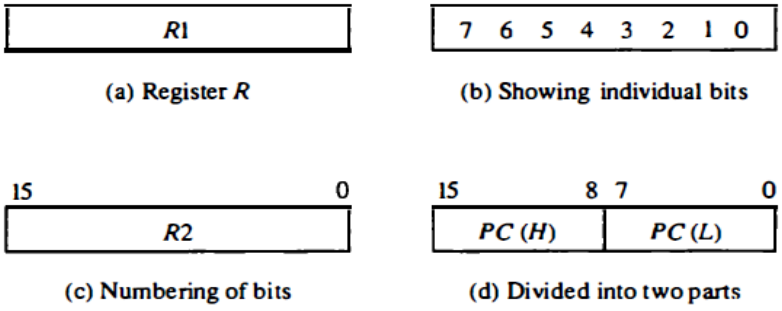


Fig: I-Block diagram of registers

- Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC . The symbol PC(0-7) or PC(L) refers to the low-order byte and PC(S-15) or PC(H) to the high-order byte.
- **Micropoeration**: The operations executed on data stored in registers are called micro operations.
  - ✓ A micro operation is an elementary operation performed on the information stored in one or more registers.
  - ✓ The result of the operation may replace the previous binary information of a register or may be transferred to another register.
  - ✓ Examples of micro operations are shift, count, clear, and load.
- **Register transfer language**- The symbolic notation used to describe the micro operation transfers among registers is called a register transfer language.
  - ✓ The term "register transfer" implies the availability of hardware logic circuits that can perform a stated micro operation and transfer the result of the operation to the same or another register.
  - ✓ A register transfer language is a system for expressing in symbolic form the micro operation sequences among the registers of a digital module.
- Information transfer from one register to another is designated in symbolic form by means of a replacement operator.

  The statement

  $R2 \leftarrow R1$

  denotes a transfer of the content of register R1 into register R2. It designates a replacement of the content of R2 by the content of R l . By definition, the content of the source register R1 does not change after the transfer

- Normally, we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if-then statement.

  If (P = 1) then ($R2 \leftarrow R1$) /* Register transfer with control function */

  Where P is a control signal generated in the control section
  - ✓ A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

    P: $R2 \leftarrow R1$

Bus and memory transfers:

- A typical digital computer h a s many registers, and paths must b e provided to transfer information from one register to another.
- The number of wires will be excessive if separate lines are used between each register and all other registers in the system
- A more efficient scheme for transferring information between registers in a multiple-register configuration is a **common bus system**.

Common Bus System:

- One way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus.
- The construction of a bus system for four registers is shown in Fig. I-B. Each register has four bits, numbered 0 through 3.
- The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S1 and S0

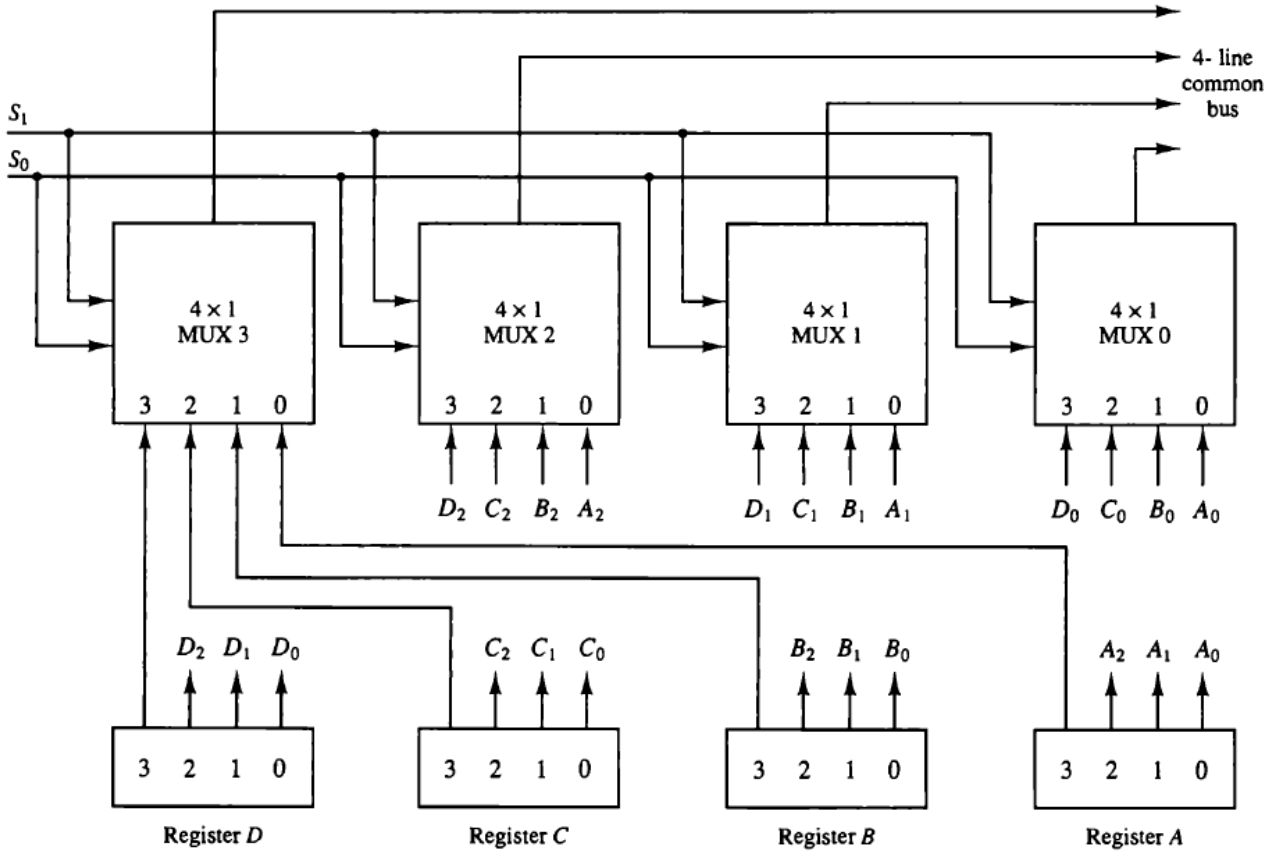


Fig I-B: Bus system for four registers

NOTE: In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers. For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labeled A1.

- The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus. Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.
- 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 S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus.
  - ✓ This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers. Similarly, register B is selected if S1S0 = 01, and so on.
- Table I-C shows the register that is selected by the bus for each of the four possible binary value of the selection lines

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

Table I-C: Function Table for Bus of Fig. I-B

NOTE: In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n , the number of bits in each register. The size of each multiplexer must be k x 1 since it multiplexes k data lines. For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

Three (Tri) - State 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 and 0 as in a conventional gate.
- The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- The graphic symbol of a three-state buffer gate is shown in Fig. I-D. It is distinguished from a normal buffer by having both a normal input and a control input.

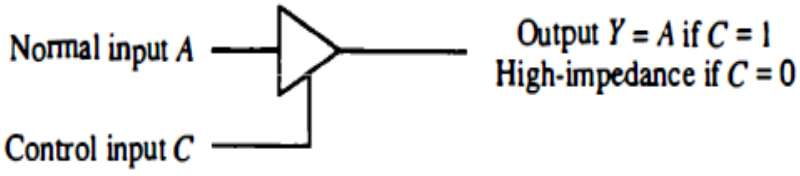


Fig. I-D : Graphic symbols for three- state buffer

- The control input determines the output state. When the control input is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input. When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.

NOTE: Q: What is the use of Tri-state buffer? Or what is the application of Tri-state Buffer?
Answer: To designing the Common bus system

Construction of a bus system with three-state buffers:

The construction of a bus system with three-state buffers is demonstrated in Fig. I-E.

- The outputs of four buffers are connected together to form a single bus line. (It must be realized that this type of connection cannot be done with gates that do not have three-state outputs.)
- 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.
- The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high -impedance state

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 as shown in Fig. I-E.

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. Only one decoder is necessary to select between the four registers

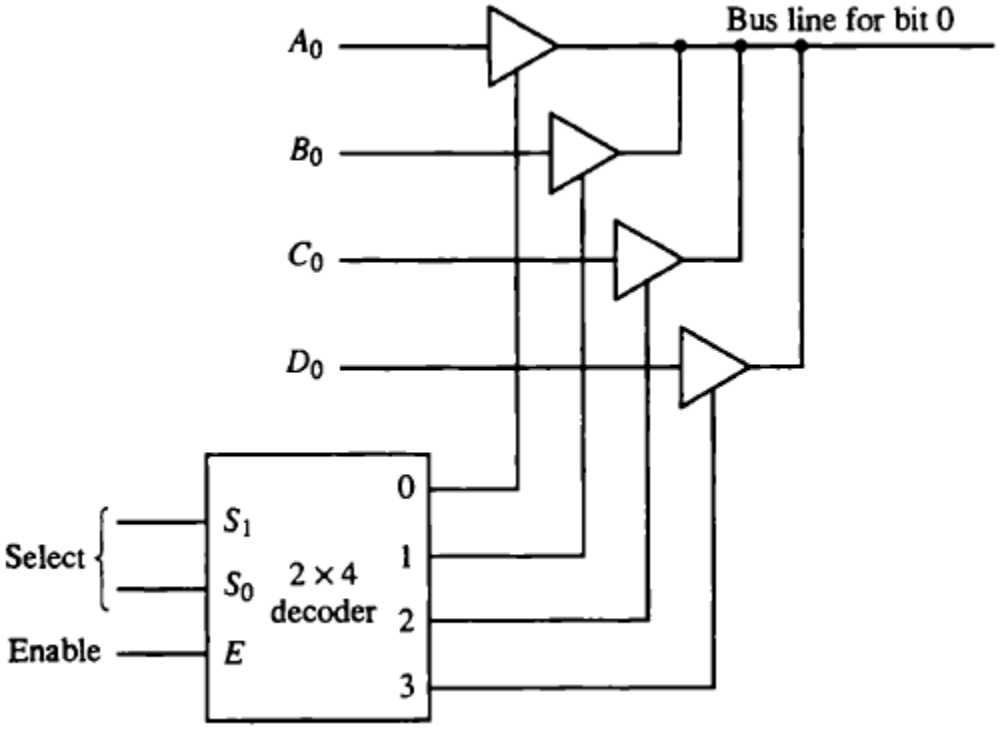
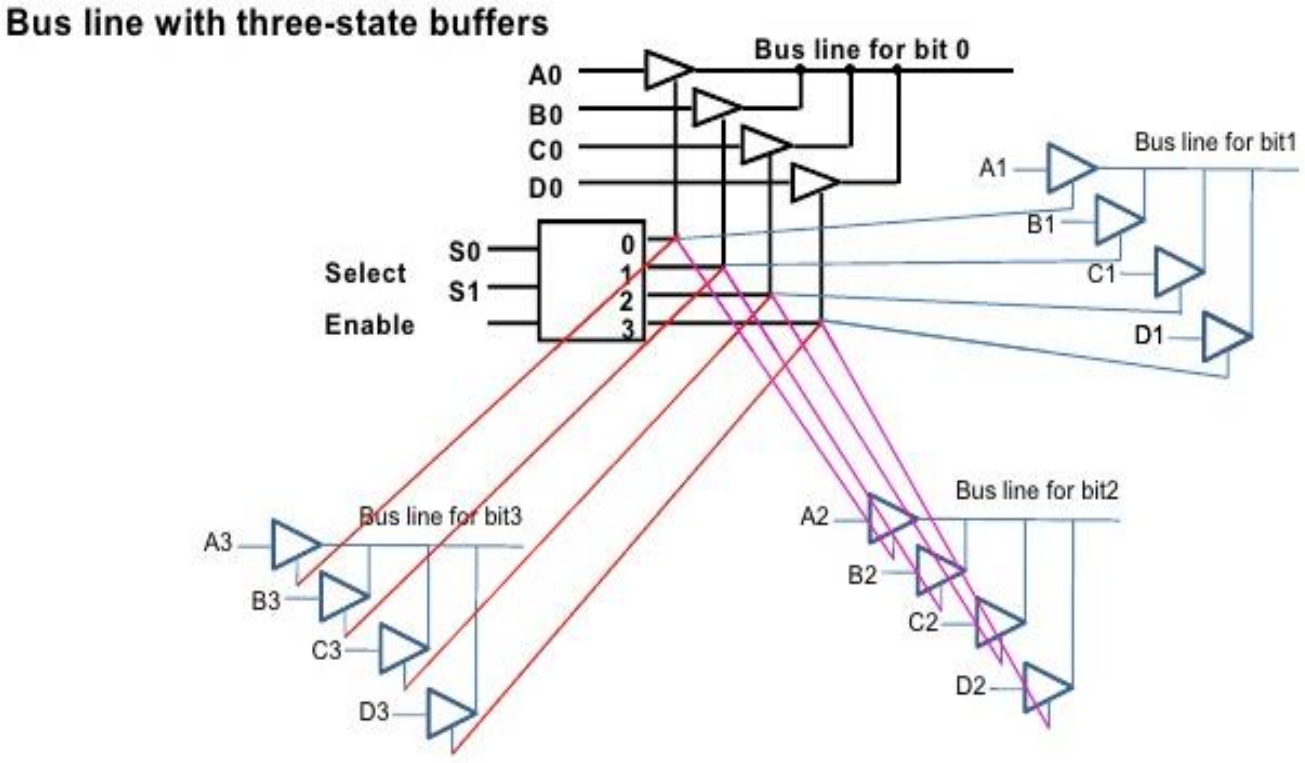


Figure I-E: Bus line with three state-buffers. (SINGLE BIT)

NOTE: Here E in decoder is to enable the decoder if E=1 or if E=0 it is disable.

- Select lines are input lines (S0S1) are to select one of the four outputs
- Here if you observe We are considering four registers with 4-bit each(i.e A0,A1,A2,A3 and B0.B1,B2,B3 and C0,C1,C2,C3 and D0,D1,D2,D3).
- In above figure we are only representing 0 bit (A0,B0,C0,D0) of all four registers similarly we can represents All 1 bit (A1,B1,C1,D1) of four register, and so on.

Here the figure showing the complete four bits of four register with three state buffer is



Arithmetic Micro operations:

In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.

- **Add Micro-Operation :**It is defined by the following statement:

  R3← R1 + R2

The above statement instructs the contents of register R1 and content of register R2 should be transferred to register R3.

- **Subtract Micro-Operation :**Let us again take an example:

  R3← R1- R2

  R3 ← R1 + R2' + 1 (2's complement)

In subtract micro-operation, instead of using minus operator we take 1's compliment and add 1 to the register which gets subtracted, i.e R1 - R2 is equivalent to R3 ← R1 + R2' + 1.

The following table shows the symbolic representation of various Arithmetic Micro-operations.

Symbolic Representation	Description
R3 ← R1 + R2	The contents of R1 plus R2 are transferred to R3.
R3 ← R1 - R2	The contents of R1 minus R2 are transferred to R3.
R2 ← R2'	Complement the contents of R2 (1's complement)
R2 ← R2' + 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2' + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one

Binary adder:

- The digital circuit that generates the arithmetic sum of two binary numbers of any length is called a binary adder.
- 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 I-F shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.

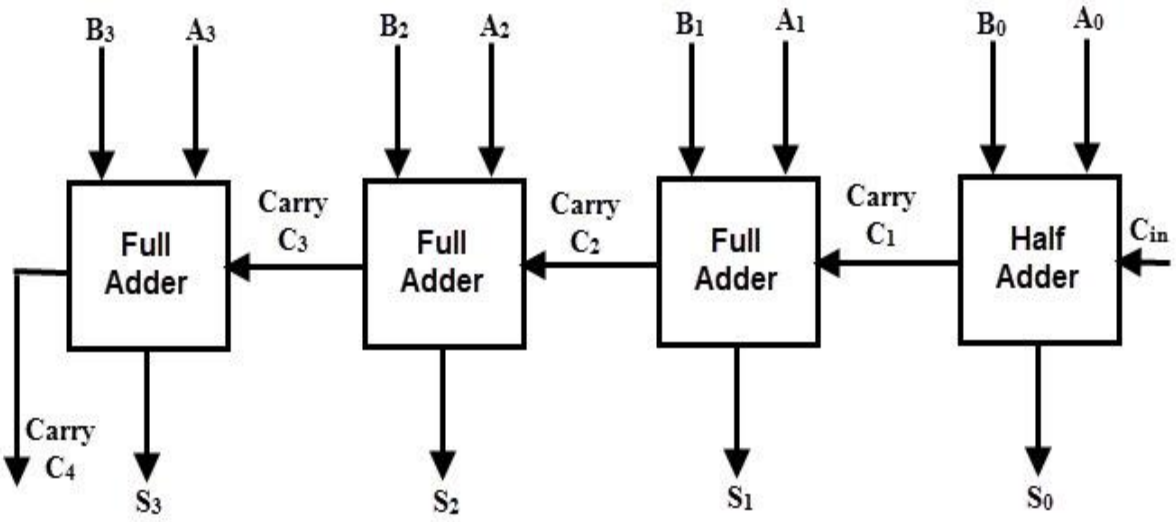


Figure I-F: 4-bit binary adder.

- The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders. The input carry to the binary adder is C_0 and the output carry is C_4. The S outputs of the full-adders generate the required sum bits.

Numerical Example:

Let us consider two 4-bit binary adder of two registers

R1 --- > 1 1 0 1 ---------- >augmend

R2 ------> 0 1 1 1---------- > addend

We need to perform addition of both R1+R2. i.e

- Example

C		1	1	1	0
R1		1	1	0	1
R2	+	0	1	1	1
Sum	1	0	1	0	0

- An n-bit binary adder requires n full-adders.

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.

> **NOTE:** Remember that the subtraction A – B can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1' s complement and adding one to the least significant pair of bits. The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry.

- A 4-bit adder-subtractor circuit is shown in Fig. I-G.

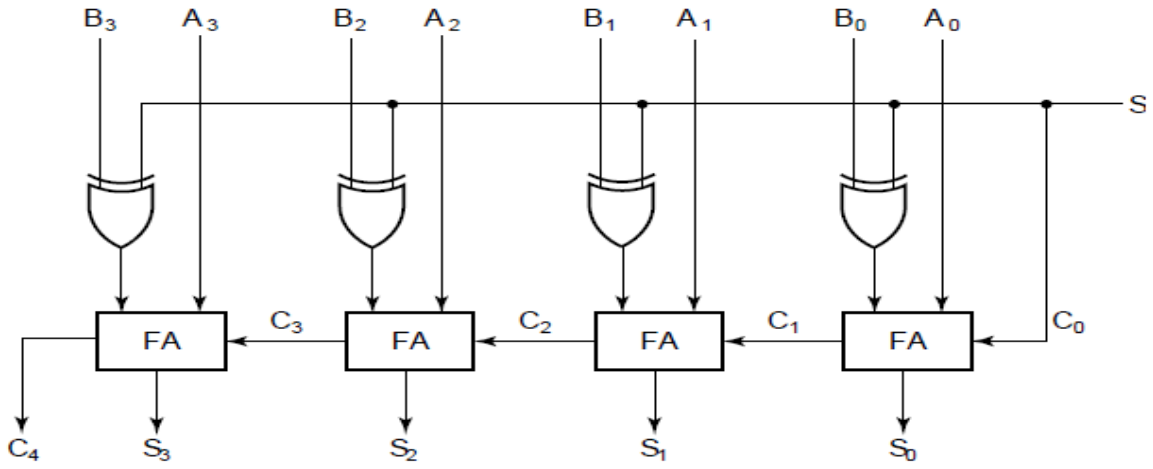


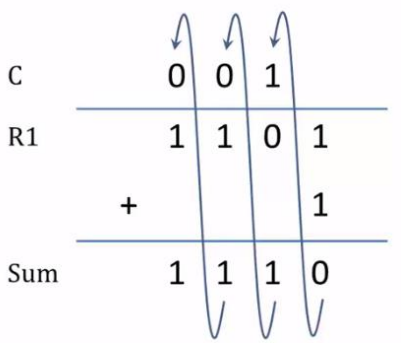
Figure I-G:4-bit adder-subtractor

The mode input S controls the operation. When S = 0 the circuit is an adder and when S = 1 the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B.

- **When S = 0**, we have B ⊕ 0 = B. The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
- **When S = 1**, we have B ⊕ 1 = B' and C_0 = 1. The B inputs are all complemented and a 1 is added through the input carry .The circuit performs the operation A plus the 2's complement of B.

Binary lncrementer:

- The increment micro operation adds one to a number in a register.
- **For example,** if a 4-bit register has a binary value 1101, it will go to 1110 after it is incremented.



- The diagram of a 4-bit combinational circuit incrementer is shown in Fig. 1-H.
- One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented
- 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 in S_0 through S_3.

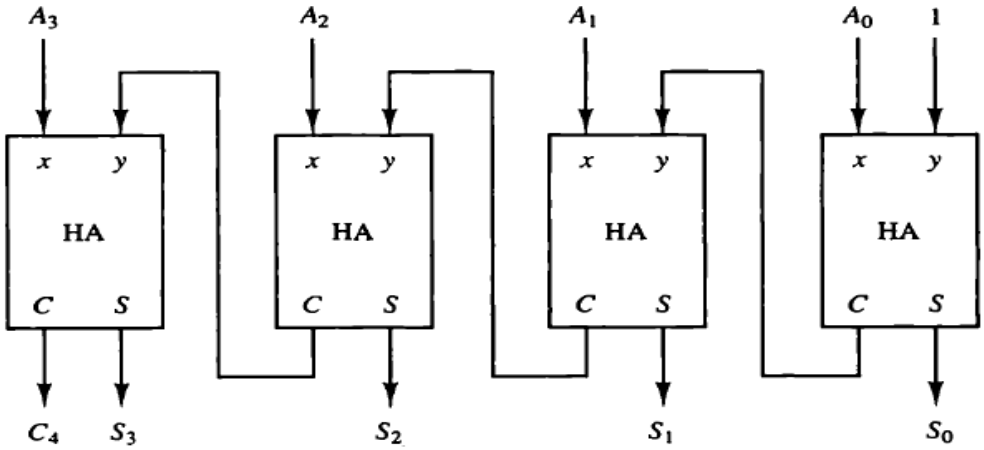


Figure I-H: 4-bit binary incrementer.

NOTE: For n-bit binary incrementer by extending the diagram to include n half-adders

Arithmetic Circuit:

- The arithmetic microoperations can be implemented in one composite 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 shown in Fig. I-I**. 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 binary adder.
- Each of the four inputs from B are connected to the data inputs of the multiplexers(i.e 0 input bit).
- The multiplexers data inputs also receive the complement of B(i.e 1 input bit) .
- The other two data inputs (i.e input bits 2 and 3 of the each multiplexer are connected to logic-0 and logic-1.
- The four multiplexers are controlled by two selection inputs, S1 and S0.
- 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.

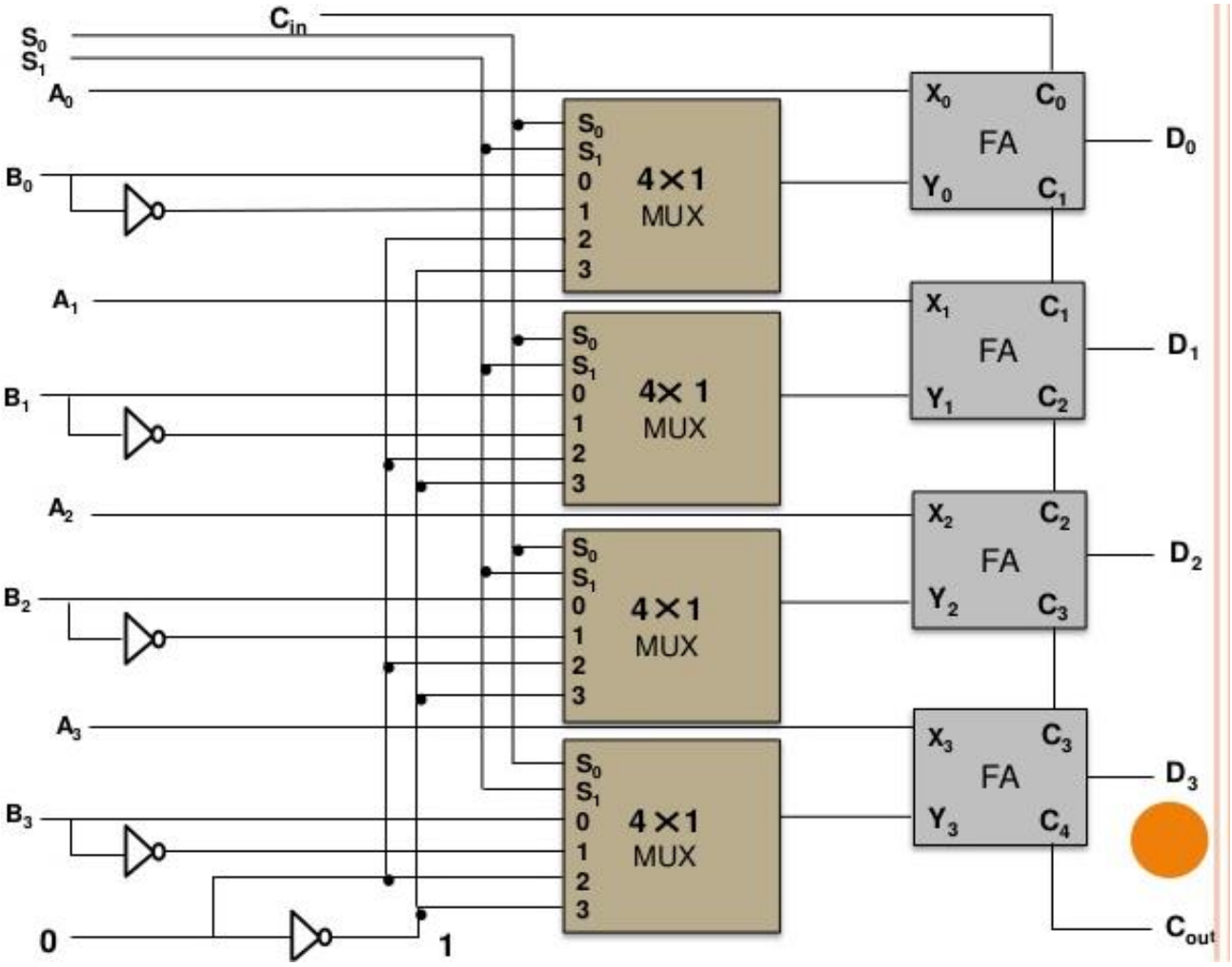


Figure I-I: 4-bit arithmetic circuit

- 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
- → 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.

- Here some are possible micro arithmetic operations listed below **Table I-J**

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 S1S0 = 00, the value of B is applied to the Y inputs of the adder.
  - ✓ If Cin = 0, the output D = A + B . If Cin = 1, output D = A + B + l.
  - ✓ Both cases perform the add micro operation with or without adding the input carry.
- When S1S0 = 01, the complement of B is applied to the Y inputs of the adder.
  - ✓ If Cin = 1, then D = A + $\overline{B}$ + 1. This produces A plus the 2's complement of B, which is equivalent to a subtraction of A - B.
  - ✓ If Cin = 0, then D = A + $\overline{B}$. This is equivalent to a subtract with borrow.
- When S1S0 = 10, the inputs from B are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes D = A + 0 + Cin·
  - ✓ If Cin=0, then D = A. Here we have a direct transfer from input A to output D.
  - ✓ If Cin = 1, then D = A + 1. Here we have the value of A is incremented by 1.
- When S1So = 11, all 1' s are inserted into the Y inputs of the adder .
  - ✓ If Cin=0, then it produce the decrement operation D = A - 1 .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 D = A + 2's complement of 1 = A - 1.
  - ✓ If Cin = 1, then D = A - 1 + 1 = A, which causes a direct transfer from input A to output D.

Logic micro operations

- 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 micro operation with the contents of two registers R 1 and R2 is symbolized by the statement.

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

It specifies a logic micro operation 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
  - ✓ Let the content of R2 be 1100.
  - ✓ The exclusive-OR micro operation stated above symbolizes the following logic computation:

1010 Content of R 1

1 100 Content of R2

0110 Content of R 1 after P = 1

- There are 16 different logic operations that can be performed with two binary variables.
- The 16 Boolean functions o f two variables x and y are expressed in algebraic form in the first column of Table I-K.

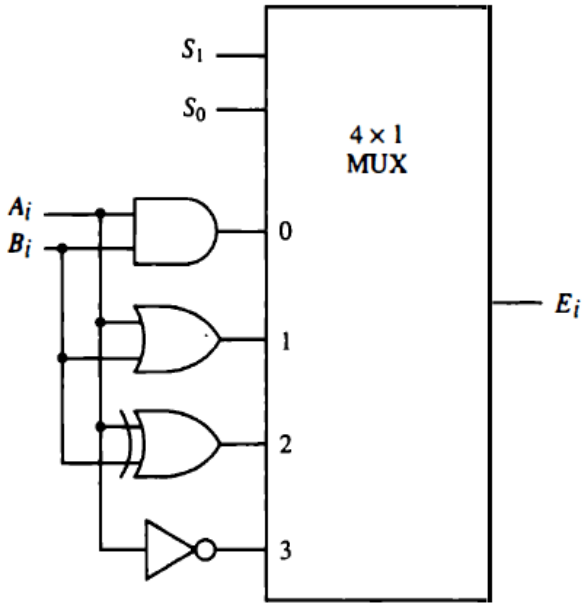
TABLE I-K: Sixteen Logic Microoperations

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

- The 16 logic micro operations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B.
- It is important to realize that the Boolean functions listed in the first column of Table I-K represents relationship between two binary variables x and y.
- The logic micro operations listed in the second column represent a relationship between the binary content of two registers A and B

Hardware Implementation:

- The hardware implementation of logic micro operations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
- Figure I-L shows one stage of a circuit that generates the four basic logic micro operations (AND, OR, XOR (exclusive-OR), and complement). It consists of four gates and a multiplexer.
- Each of the four logic operations is generated through a gate that performs the required logic. The outputs of the gates are applied to the data inputs of the multiplexer.
- The two selection inputs S1 and S0 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 I-L: One stage of logic circuit

Applications of Logic Micro operations:

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.

   Example:

   ```
   1010      A before
   1100      B (logic operand)
   ------------------------
   1110      A after
   ------------------------
   ```

   - The two leftmost bits of B are 1's, so the corresponding bits of A are set to 1 . The two bits of A with corresponding 0's in B remain unchanged.
   - From the truth table we note that the bits of A after the operation are obtained from the logic-OR operation of bits in B and previous values of A .
   - Therefore, the **OR micro operation** can be used to selectively set bits of a register.

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 .

   Example:

   ```
   1010      A before
   1100      B (logic operand)
   ------------------------------------
   0110      A after
   ------------------------------------
   ```

   - Again the two leftmost bits of B are 1's, so the corresponding bits of A are complemented.
   - This example again can serve as a truth table from which one can deduce that the selective-complement operation is just an exclusive-OR micro operation.
   - Therefore, the **exclusive-OR micro operation** can be used to selectively complement bits of a register.

3. **selective-clear**

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

Example:

1010	A before
1 100	B (logic operand)
---	---
0010	A after
---	---

- Again the two leftmost bits of B are 1' s, so the corresponding bits of A are cleared to 0. One can deduce that the Boolean operation performed on the individual bits is AB' .
- The corresponding logic micro operation is $\mathbf{A \leftarrow A \wedge \bar{B}}$

4. Mask operation:

- The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in 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
---	---

- The two rightmost bits of A are cleared because the corresponding bits of B are 0's. The two leftmost bits are left unchanged because the corresponding bits of B are 1's.
- The mask operation is more convenient to use than the selective clear operation because most computers provide an AND instruction, and few provide an instruction that executes the micro operation for selective-clear.

5. Insert operation:

- 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, suppose that an A register contains eight bits, 0110 1010.

To replace the four leftmost bits by the value 1001 we first mask the four unwanted bits:

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

and then insert the new value:

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

- The mask operation is an AND micro operation and the insert operation is an OR micro operation.

6. Clear operation:

- 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 micro operation** as shown by the following Example:

Example:

```
1010          A
1010          B
-------------------------------
0000          A ← A ⊕ B
-------------------------------
```

- When A and B are equal, the two corresponding bits are either both 0 or both 1. In either case the exclusive-OR operation produces a 0.
- The all-0's result is then checked to determine if the two numbers were equal.

Shift micro operations:

- Shift micro operations 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. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input.
  - ✓ During a shift-left operation the serial input transfers a bit into the rightmost position.
  - ✓ During a shift-right operation the serial input transfers a bit into the leftmost position.
- The information transferred through the serial input determines the type of shift.

There are three types of shifts: **logical, circular, and arithmetic.**

1. logical shift:

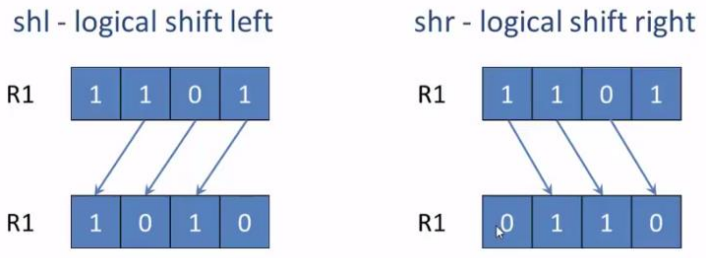
- A logical shift is one that transfers 0 through the serial input. We will adopt the symbols shl and shr for logical shift-left and shift-right micro operations.

For example:

R1 ← shl R1

R2 ← shr R2

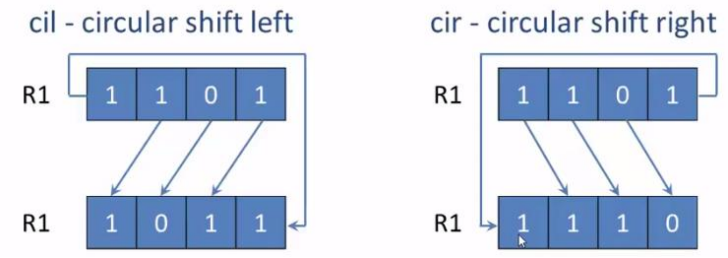
are two micro operations that specify a 1-bit shift to the left of the content of register R1 and a 1-bit shift to the right of the content of register R2.



- The bit transferred to the end position through the serial input **is assumed to be 0 during a logical shift.**

2. Circular shift:

- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input.
- We will use the symbols **cil** and **cir** for the circular shift left and right, respectively.



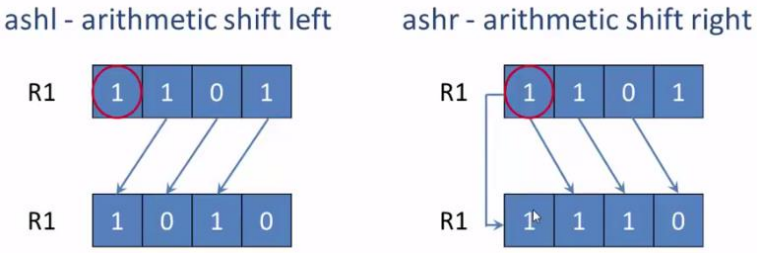
- The symbolic notation for the shift micro operations is shown in Table I-.M

TABLE I-M: Shift Micro operations

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

3. Arithmetic shift:

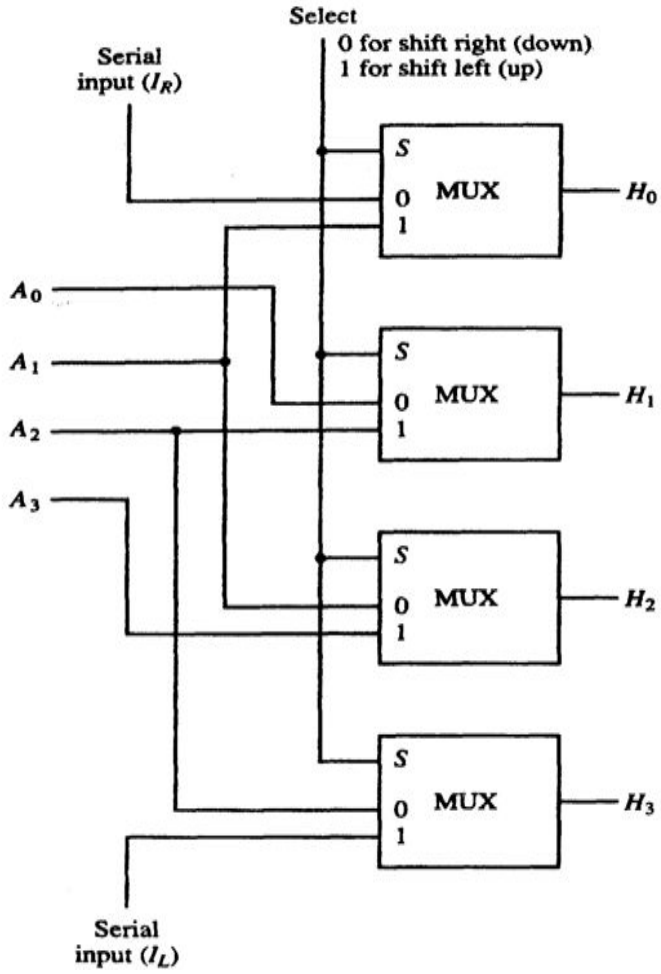
- An arithmetic shift is a micro operation that shifts a signed binary number to the left or right.
  - ✓ An arithmetic shift-left multiplies a signed binary number by 2.
  - ✓ An arithmetic shift-right divides the number by 2.
- Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same.



Hardware Implementation:

- A combinational circuit shifter can be constructed with multiplexers as shown in Fig. I-N. The 4-bit shifter has four data inputs, A0 through A3, and four data outputs, H0 through H3.
- 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 function table in Fig. I-N shows which input goes to each output after the shift. A shifter with n data inputs and outputs requires n multiplexers.
- The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.



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 I-N: 4-bit combinational circuit shifter

- 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 function table in Fig. I-N shows which input goes to each output after the shift. A shifter with n data inputs and outputs requires n multiplexers.
- The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts

Arithmetic Logic Shift Unit:

- Instead of having individual registers performing the micro operations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated **ALU**.
- The shift micro operations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU. The arithmetic logic shift unit is shown in Fig. I-O

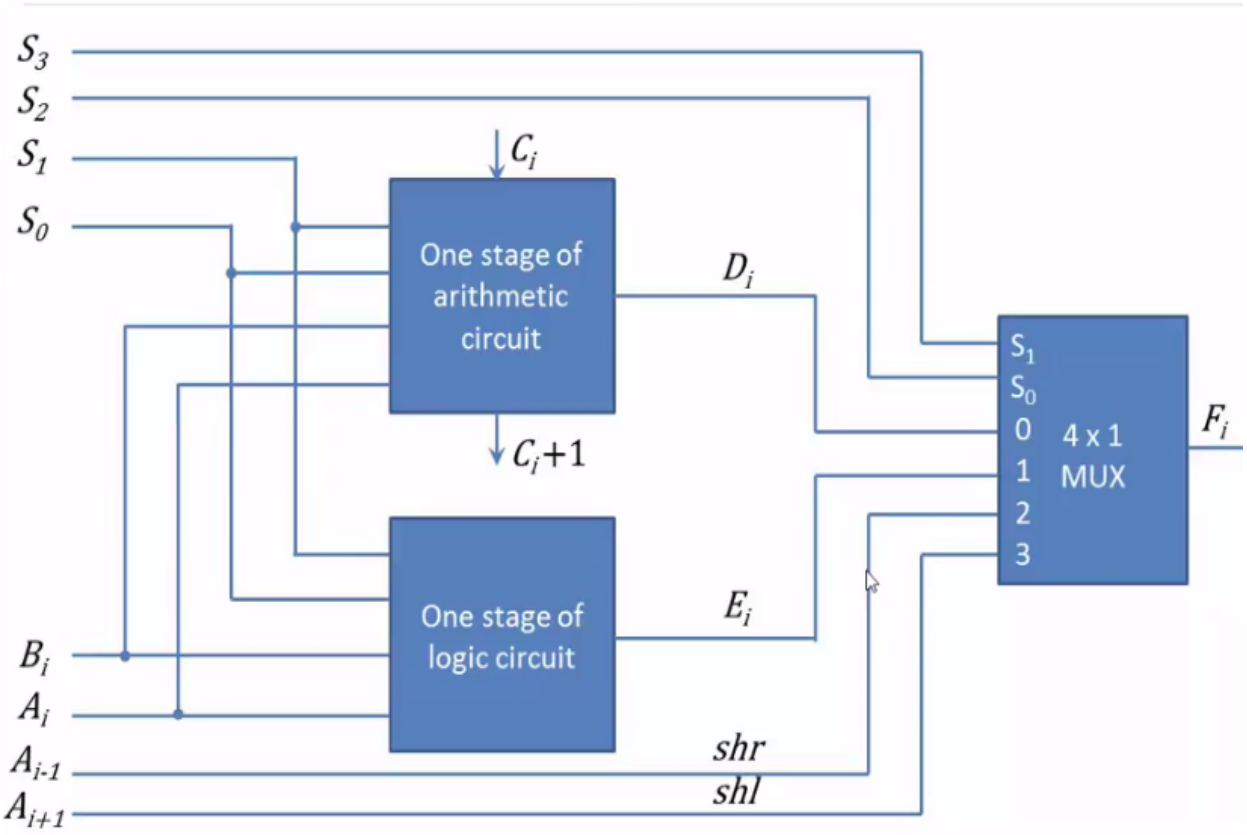


Figure 1-O: One stage of an arithmetic logic shift unit

- The subscript **i** designates a typical stage. Inputs $\mathbf{A_i}$ and $\mathbf{B_i}$ are applied to both the arithmetic and logic units.
- A particular micro operation is selected with inputs S1 and S0. 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 S3 and S2. The other two data inputs to the multiplexer receive inputs $\mathbf{A_{i-1}}$ for the shift-right operation and $\mathbf{A_{i+1}}$ for the shift-left operation.

- The circuit of Fig. I-O must be repeated n times for an n-bit ALU. The output carry C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence.
- The input carry to the first stage is the input carry C_{in}, which provides a selection variable for the arithmetic operations.
- The circuit whose one stage is specified in Fig. I-O provides **eight arithmetic operation, four logic operations, and two shift operations.**
- Each operation is selected with the five variables S3, S2, S1, S0, and C_{in}. The input carry C_{in} is used for selecting an arithmetic operation only.
- Table I-P lists the 14 operations of the ALU. The first eight are arithmetic operations (see Table I-J) and are selected with S3S2 = 00. The next four are logic operations (see Fig. I-K) and are selected with S3S2 = 01.
- The input carry has no effect during the logic operations and is marked with don't-care x ' s .
- The last two operations are shift operations and are selected with S3S2 = 10 and 11 . The other three selection inputs have no effect on the shift.

TABLE I-P: 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

Instruction Codes

- The internal organization of a digital system is defined by the sequence of micro operations it performs on data stored in its registers.
- The user of a computer can control the process by means of a program. A program is a set of instructions that specify the operations, operations operands, and the sequence by which processing has to occur.

- A computer instruction is a binary code that specifies a sequence of micro operations for the computer. Instruction codes together with data are stored in memory.
- The computer reads each instruction from memory and places it in a control register.
- The control then interprets the binary code of the instructions and proceeds to execute it by issuing a sequence of micro operations.
- Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general-purpose computer.

- An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually **divided into parts**, each having its **own particular interpretation**.
- The most basic part of an instruction code is its **operation part**. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
- The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.
- The operation code must consists of at least n bits for a given 2" (or less) distinct operations.
- **As an illustration, consider** a computer with 64 distinct operations, one of them being an ADD operation. The operation code consists of six bits, with a bit configuration 110010 assigned to the ADD operation.
- When this operation code is decoded in the control unit, the computer issues control signals to read an operand from memory and add the operand to a processor register.
- At this point we must recognize the relationship between a computer operation and a micro operation.
- An operation is part of an instruction stored in computer memory. It is a binary code tells the computer to perform a specific operation.
- The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control signals to initiate micro operations in internal computer registers.

- For every operation come, the control issues a sequence of micro operations needed for the hardware implementation of the specified operation.
- For this reason, an operation code is sometimes called a micro operation because it specifies a set of micro operations.

The operation part of an instruction code specifies the operation to be performed.

- This operation must be performed on some data stored in processor registers or in memory.
- An instruction code must therefore specify not only the operation but also the registers or the memory words where the operands are to be found, as well as the register or memory word where the result is to be stored.
- **Memory words can be specified in instruction codes by their address**.
- Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2k registers. There are many variations for arranging the binary code of instructions, and each computer has its own particular **instruction code format**.
- Instruction code formats are conceived computer designers who specify the architecture of the computer.

Stored Program Organization:

- The simplest way to organize a computer is to have one processor register (Accumulator AC) and instruction code format with two parts.
  - The first part specifies the operation to be performed.
  - 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 1-AA** 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 op code) 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.

- 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.

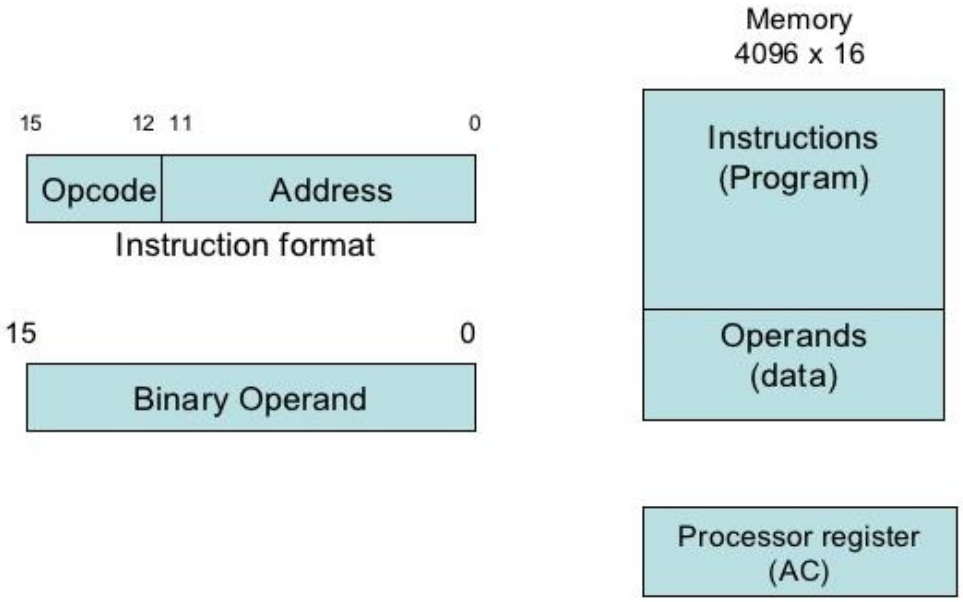


Figure 1-AA: Stored Program Organization

NOTE:

- 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 of an instruction code specifies the address of an operand, the instruction is said to have a **direct address**.
- When the second part of an instruction code specifies an address of a memory word in which the address of the operand is found. It is called **indirect address.**
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.

As an illustration, consider the instruction code format shown in Fig. 1-BB(a).

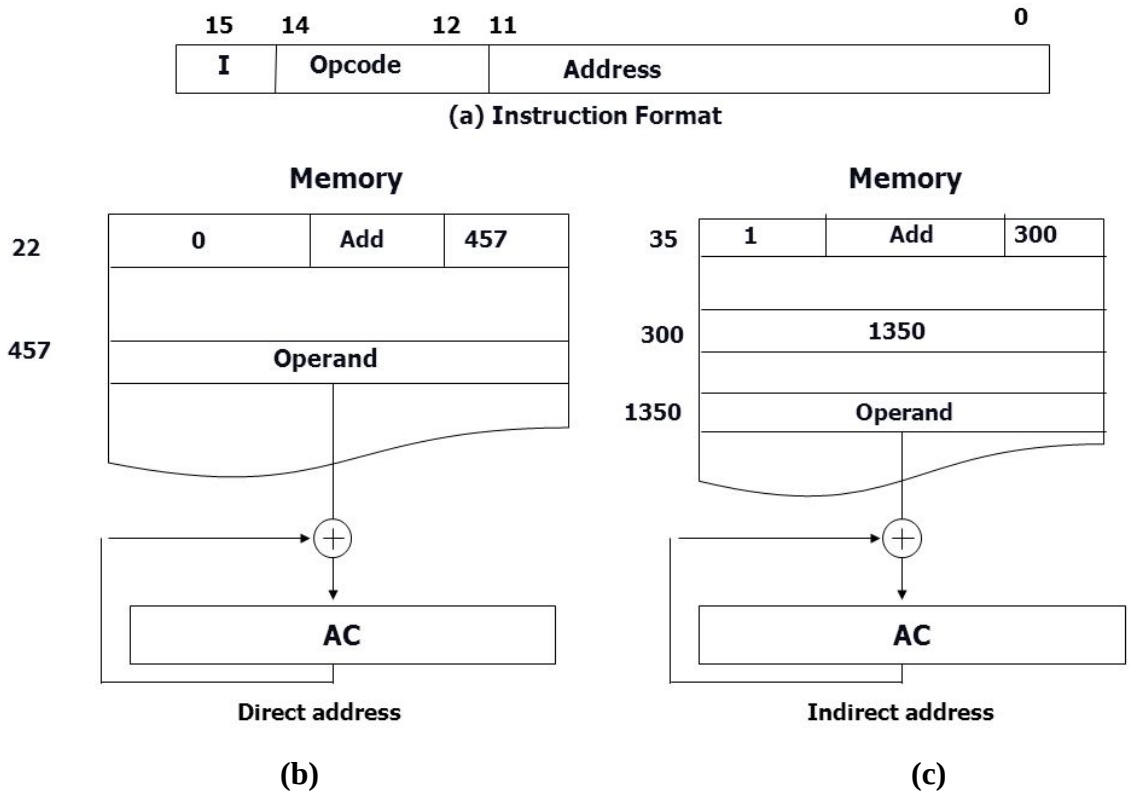


Figure-1-BB: Demonstration of direct and indirect addresses

- It consists of a 3-bit operation code, 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. 1-B(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 op code 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.

➢ A indirect address instruction is shown in Fig. 1-B(c) The instruction in address 35

- The mode bit I = 1. Therefore, it is recognized as an indirect address instruction.
- The address part is the binary equivalent of 300. The control gives 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.

Computer Registers

- Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time.
- The **control** reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it, and so on.
- This type of instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed.
- It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory.
- The computer needs processor registers for manipulating data and a register for holding a memory address. The registers are listed in Table 1-1

Register Symbol	Register Name	Number of Bits	Description
AC	Accumulator	16	Processor Register
DR	Data Register	16	Hold memory data
TR	Temporary Register	16	Holds temporary Data
IR	Instruction Register	16	Holds Instruction Code
AR	Address Register	12	Holds memory address
PC	Program Counter	12	Holds address of next instruction
INPR	Input Register	8	Holds Input data
OUTR	Output Register	8	Holds Output data

Table 1-1

- The memory unit has a capacity of 4096 words and each word contains 16 bits.
- Twelve bits of an instruction word are needed to specify the address of an operand. This leaves three bits for the operation part of the instruction and a bit to specify a direct or indirect address.
  - The data register (DR) holds the operand read from memory.
  - The accumulator (AC) register is a general purpose processing register.
  - The instruction read form memory is placed in the instruction register (IR).
  - The temporary register (TR) is used for holding 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 address of the next instruction to be read from memory after the current the instruction is executed.

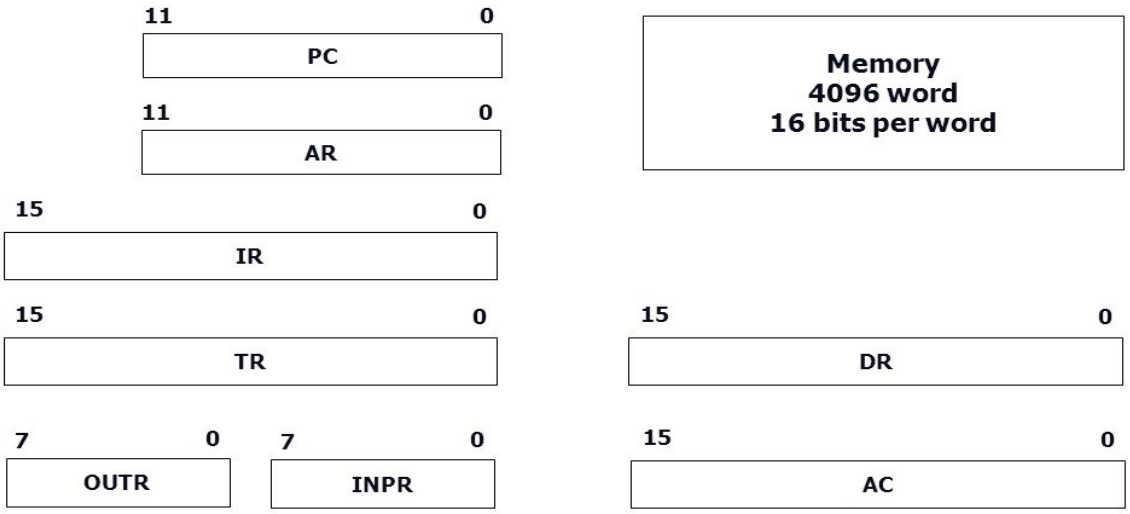


Figure 1-CC: Basic computer registers and memory.

- 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 eight 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.
- It is known that how to construct a bus system using multiplexers or three-state buffer gates.
- The connection of the registers and memory of the basic computer to a common bus system is shown in Fig. 1-DD.

- 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_2 S_1 S_0 = 011** since this is the binary value of decimal 3.

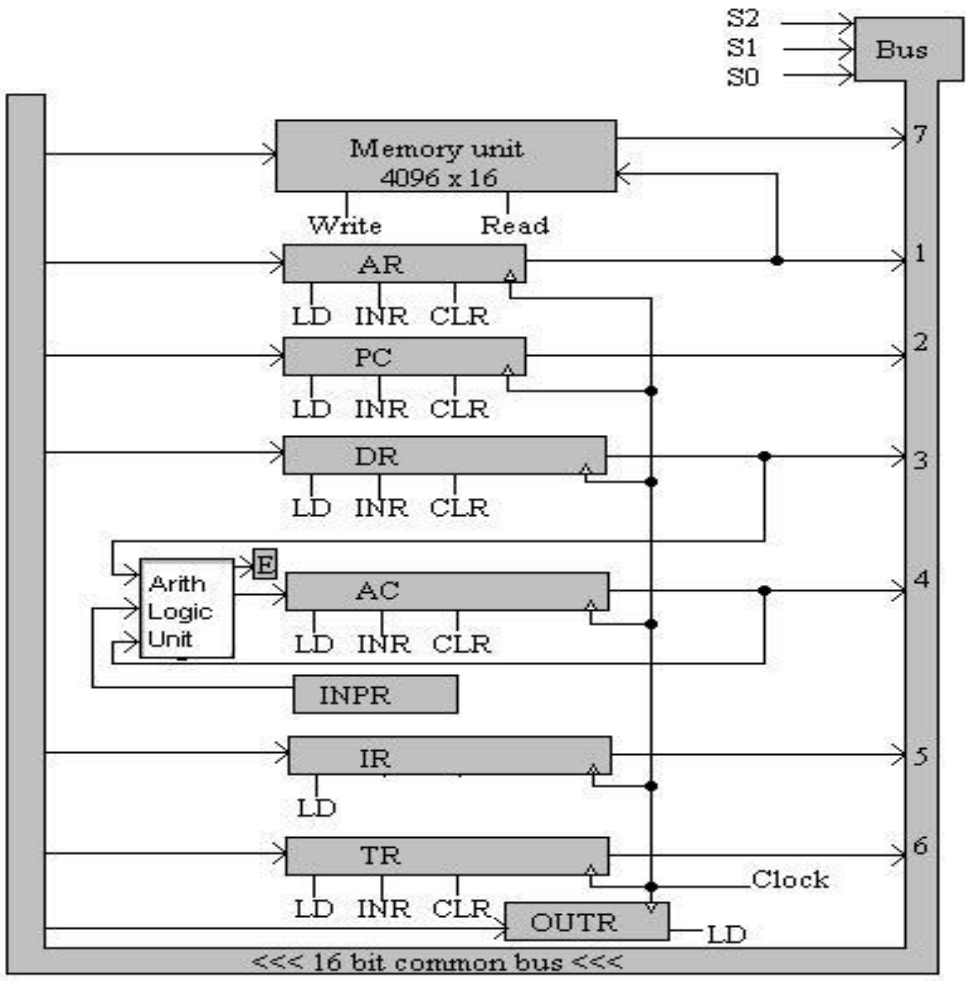


Figure 1-DD: Basic registers which are connected to a common bus

- 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 transition.
- The memory receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and $S_2\ S_1\ S_0 = 111$.
- 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 (LSB) in the bus.

- INPR is connected to provide information to the bus but OUTR can only receive information from the bus. This is because INPR receives a character from an input device which is then transferred to AC. OUTR receives a character from AC and delivers it to an output device.
- The 16 lines of the common bus receive information from six registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory.
- Five registers have three control inputs: LD (load), INR (increment) and CLR (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. Therefore, AR must always be used to specify a memory address.
- The content of any register can be specified for the memory data input during a write operation. Similarly, any register can receive the data from memory after a read operation except AC.
- 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 micro-operations 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 micro-operations, 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.

  - **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 micro-operations: **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 S2S1S0 = 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.

Computer Instructions

- The basic computer has three instruction code formats. Each format has 16 bits. The **operation code** (opcode) part of the instruction contains **three bits** and 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.
- **An input-output instruction** does not need a reference to memory and is recognized by the operation code 111 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.

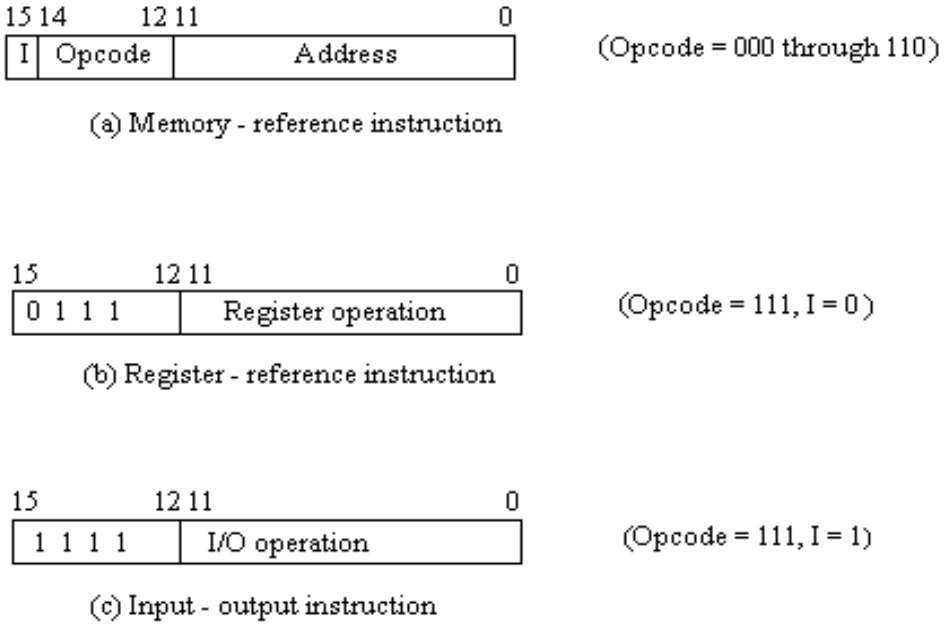


Figure 1-E: Basic computer instruction formats.

- 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 op code bits in positions 12 through 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.

- If the three op code bits in positions 12 through 14 are equal to 111 , control then inspects the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input-output type.

NOTE: Note that the bit in position 15 of the instruction code is designated by the symbol **I** but is not used as a mode bit when the operation code is equal to 111.

- Only three bits of the instruction are used for the operation code. It may seem that the computer is restricted to a maximum of eight distinct operations. However, since register reference and input-output instructions use the remaining 12 bits as part of the operation code, the total number of instruction chosen for the basic computer is equal to 25.

TABLE 1-2: Basic Computer Instructions

Symbol	Hex Code I = 0	Hex Code I = 1	Description	
AND	0xxx	8xxx	AND memory word to AC	Memory reference instructions
ADD	1xxx	9xxx	Add memory word to AC	
LDA	2xxx	Axxx	Load AC from memory	
STA	3xxx	Bxxx	Store content of AC into 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	Register-reference instructions
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 instr. if AC is positive	
SNA	7008		Skip next instr. if AC is negative	
SZA	7004		Skip next instr. if AC is zero	
SZE	7002		Skip next instr. if E is zero	
HLT	7001		Halt computer	
INP	F800		Input character to AC	Input-output instructions
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	

- The instructions for the computer are listed in Table 1-2. The symbol designation is a three letter word and represents an abbreviation intended for programmers and users. The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.
- A **memory-reference instruction** has an address part of 12 bits. The address part is denoted by three x's and stand for the three hexadecimal digits corresponding to the 12-bit address.
- The last bit of the instruction is designated by the symbol **I**.

- When I = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is 0. When I = 1, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E since the last bit is I.
- **Register-reference instructions** use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7. The other three hexadecimal digits give the binary equivalent of the remaining 12 bits.
- The **input-output instructions** also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

Timing and Control

→ The timing for all registers in the basic computer is controlled by a master clock generator.

→ The clock pulses do not change the state of a register unless the register is enabled by a control signal.

→ The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro operations or the accumulator.

→ There are two major types of control organization:

Hardwired control and **micro programmed control**.

→ **In the hardwired organization**, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation.

→ **In the micro programmed organization**, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations.

Block diagram of control unit:

- The block diagram of the control unit is shown in Fig. 1-FF. It consists of two decoders, a sequence counter, and a number of control logic gates.
- An instruction read from memory is placed in the instruction register (IR): .The instruction register is shown again in Fig. 1-FF where it is divided into three parts:

  The I bit, the operation code, and bits 0 through 11.

- The operation code in bits 12 through 14 are decoded with a 3 × 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code.
- Bits 15 of the instruction is transferred to a flip-flop designated by the symbol **I**. Bits 0 through 11 are applied to the control logic gates.

- The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15.

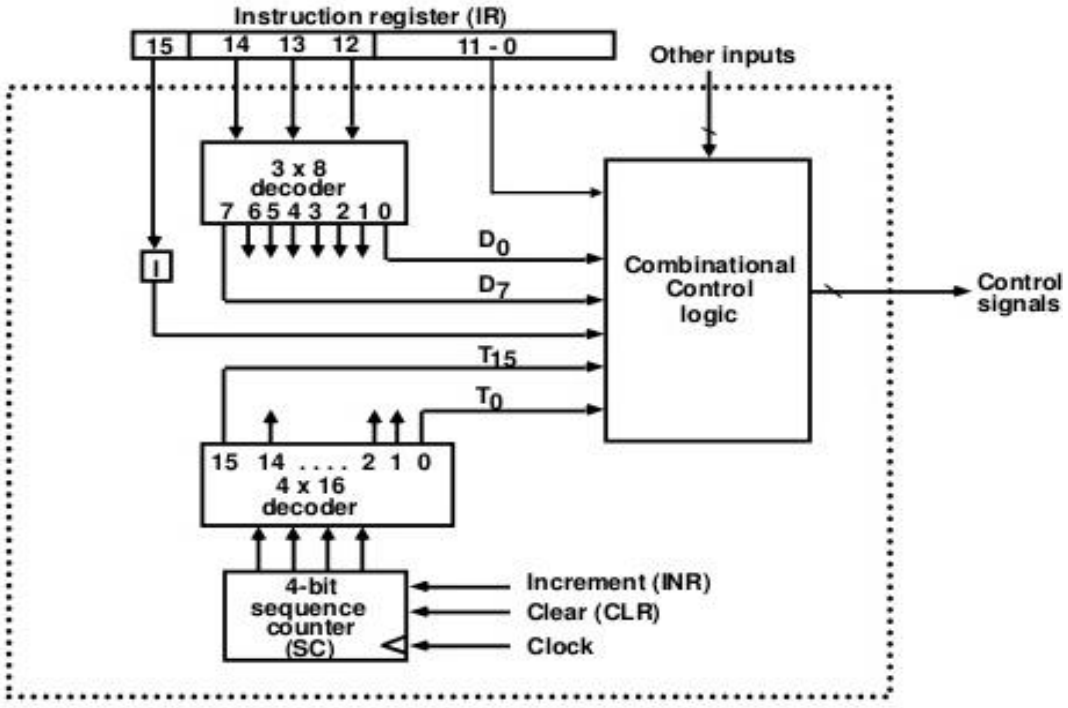


Figure 1-FF: Control unit basic computer

- The **sequence counter** SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 × 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be T_0.
- **As an example**, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3, and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement:

$$D_3\,T_4: SC \leftarrow 0$$

The Timing diagram:

- The timing diagram of Fig. 1-GG shows the time relationship of the control signals. The sequence counter SC responds to the positive transition of the clock.
- Initially, the CLR input of SC is active. The first positive transition of the Clock clears SC to 0, which in turn activates the timing signal T_0 out of the decoder.
- T_0 is active during one clock cycle. The positive clock transition labeled T_0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T_0.
- SC is incremented with every positive clock transition, unless its CLR input is active. This produces the sequence of timing signals T_0, T_1, T_2, T_3, T_4, and so on, as shown in the diagram (Note the

relationship between the timing signal and its corresponding positive clock transition.) If SC is not cleared, the timing signals will continue with T_5, T_6, up to T_{15} and back to T_0.

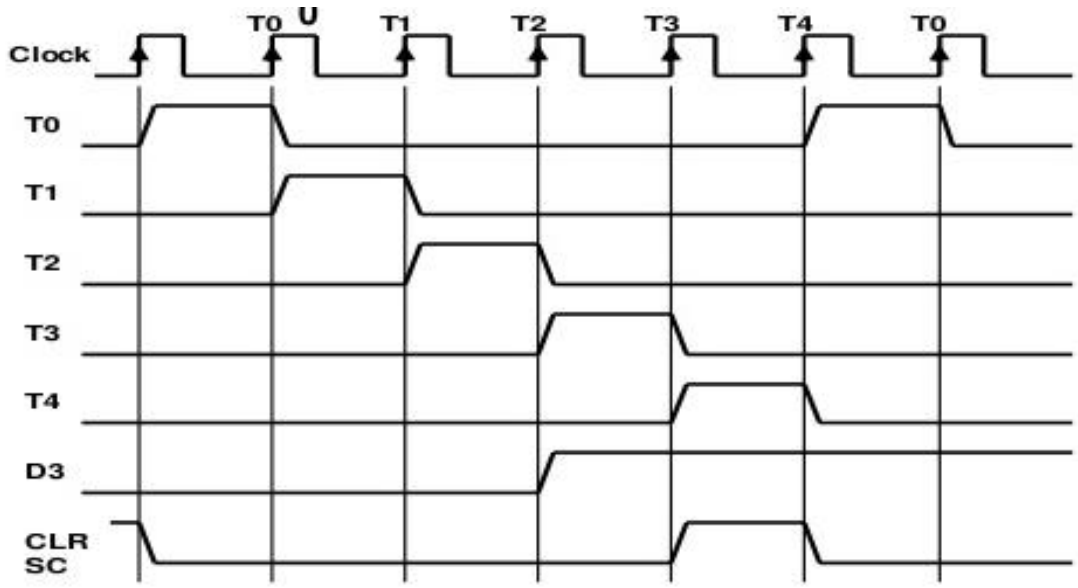


Figure 1-GG: Example of control timing signals

- The last three waveforms in Fig 1-GG show how SC is cleared when $D_3T_4 = 1$. Output D_3 from the operation decoder becomes active at the end of timing signal T_2.
- When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3D_4 becomes active.
- This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T_4 in the diagram) the counter is cleared to 0.
- This causes the timing signal T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

Instruction Cycle:

- In the basic computer each instruction cycle consists or 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.

❖ Upon 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.

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 T0.
- After each clock pulse, SC is incremented by one so that the timing signals go through a sequence T0, T1, T2, and so on.
- The micro operations for the fetch and decode phases can be specified by the following register transfer statements.

T0: AR ← PC
T1: IR ← M[AR], PC ← PC + 1
T2: D0… D1 ← Decode IR (12-14), AR ← IR (0 −11), 1 ← IR (15)

- 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 T0**.
- The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing **signal T1**.
- At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.
- At time T2, 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.
- Note that SC is incremented after each clock pulse to produce the sequence T0, T1, and T2.

Figure 1-HH shows how 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 T0 to achieve the following connection:

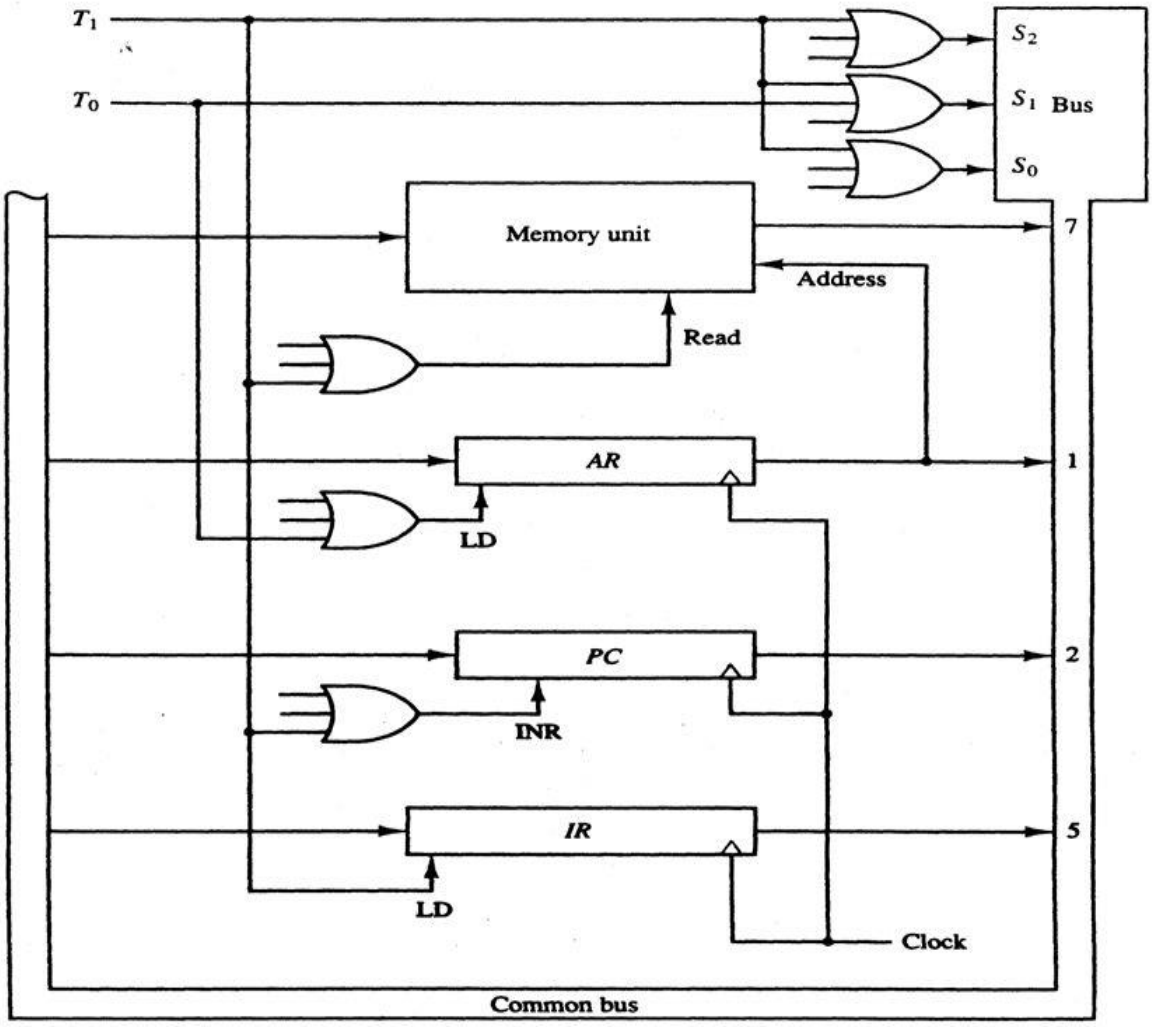


Figure 1-HH: Register transfers for the fetch phase

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR.

- The next clock transition initiates the transfer form PC to AR since T0 = 1. In order to implement the second statement

$$T1: IR \leftarrow M[AR], PC \leftarrow PC + 1$$

- It is necessary to use timing signal T1 to provide the following connections in the bus system.
  a) Enable the read input of memory.
  b) Place the content of memory onto the bus by making S2S1S0 = 111.
  c) Transfer the content of the bus to IR by enabling the LD input of IR.
  d) Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since T1 = 1.

Figure 1-HH duplicates a portion of the bus system and shows how T0 and T1 are connected to the control inputs of the registers, the memory, and the bus selection inputs.

Determine the Type of Instruction:

- Decoder output D_7 is equal to 1 if the operation code is equal to binary 111. We determine that if D_7 =1, the instruction must be a register-reference or input-output type.
- If D_7 = 0, the operation code must be one of the other seven values 000 through 110, specifying memory reference instruction.
- Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I.
- If D_7 = 0 and I = 1, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory.
- The micro operation for the indirect address condition can be symbolized by the register transfer statement

  $$AR \leftarrow M [AR]$$

- Initially, AR holds the address part of the instruction. This address is used during the memory read operation. The word at the address given by AR is read from memory and placed on the common bus. The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

❖ The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. This can be symbolized as follows:

D_7 = 0	I = 1	D'_7 IT3: AR← M [AR]
0	0	D'_7 I'T3; Nothing
1	0	D_7 I' T3: Execute a register-reference instruction
1	1	D_7 IT3: Execute an input-output instruction

❖ The flowchart of Fig. 1-II presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.

Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock transition. We will adopt the convention that if SC is incremented, we will not write the statement SC ← SC + 1, but it will be implied that the control goes to the next timing signal in sequence. When SC is to be cleared, we will include the statement Sc ← 0.

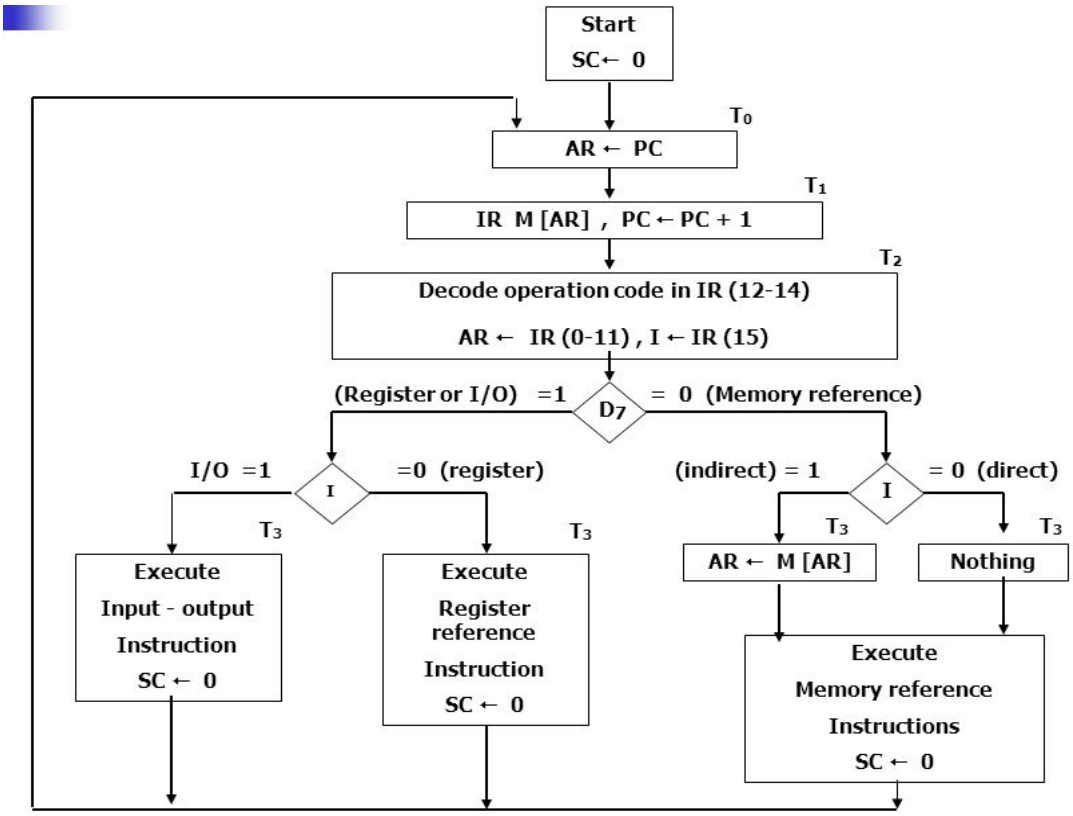


Figure 1-II : Flowchart for instruction cycle (initial configuration).

Register-Reference Instructions:

- Register-reference instructions are recognized by the control when D7 = 1 and I = 0.
- These instructions use bits 0 through 11 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 T2.

The control functions and micro operations for the register-reference instructions are listed in **Table 1- 3**

$D_7 I' T_3 = r$ (Common All Register Instruction)
IR(I)= Bi [0—11 that Specifies The Instruction]

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

- Each control function needs the Boolean relation D7 I' T3, 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 Bi to bit i of IR, all control functions can be simply denoted by rBi.
- **For example**, the instruction CLA has the hexadecimal code 7800 (see Table 3.1), which gives the binary equivalent 0111 1000 0000 0000.
- The first bit is a zero and is equivalent to I'. The next three bits constitute the operation code and are recognized from decoder output D7. Bit 11 in IR is 1 and is recognized from B11. The control function that initiates the micro operation for this instruction is D7 I' T3 B11 = rB11.
- The first seven register-reference instructions perform clear, complement, circular shift, and increment micro operations 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 **HLT** instruction clears a start-stop flip-flops S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

Memory-Reference Instructions:

- Table 1-4 lists the seven memory-reference instructions. The decoded output Di 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 T2 when I = 0, or during timing signal T3 when I = 1.
- The execution of the memory-reference instructions starts with timing signal T4 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 micro operations. 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 1-4: Memory-Reference Instructions

Symbol	Operation Decoder	Symbolic Description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
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 $PC \leftarrow PC+1$

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 micro operations that execute this instruction are:

D_0T_4 : DR ← M [AR]

D_0T_5: AC ← AC Λ DR, SC ← 0

- Two timing signals are needed to execute the instruction. The clock transition associated with timing signal T4 transfers the operand from memory into DR.
- The clock transition associated with the next timing signal T5 transfers to AC the result of the AND logic operation between the contents of DR and AC.
- The same clock transition clears SC to 0, transferring control to timing signal T0 to start a new instruction cycle.

2) ADD to AC

The 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. The micro operations needed to execute this instruction are:

D_1T_4: DR ← M [AR]

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0

The same two timing signals, T4 and T5, are used again but with operation decoder D1 instead of D0.

3) 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

- From the bus diagram we note 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.

4) 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 micro operation:

D_3T_4: M [AR] ← AC, SC ← 0

5) **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.
- The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.
- The instruction is executed with one micro operation:

$$D_4T_4: PC \leftarrow AR, SC \leftarrow 0$$

- The effective address from AR is transferred through the common bus to PC.
- Resetting SC to 0 transfers control to T0. The next instruction is then fetched and executed from the memory address given by the new value in PC.

6) 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 subordinate. This operation was specified in Table 1-5 with the following register transfer:

$$M[AR] \leftarrow PC, PC \leftarrow AR + 1$$

❖ **An Example:** Demonstrates how **BSA** instruction is used with a subordinate is shown in Fig. 1-J. The BSA instruction is 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:

$$M[135] \leftarrow 21, PC \leftarrow 135 + 1 = 136$$

❖ 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 subordinate 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.

- When the BUN instruction is executed, the effective address 21 is transferred to PC.
- The next instruction cycle finds PC with the value 21, so control continues to execute the instruction at the return address.

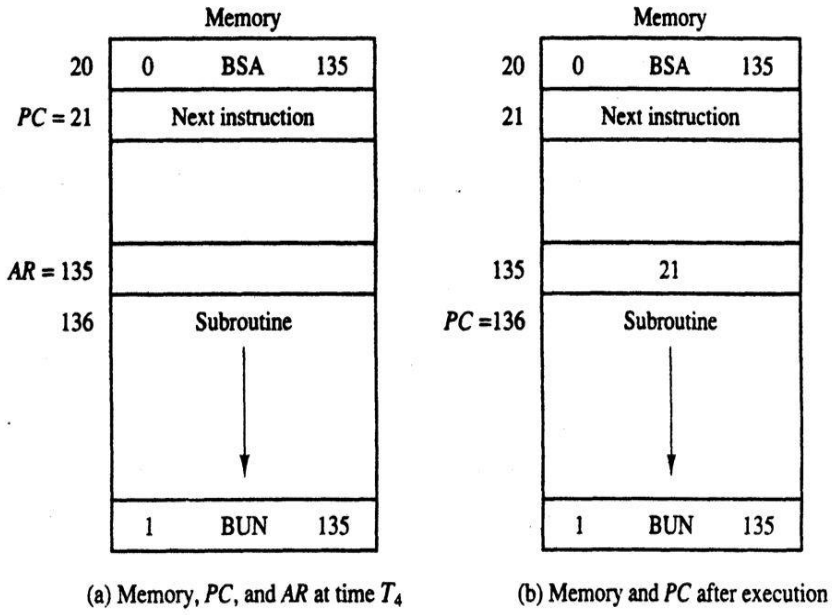


Figure 1-JJ: Example of BSA instruction execution.

- 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 T4 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.

7) ISZ : Increment and Skip if Zero

- 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 one 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

CONTROL FLOWCHART:

A flowchart showing all micro operations for the execution of the seven memory-reference instructions is shown in Fig 1-KK. The control functions are indicated on top of each box. The micro operations that are performed during time T4, T5, or T6 depend on the operation code value.

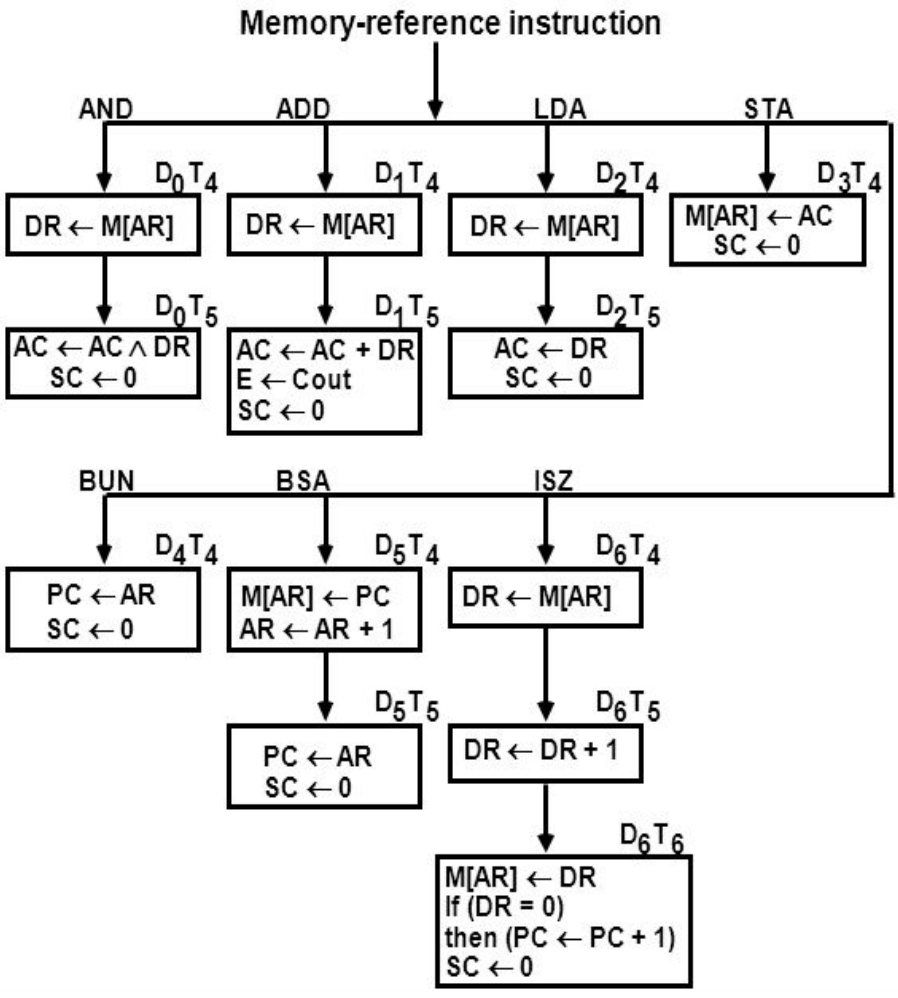


Figure 1-KK: Control flowchart for memory-reference instructions

Input-Output and Interrupt:

Commercial computers include many types of input and output devices. To demonstrate the most basic requirements for input and output communication.

Input-Output Configuration

- The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code.
- 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.

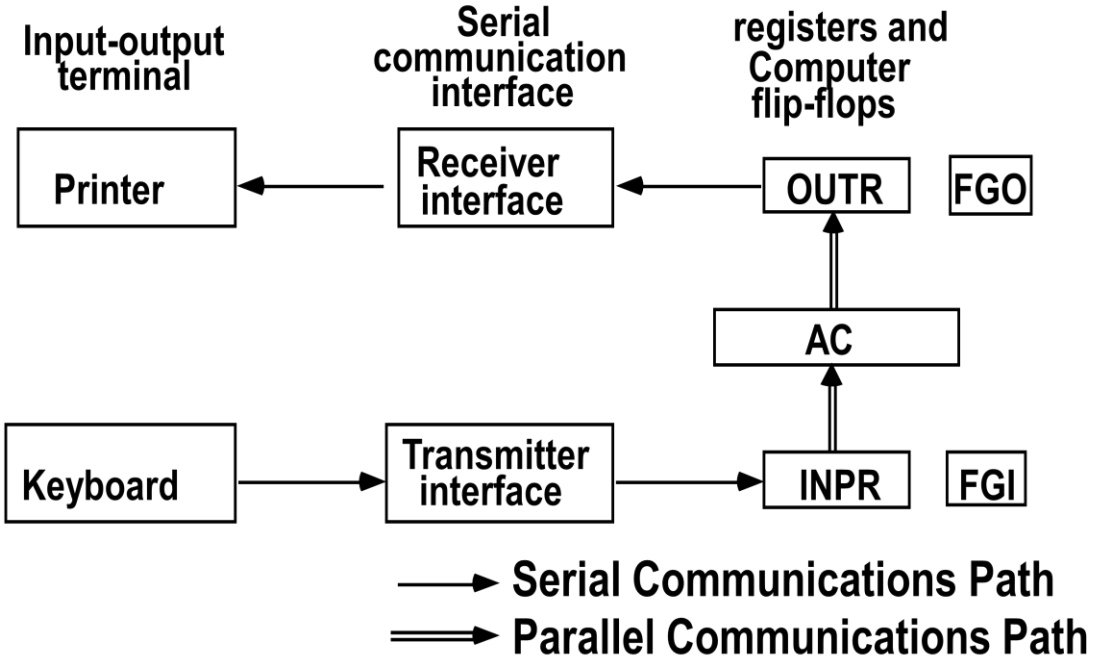


Figure 1-LL: Input Output Configuration

The process of information transfer is as follows.

- Initially, the input flag FGI is cleared to 0.
- When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1.

- The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0.
- Once the flag is cleared, new information can be shifted into INPR by striking another key.

The process of information receiver is as follows.

- 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 coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 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.

Input Output Instructions:

- Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility.
- Input-output instructions have an operation code 1111 and are recognized by the control when D7 = and I = 1.
- The remaining bits of the instruction specify the particular operation. The control functions and micro operations for the input-output instructions are listed in Table 1-5.
- These instructions are executed with the clock transition associated with timing signal T3.
- Each control function needs a Boolean relation D_7IT_3, which we designate for convenience by the symbol p.
- The control function is distinguished by one of the bits in IR (6-11). By assigning the symbol B_i to bit i of IR, all control functions can be denoted by pB_i for i = 6 though 11. The sequence counter SC is cleared to 0 when $p = D_7IT_3 = 1$.

Table 1-5: Input-Output Instructions

$D_7\ IT_3 = p$ (common to all input-output instructions)

IR (i) = B_i [bit in IR (6−11) that specifies the instruction]

	p:	SC ← 0	Clear SC
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	pB_9:	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off

- **INP**: The INP instruction transfers the input information from INPR into the eight low-order bits of AC and also clears the input flag to 0.
- **OUT:** The OUT instruction transfers the eight least significant bits of AC into the output register OUTR and clears the output flag to 0.
- **SKI and SKO:** These two instructions check the status of the flags and cause a skip of the next instruction if the flag is 1.

  The instruction that is skipped will normally be a branch instruction to return and check the flag again. The branch instruction is not skipped if the flag is 0. If the flag is 1, the branch instruction is skipped and an input or output instruction is executed.
- **ION and IOF**: These two instructions set and clear an interrupt enable flip-flop IEN. The purpose of IEN is explained in conjunction with the interrupt operation.

The interrupt cycle:

- The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. 1-MM.
- An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle.
- During the execute phase of the instruction cycle IEN is checked by the control. **If it is 0**, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- **If IEN is 1**, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle.
- If either flag is set to 1 while IEN = 1, flip-flop **R is set to 1**.
- At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted.
- This location may be a processor register, a memory stack, or a specific memory location.
- Here we choose the memory location **at address 0** as the place for storing the return address.
- Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.

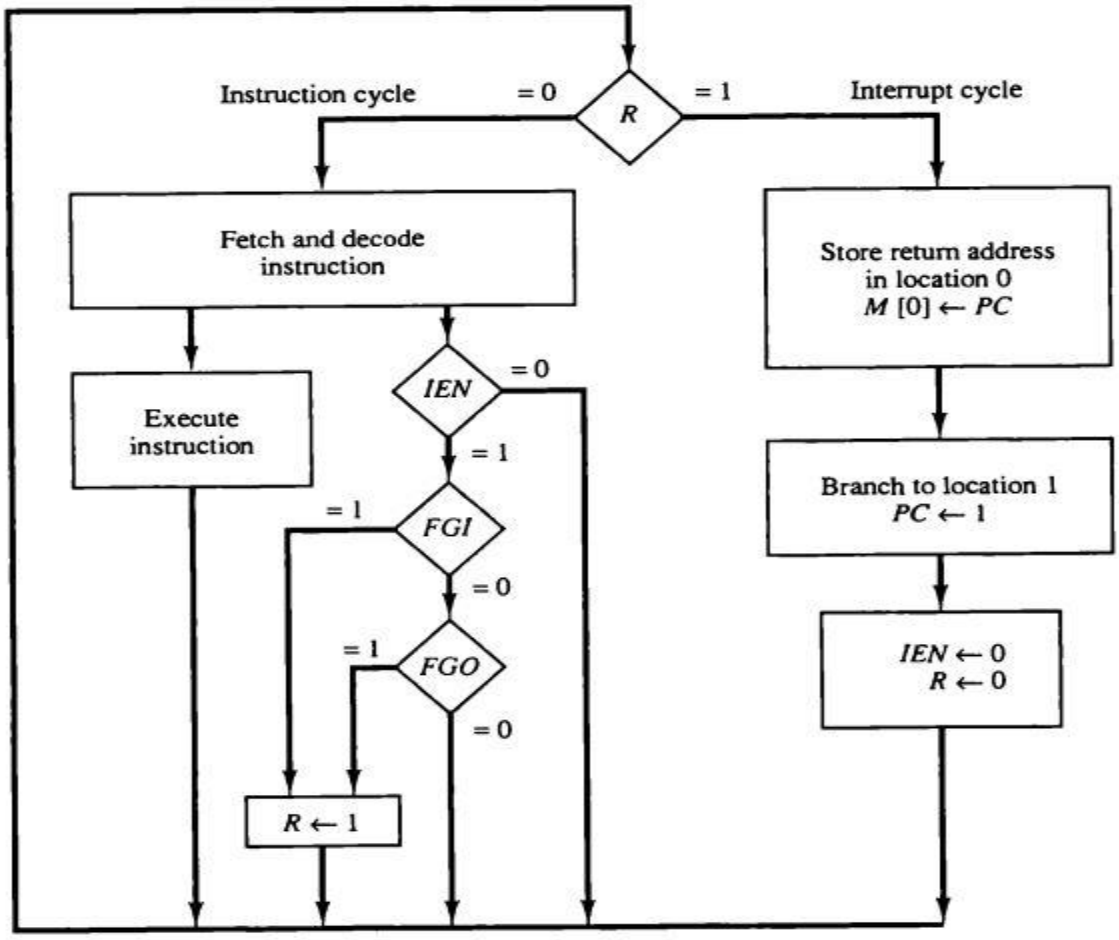


Figure 1-MM: Flowchart for interrupt cycle

- **An example** that shows what happens during the interrupt cycle is shown in Fig 1-NN.
- Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255.

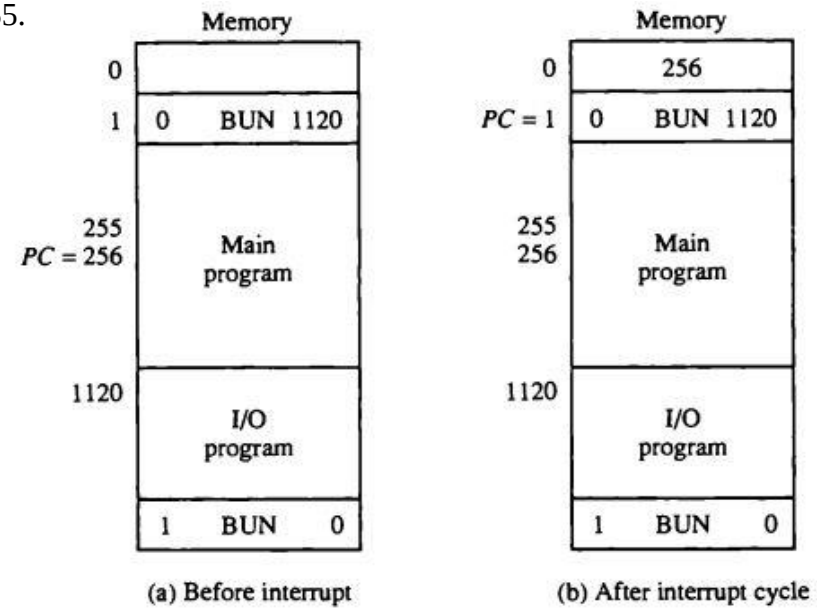


Figure 1-NN: Demonstration of the interrupt cycle

- At this time, the return address 256 is in PC.
- The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig. 1-NN (a).
- When control reaches timing signal T0 and finds that R = 1, it proceeds with the interrupt cycle.
- The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC.
- The **branch instruction** (BUN) at address 1 causes the program to transfer to the input-output service program at address 1120.
- This program checks the flags, determines which flag is set, and then transfers the required input or output information.
- Ones this is done, the instruction **ION** is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted. This is shown in Fig. 1-NN (b).
- The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0.
- This instruction is placed at the end of the I/O service program. After this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I = 1) to read the effective address. The effective address is in location 0 and is the return address that was stored there during the previous interrupt cycle.
- The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Complete Computer Description (Instructions):

The final flowchart of the instruction cycles (including the interrupt cycle) for the basic computer, is shown in Fig. 1-OO.

- The interrupt flip-flop R may be set at any time during the indirect or execute phases.
- Control returns to timing signal T_0 after SC is cleared to 0. If R = 1, the computer goes through an **interrupt cycle**. If R = 0, the computer goes through an **instruction cycle**.
- If the instruction is one of the memory-reference instructions, to execute the decoded instruction according to the flowchart of Fig. 1-KK.
- If the instruction is one of the register-reference instructions, it is executed with one of the micro operations listed in Table 1-3.
- If it is an input-Output instruction, it is executed with one of the micro operations listed in Table 1-5.

- We now modify the fetch and decode phases of the instruction cycle. Instead of using only timing signals T_0, T_1, and T_2 .
- 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 = I, the control will go through an interrupt cycle.
- The interrupt cycle stores the return 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 micro operations.

RT_0: AR ← 0, TR ← PC
RT_1: M[AR] ← TR, PC ← 0
RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0.

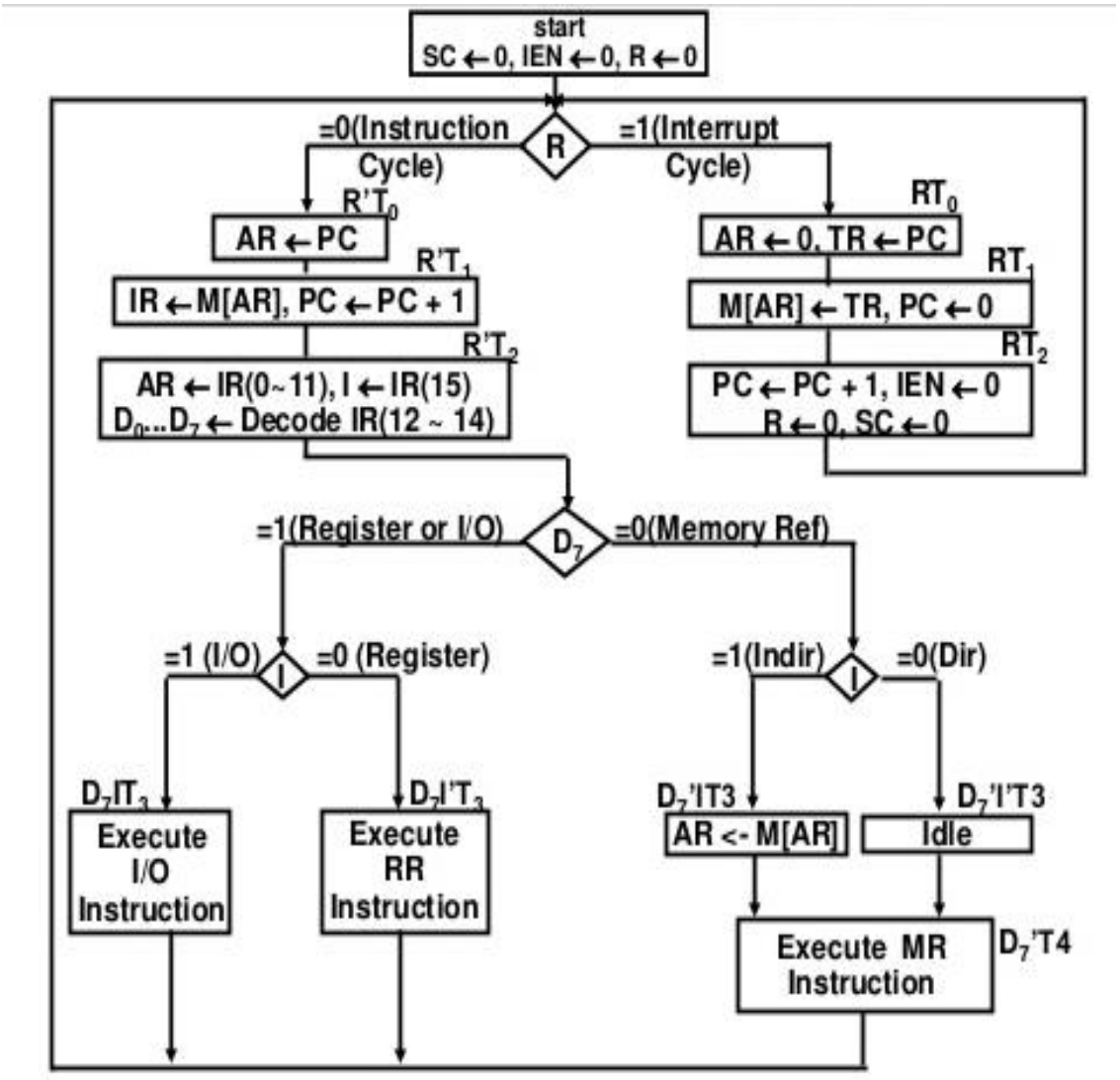


Figure 1-OO: Flowchart for computer operation.