# Unit # 1

By: Dr. Rajesh Tiwari

# Computer

* We can define computer as:
  + an electronic device used to perform a variety of operations.
  + a calculating device used to perform arithmetic operations.
  + a device that operates upon data.

**or**

* We can say that a computer is a general purpose electronic machine used to manipulates raw facts according to a set of instructions (program) that are fed into it.

# Meaning in Computer World

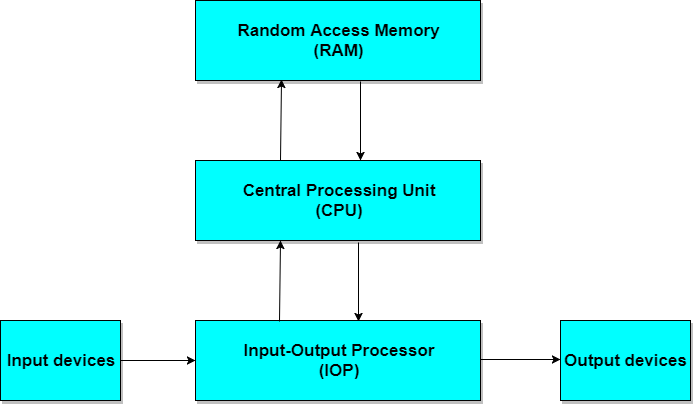
* In computer world:
  + Raw facts are known as data.
  + Meaningful data are known as information.
  + Command that is used to tell the computer what to do or what not to do are known as instructions.
  + All the computer equipment Associated with it are known as computer hardware such as monitor, keyboard, mouse, CPU, modem, speaker, and printer etc.
  + Set of instructions that is used to tell the computer what to do are known as program.
  + Set of programs that are used to perform particular task is known as computer software.
* There are two basic types of computer as given below:
  + Analogue computer
  + Digital computer

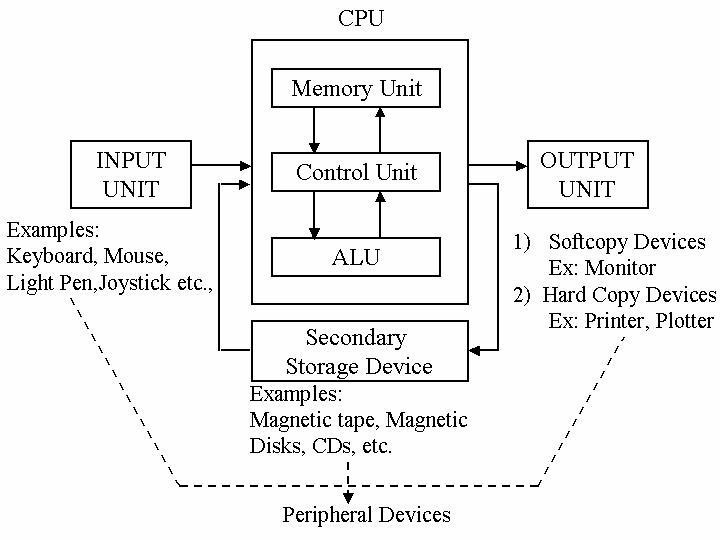
# Digital Computer

* 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.
* These values are processed internally by components that can maintain a limited number of discrete states.
* The decimal digits 0, 1, 2, ..., 9, for example, provide 10 discrete values.
* The first electronic digital computer, developed in the late 1940s, was used primarily for numerical computations and the discrete elements were the digits.
* From this application the term **digital** computer emerged.

# Von Neumann Architecture

* The **von Neumann architecture** describes a general framework, or structure, that a computer's hardware, programming, and data should follow.





* It is the brain of a computer system.
* All major calculation and comparisons are made inside the CPU and it is also responsible for activation and controlling the operation of other unit.
* This unit consists of two major components, that are arithmetic logic unit (ALU) and control unit (CU).
* Arithmetic logic unit performs all arithmetic operations such as addition, subtraction, multiplication and division.
* It also uses logic operation for comparison.
* Control unit of a CPU controls the entire operation of a computer.
* It also controls all devices such as memory, input/output devices connected to the CPU.
* CU fetches instructions from memory, decodes the instruction, interprets the instruction to know what the task are to be performed and sends suitable control signals to the other components to perform for the necessary steps to executes the instruction.
* The input/output unit consists of devices used to transmit information between the external world and computer memory.
* The information fed through the input unit is stored in computer's memory for processing and the final result stored in memory can be recorded or display on the output medium.
* Memory unit is an essential component of a digital computer.
* It is where all data intermediate and final results are stored.
* The data read from the main storage or an input unit are transferred to the computer's memory where they are available for processing.
* This memory unit is used to hold the instructions to be executed and data to be processed.

**Disk Storage Unit**

* Data and instruction enters into a computer system through input device have to stored inside the computer before actual processing start.
* Two types of storage unit are primary and secondary storage unit.
* **Primary Storage Unit**
  + Primary memory has direct link with input unit and output unit. It stores the input data, calculation result.

#### Secondary Storage Unit

* + The primary storage is not able to store data permanently for future use.
  + So some other types of storage technology is required to store the data permanently for long time, it is called secondary or auxiliary storage.

# Computer organization

* It 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.

# Computer Design

* It 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

* 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

* A digital system is an interconnection of digital hardware module that accomplish a specific information processing task.
* Digital systems vary in size and complexity from a few integrated circuits to a complex of interconnected and interacting digital computers.
* Digital system design invariably uses a modular approach.
* The modules are constructed from such digital components as registers, decoders, arithmetic elements, and control logic.
* Various modules are interconnected with common data and control paths to form a digital computer system.

Microoperations

* The operations executed on data stored in registers are called microoperations.
* A microoperations 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 microoperations are shift, count, clear, and load.
* For example, a counter with parallel load is capable of performing the microoperations increment and load.
* The internal hardware organization of a digital computer is defined by specifying:
  + The set of registers it contains and their function.
  + The sequence of microoperations performed on the binary information stored in the registers.
  + The control that initiates the sequence of microoperations.

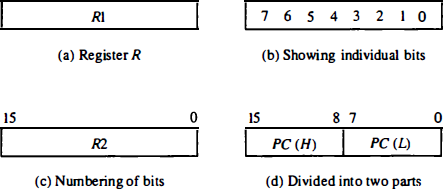
# Register Transfer Language

* The symbolic notation used to describe the microoperation 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 microoperation and transfer the result of the operation to the same or another register.
* The word "language" is borrowed from programmers, who apply this term to programming languages.
* A programming language is a procedure for writing symbols to specify a given computational process.
* A natural language such as English is a system for writing symbols and combining them into words and sentences for the purpose of communication between people.
* A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers

of a digital module.

## Register Transfer

* Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.
* For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR.
* Other designations for registers are PC (for program counter), IR (for instruction register, and R 1 (for processor register).
* The individual flip-flops in an n-bit register are numbered in sequence from 0 through n - 1, starting from 0 in the rightmost position and increasing the numbers toward the left.



* Information transfer from one register to another is designated in symbolic form by means of a replacement operator.
* The statement

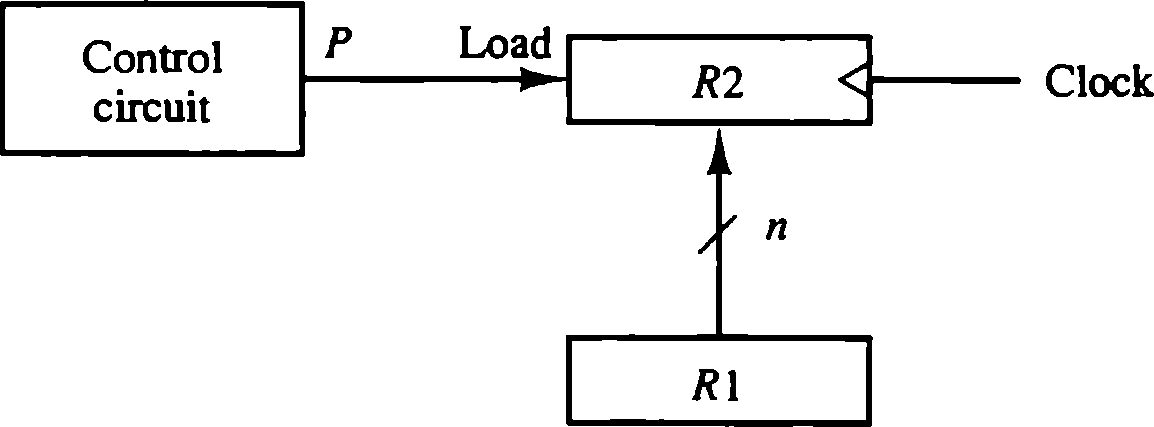
R2 <--R1

denotes a transfer of the content of register R1 into register R2.

* Figure shows the block diagram that depicts the transfer from R1 to R2.
* The n outputs of register R1 are connected to the n inputs of register R2.
* The letter n will be used to indicate any number of bits for the register.
* It will be replaced by an actual number when the length of the register is known.
* Register R2 has a load input that is activated by the control

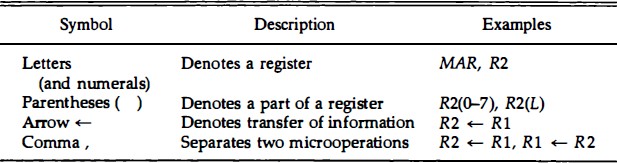
variable P.

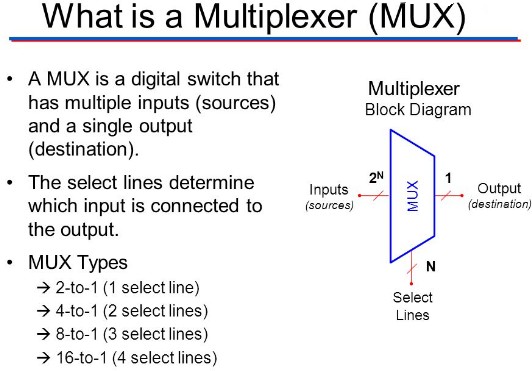
* It is assumed that the control variable is synchronized with the same clock as the one applied to the register.

.

• Figure 4·2 Transfer from Rl to R2 when P = I

**Figure: Transfer from Rl to R2 when P = I .**





|  |  |  |  |
| --- | --- | --- | --- |
| **Select** | **A** | **B** | **Y** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

* A digital computer has many registers, and paths must be 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.

##### A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.

* Control signals determine which register is selected by the bus during each particular register transfer.

## BUS

* 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 next Fig..
* 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.
* 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.

**Bus System for 4 Registers**

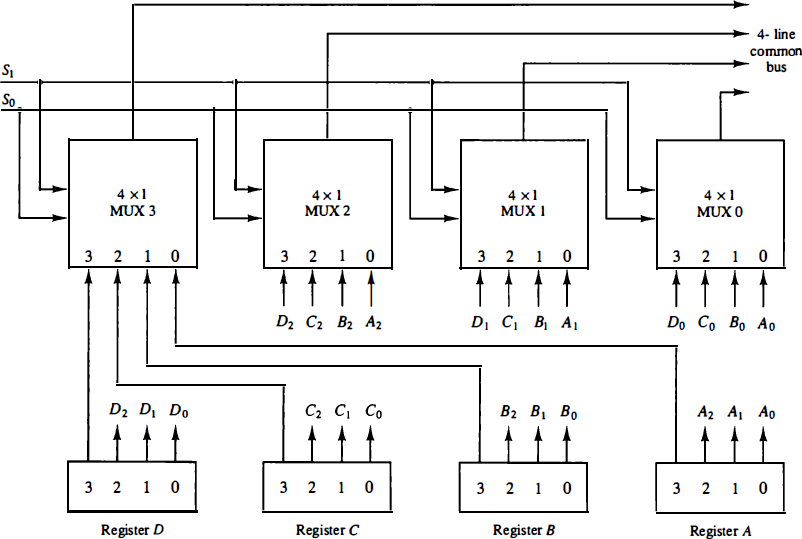


Figure : A

* 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.
* Next Table shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

Function Table of Fig. A

|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **Register Selected** |
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

* + 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-State Bus Buffers

* + 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 a logic significance.
  + Three-state gates may perform any conventional logic, such as AND or NAND.
  + However, the one most commonly used in the design of a bus system is the buffer gate.
  + The graphic symbol of a three-state buffer gate is shown in Fig. B.



Figure B: Graphic symbols for three-state buffer.

* + It is distinguished from a normal buffer by having both a normal input and a control input.
  + 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.
  + The high-impedance state of a three-state gate provides a special feature not available in other gates.
  + Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.

# Memory Transfer

* + The transfer of information from a memory word to the outside environment is called a read operation.
  + The transfer of new information to be stored into the memory is called a write operation.
  + A memory word will be symbolized by the letter M .
  + The particular memory word among the many available is selected by the memory address during the transfer.
  + It is necessary to specify the address of M when writing memory transfer operations.
  + This will be done by enclosing the address in square brackets following the letter M .
  + Consider a memory unit that receives the address from a register, called the address register, symbolized by AR .
  + The data are transferred to another register, called the data register, symbolized by DR .
  + The read operation can be stated as follows:

Read: DR M [AR]

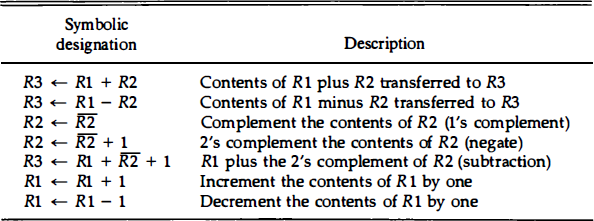
* + This causes a transfer of information into DR from the memory

word M selected by the address in AR .

* + The write operation transfers the content of a data register to a memory word M selected by the address.
  + Assume that the input data are in register R l

R3 R1 + R2 + 1

* + R2 is the symbol for the 1' s complement of R2. Adding 1 to the 1's complement produces the 2's complement. Adding the contents of R 1 to the 2's complement of R2 is equivalent to R1 - R2.



* To implement the add microoperation with, we need the registers that hold the data and the digital component that performs the arithmetic addition.
* The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a **full-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 : C shows the interconnections of four full- adders (FA) to provide a 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 C0 and the output carry is C,.
  + The S outputs of the full-adders generate the required sum bits.

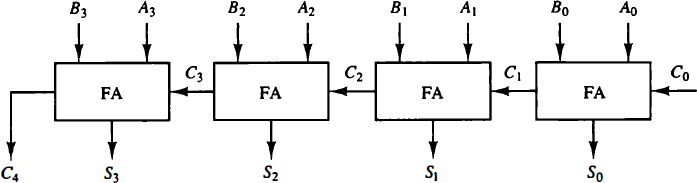


Fig. C: 4 bit binary adder

* + An n-bit binary adder requires n full-adders.
  + The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.
  + The n data bits for the A inputs come from one register (such as R1), and the n data bits for the B inputs come from another register (such as R2).
  + The sum can be transferred to a third register or to one of the source registers (R 1 or R2), replacing its previous content.

**Binary Adder - Subtractor**

* The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full- adder.
* A 4-bit adder-subtractor circuit is shown in Fig. D.
* The mode input M controls the operation. When M = 0 the circuit is an **adder** and when M = 1 the circuit becomes a **subtractor**.
* Each exclusive-OR gate receives input M and one of the inputs of B.
* When M = 0, we have B 0 = B.
* The full-adders receive the value of B, the input carry is O, and the circuit performs A plus B .

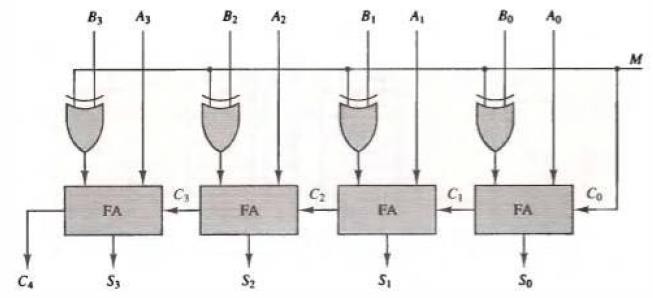
# Binary Adder - Subtractor

* + When M = 1, we have B 1 = B' and C0 = 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.
  + For unsigned numbers, this gives A - B if A '" B or the 2's complement of (B - A) if A < B.
  + For signed numbers, the result is A – B provided that there is no overflow.

# 4 bit Adder - Subtractor



**Fig. D: 4 bit Adder - Subtractor**

* + The increment microoperation adds one to a number in a register.
  + For example, if a 4-bit register has a binary value 0110, it will go

0111 after it is incremented.

* + The diagram of a 4-bit combinational circuit incrementer is shown in Fig. E.
  + 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 A0 through A3, adds one to it, and generates the incremented output in S0 through S3.
  + The output carry C4, will be 1 only after incrementing binary 1111.
  + This also causes outputs S0 through S3 to go to 0.
  + The circuit of Fig. E can be extended to an n-bit binary incrementer by extending the diagram to include n half-adders.

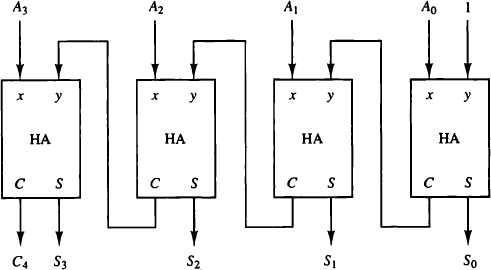


Fig. E: 4-bit binary incrementer

# Logic Microoperations

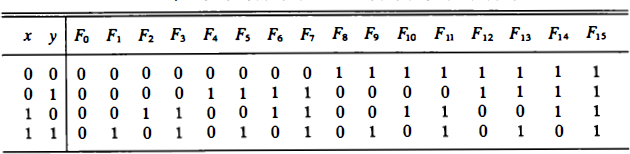
* + Logic microoperations specify binary operations for strings of bits stored in registers.
  + These operations consider each bit of the register separately and treat them as binary variables.
  + For example, the exclusive-OR microoperation with the contents of two registers R1 and R2 is symbolized by the statement:
  + It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1.

P: R1 R1 R2

* + Special symbols will be adopted for the logic microoperations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions.
  + The symbol V will be used to denote an OR microoperation and the symbol to denote an AND microoperation.
  + The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.
  + By using different symbols, it will be possible to differentiate between a logic microoperation and a control (or Boolean) function.

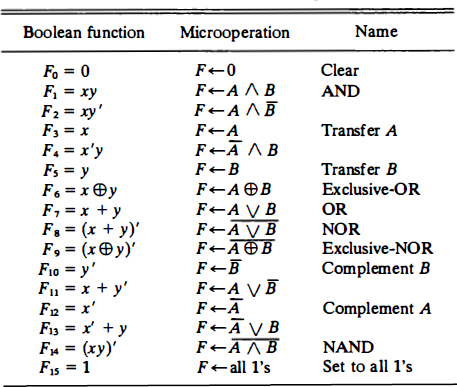
TABLE A:

Truth Tables for 16 Functions of Two Variables



* + There are 16 different logic operations that can be performed with two binary variables.
  + They can be determined from all possible truth tables obtained with two binary variables as shown in Table A.
  + In this table, each of the 16 columns F0 through F15 represents a truth table of one possible Boolean function for the two variables x and y.
  + Note that the functions are determined from the 16 binary combinations that can be assigned to F .

Sixteen Logic Microoperations



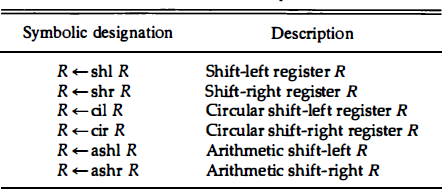
Shift Microoperations

* + Shift rnicrooperations are used for serial transfer of data.
  + They are also used in conjunction with arithmetic, logic, and other data-processing operations.
  + The contents of a register can be shifted to the left or the right.
  + 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.

Shift Microoperations



## Arithmetic Logic Shift Unit

**Basic Computer Organization and Design**

## Instruction Codes

* + 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,

operands, and the sequence by which processing has to occur.

* + The data processing task may be altered by specifying a new program with different instructions or specifying the same instructions with different data.
  + A **computer instruction** is a binary code that specifies a sequence of microoperations for the computer.
  + Instruction codes together with data are stored in memory.

# Instruction Codes

* + The computer reads each instruction from memory and places it in a control register.
  + The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.
  + 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 .

## Instruction Codes

* + **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

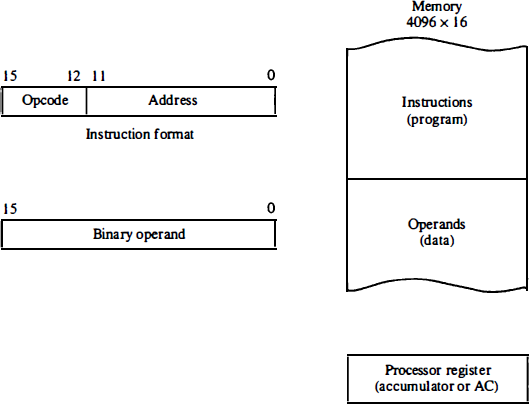
interpretation.

* + 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.**

# Instruction Codes

* + The operation code must consist of at least n bits for a given 2n (or less) distinct operations.
  + **For example** , 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.
  + The simplest way to organize a computer is to have one processor register and an instruction code format with two parts.
  + The first part specifies the operation to be performed and the second specifies an address.
  + The memory address tells the control where to find an operand in memory.
  + This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.
  + Figure F: 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 212 = 4096.
  + If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.
  + The control reads a 16-bit instruction from the program portion of memory.
  + It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.
  + It then executes the operation specified by the operation code.

## Stored Program Organization



**Fig. F: Stored program organization**.

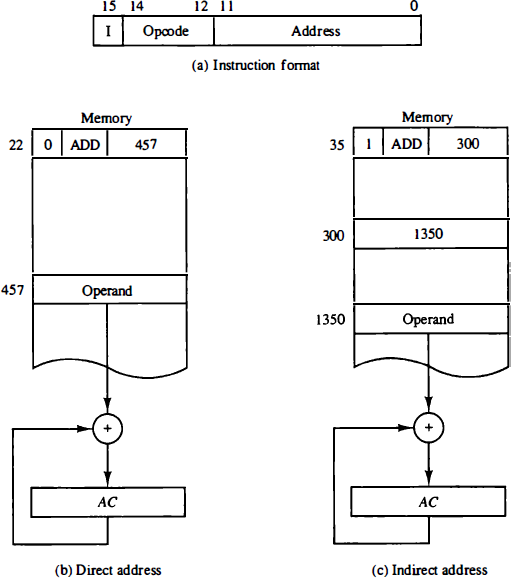
# Stored Program Organization

* + Computers that have a single-processor register usually assign to it the name accumulator and label it AC .
  + The operation is performed with the memory operand and the content of AC .
  + If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes.
  + For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register.
  + They do not need an operand from memory.
  + For these types of operations, the second part of the instruction code (bits 0 through 1 1 ) is not needed for specifying a memory address and can be used to specify other operations for the computer.

## Indirect Address

* + When the second part of an instruction code specifies an operand, the instruction is said to have an **immediate operand**.
  + When the second part specifies the address of an operand, the instruction is said to have a **direct address**.
  + This is in contrast to a third possibility called **indirect address**, where the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.
  + One bit of the instruction code can be used to distinguish between a direct and an indirect address.
  + Consider the instruction code format shown in Fig. G(a).
  + It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.
  + The mode bit is 0 for a direct address and 1 for an indirect address.
  + A direct address instruction is shown in Fig. G(b).
  + It is placed in address 22 in memory.
  + The I bit is 0, so the instruction is recognized as a direct address instruction.
  + The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
  + The control finds the operand in memory at address 457 and adds it to the content of AC .
  + The instruction in address 35 shown in Fig. G ( c) has a mode bit I = 1.
  + Therefore, it is recognized as an indirect address instruction.
  + The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand.
  + The address of the operand in this case is 1350. The operand found in address 1350 is then added to the content of AC .
  + The indirect address instruction needs two references to memory to fetch an operand.
  + The first reference is needed to read the address of the

operand; the second is for the operand itself.

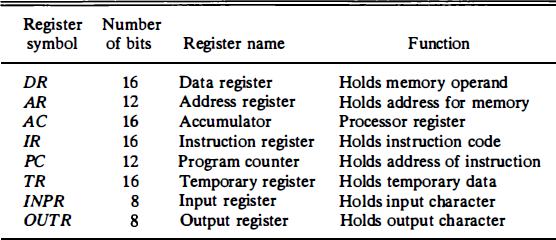


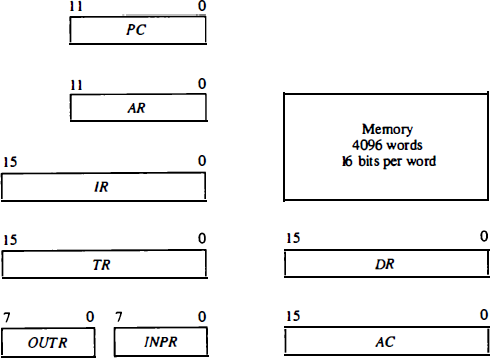
**Fig. G: Demonstration of direct and indirect address**

* + The **effective address** to be the address of the operand in a computation-type instruction or the target address in a branch- type instruction.
  + Thus the **effective address** in the instruction of Fig. G (b) is 457 and in the instruction of Fig G ( c) is 1350.
  + 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.





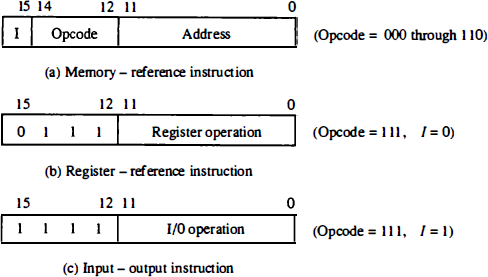
**Figure. H: Basic computer registers and memory**

## Computer Instructions

* + The basic computer has three instruction code formats, as shown in Fig. I.
  + Each format has 16 bits.
  + The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.
  + A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I.
  + The register reference instructions are recognized by the operation

code 111 with a 0 in the leftmost bit (bit 15) of the instruction.

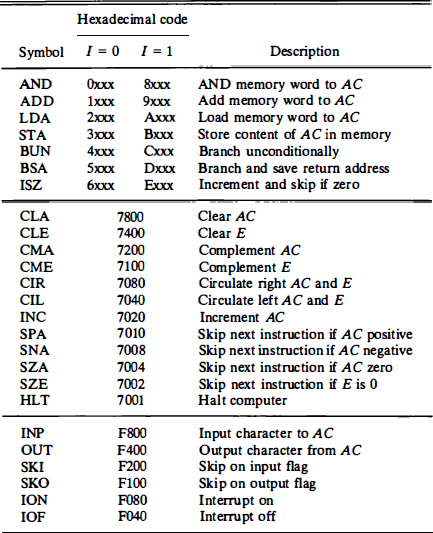
* + A register-reference instruction specifies an operation on or a test of the AC register.
  + An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
  + Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 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.



**Fig. I: 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 opcode bits in positions 12 though 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 3-bit opcode is 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 i s 1, the instruction is an input-output type.
  + 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 1 1 1 .

## Basic Computer Instructions



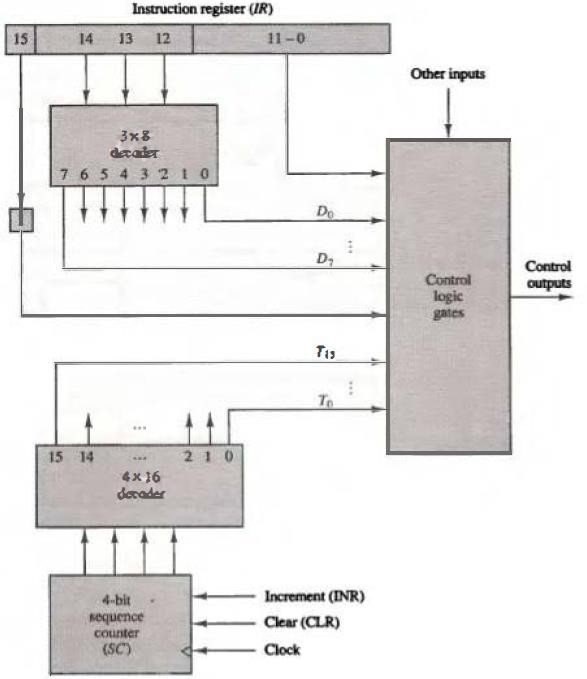
* + The timing for all registers in the basic computer is controlled by a master clock generator.
  + The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
  + The clock pulses do not change the state of a register unless 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 microoperations for the accumulator.
  + There are two major types of control organization: **hardwired control** and **microprogrammed control**.

##### 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.

##### Microprogrammed organization

* + - The control information is stored in a control memory.
    - The control memory is programmed to initiate the required sequence of microoperations.
  + A **hardwired control** requires changes in the wiring among the various components if the design has to be modified or changed.
  + A **microprogrammed control** changes or modifications can be done by updating the microprogram in control memory.

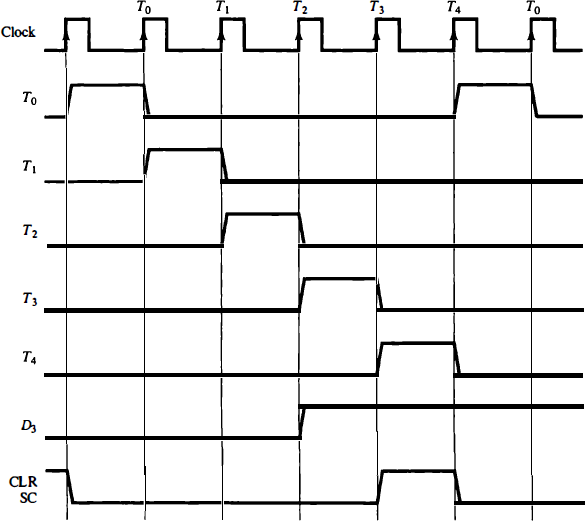


**Fig. J: Control unit of basic Computer**

* 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. J, 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 x 8

decoder.

* The eight outputs of the decoder are designated by the symbols D0 through D7 .
  + 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.
  + Bit 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 .
  + 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 x 16 decoder.
  + Once in awhile, the counter is cleared to 0, causing the next active timing signal to be To.
  + For example, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3, and T4 in sequence.
  + At time T4, SC is cleared to 0 if decoder output D3 is active.
  + This is expressed symbolically by the statement D3T4: SC <- 0



**Fig. K: Example of control timing signals**.

# Timing and Control

* + A program residing in the memory unit of the computer consists of a sequence of instructions.
  + The program is executed in the computer by going through a cycle for each instruction.
  + Each instruction cycle in turn is subdivided into a sequence of

sub-cycles or phases.

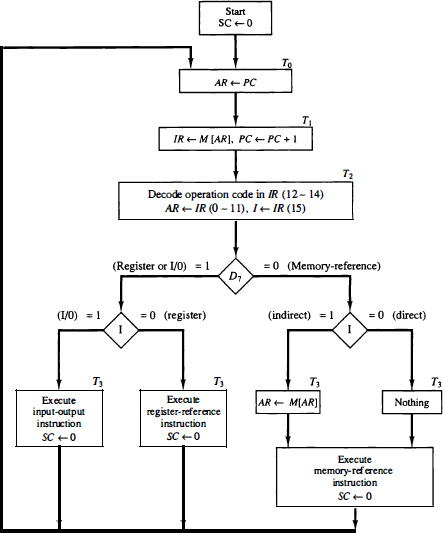
* + In the basic computer each instruction cycle consists of the following phases:

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

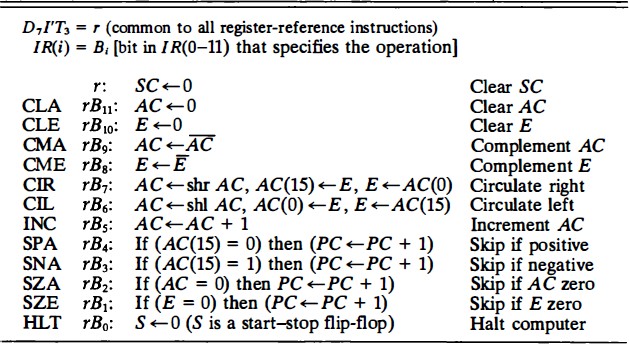
indirect address.

1. Execute the instruction.

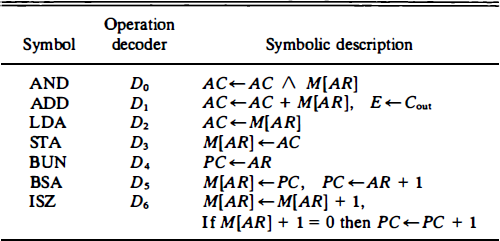
## Instruction Cycle



**Instructions**



* + The seven memory-reference instructions are given in the next table.
  + 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.



### 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 rnicrooperations that execute this instruction are:
      * D0T4: DR <- M [AR]
      * D0T5: AC <- AC /\ DR, SC <--- 0

### ADD to AC

* + - This instruction adds the content of the memory word specified by the effective address to the value of AC .
    - The sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flip-flop.
    - The microoperations needed to execute this instruction are

D1T4 : DR <--- M [AR]

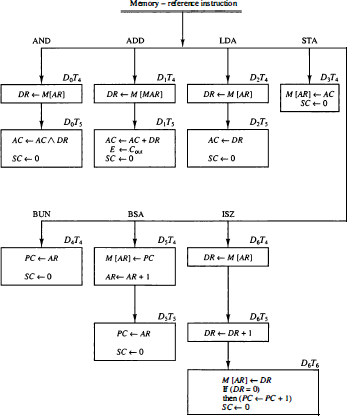
D1T5: AC <--- AC + DR, E <--- Cout, SC <--- 0

* + The same two timing signals, T4 and T5, are used again but with operation decoder D1 instead of D0, which was used for the AND instruction.
  + After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of microoperations that the control follows during the execution of a memory- reference instruction.

### LDA: Load to AC

* + - This instruction transfers the memory word specified by the effective address to AC.
    - The rnicrooperations needed to execute this instruction are
      * D2T4: DR <--- M [AR]
      * D2T5: AC <--- DR, SC <--- 0
  + **STA: Store AC**
  + **BUN: Branch Unconditionally**
  + **BSA: Branch and Save Return Address**
  + **ISZ: Increment and Skip if Zero**

**Flowchart for memory-reference instructions**



# Thank You