Chapter 1 Digital Computers

UNIT - I

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

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

.

VISHNU
UNIVERSAL LEARNING

1.1 Digital Computers

The digital computer is a digital system that performs various computational tasks.

- ✓ The word digital implies information in the computers 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.
- ✓ Decimal digits 0, 1, … 9 provide ten discrete values.
- ✓ The physical restrictions of the components that are used, restrict us to two states: true/false, on/off, yes/no, high/low, 0/1 are said to be binary.
- ✓ Digital computers use the binary number system, which has two digits: 0 and 1.
- ✓ A Binary digit, 0 or 1, is called bit.
- ✓ Information is represented in digital computers in groups of bits. The group of bits are used to develop complete sets of instructions for performing various types of computations.
- ✓ Groups of bits can represent binary numbers, decimal digits, or letters.
  01000001b = 65d = 41h = ‘A’ = some control word

Computer system is divided into two functional entities: hardware & software. Hardware: Lowest level in a computer. It comprises all of the physical parts (electronic and electromechanical devices) of a computer.
Software: Sequences of instructions and data that make computers do useful work.

Program: sequence of instructions for a particular task done by computer.

Database: The data that are manipulated by the program
A computer system is composed of its hardware and the system software available for its use.

Operating system: programs included in system software package. link between hardware and user needs.
Compiler is system software.

1.1.1 Three components of hardware: (Block diagram of a Digital Computer)

1. Central Processing Unit (CPU):
2. Memory
3. Input and Output Processor(IOP):

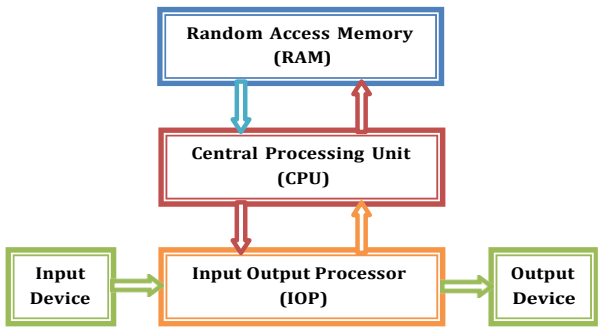


Figure: Block diagram of a Digital Computer

Central Processing Unit (CPU):

- ✓ Controls operation of system by issuing timing and control signals
- ✓ Contains ALU which does all computation on data
- ✓ Contains registers for storing data, control circuits for fetching and executing instructions

Memory:

- ✓ Storage for instructions and data (no distinction).
- ✓ It is called Random Access Memory (RAM) because CPU can access any location in memory at random and retrieve the binary info within a fixed interval of time.

Input and Output Processor (IOP):

- ✓ Contains electronic circuits for communicating and controlling the (interface) transfer of information between computer and outside world, e.g. keyboard, printer, scanner, mass storage, etc.

1.1.2 Three viewpoints for hardware:

1. Computer Organization
2. Computer Design
3. Computer Architecture

Computer organization:

- ✓ It is concerned with the way the hardware components operate and the way they are connected together to form the computer system.(interconnection of h/w to form the computer system)

Computer design:

- ✓ It is concerned with the hardware design of the computer.
- ✓ Computer design is concerned with the determination of what hardware should be used and how the parts should be connected.
- ✓ This aspect is referred to as computer implementation.

Computer architecture:

- ✓ It is concern with the structure and behavior of the computer perceived by the user.
- ✓ It Includes instruction format, instruction set, and techniques for addressing memory.
- ✓ It is also concern with specifications of the various functional modules like processor and memory, and structuring them.

1.1.3 Two Basic Types of Computer Architectures:

1. Von Neumann Architecture
2. Harvard Architecture

1. Von Neumann Architecture:

- ✓ The von Neumann architecture describes a general framework, or structure, that a computer's hardware, programming, and data should follow.
- ✓ All computers more or less based on the same basic design, the von Neumann Architecture! whatever it be a multi-million dollar mainframe or a Palm Pilot.

✓ Von Neumann architecture was first published by John von Neumann in 1945.

Von Neumann computer architecture design consists of a Control Unit, Arithmetic and Logic Unit (ALU), Memory Unit, Registers and Inputs/ Outputs.

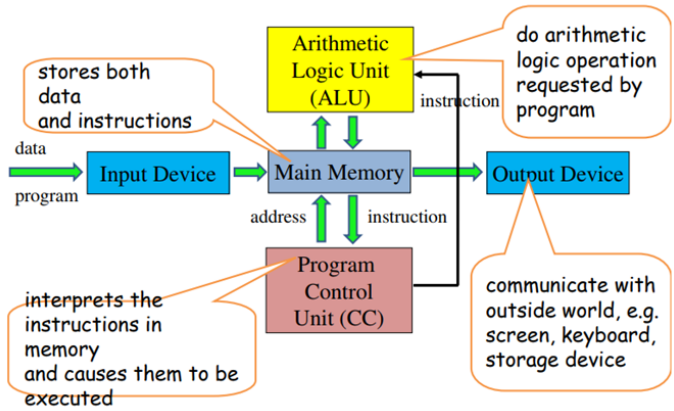
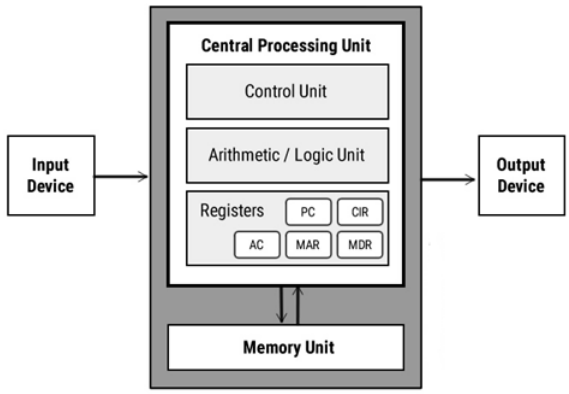


Figure 1 von Neumann architecture



Components of von Neumann Architecture:

1. The central arithmetic unit i.e., ALU performs computational and logical operations.
2. Memory: main memory, fast such as RAM.
3. Control Unit: directs other components of the computer to perform certain actions. The control unit controls the operation of the computer's ALU, memory and input/output devices, telling them how to respond to the program instructions it has just read and interpreted from the memory unit. The control unit also provides the timing and control signals required by other computer components.
4. Man-Machine interfaces: i.e., input and output devices.

Central Processing Unit (CPU):

- ✓ The Central Processing Unit (CPU) is the electronic circuit responsible for executing the instructions of a computer program.
- ✓ It is sometimes referred to as the microprocessor or processor.
- ✓ The CPU contains the ALU, CU and a variety of registers.
- ✓ Registers are high speed storage areas in the CPU. All data must be stored in a register before it can be processed.

Stored-Program Principle:

- ✓ Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory.
- ✓ Instructions execution is carried out sequentially, one instruction at a time.
- ✓ CPU can be either reading an instruction or reading/writing data from/ to the memory.
- ✓ Both cannot occur at the same time since the instructions and data use same signal pathways and memory.
- ✓ This design is still used in most computers produced today.

  Eg: Desktop Personal Computer

2. Harvard Architecture

- ✓ The Harvard architecture is computer architecture with physically separate storage and signal pathways for instructions and data.
- ✓ The term originated from the Harvard Mark I relay-based computer, which stored instructions on punched tape (24 bits wide) and data in electro-mechanical counters.
- ✓ The CPU can read an instruction and data from memory at the same time, leading to double the memory bandwidth.

  Eg: Microcontroller – based computer systems, DSP(Digital Signal Processing) base computer systems.

What is the difference between von Neumann architecture and a Harvard architecture?

VAN-NEUMANN ARCHITECTURE	HARVARD ARCHITECTURE
The data and program are stored in the same memory	The data and program memories are separate
Used in conventional processors found in PCs and Servers, and embedded systems with only control functions.	Used in DSPs and other processors found in latest embedded systems and Mobile
communication systems, audio, speech, image processing systems	
The code is executed serially and takes more clock cycles	The code is executed in parallel
There is no exclusive Multiplier	It has MAC (Multiply Accumulate)
Absence of Barrel Shifter	Barrel Shifter help in shifting and rotating operations of the data
The programs can be optimized in lesser size	The program tend to grow big in size

1.2 Basic Computer Organization and Design

The organization of the computer is defined by its internal registers, timing and control structure, and the set of instructions that it uses.

- Every different processor type has its own design (different registers, buses, microoperations, machine instructions, etc)
- Modern processor is a very complex device it contains
  - Many registers
  - Multiple arithmetic units, for both integer and floating point calculations
  - The ability to pipeline several consecutive instructions to speed execution
  - Etc.
- However, to understand how processors work, we will start with a simplified processor model

1.2.1 Instruction code

The internal organization of a digital system is defined by the sequence of microoperations it performs on data stored in its registers.

A program -a set of instructions that specify the operations, operand, and the sequence by which processing has to occur.

Instruction Cycle

- ✓ A computer instruction is a binary code that specifies a sequence of micro operations for the computer.
- ✓ Instructions codes and data are stored in memory.
  - – **Fetch:** The computer reads each instruction from memory and places it in control register.
  - – **Decode:** The control then interprets the binary code of the instruction
  - – **Execution:** proceeds to execute it by issuing a sequence of micro operations
- ✓ The computer reads each instruction from memory and places it in a control register
- ✓ The control unit interprets the binary code of the instruction 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 is called **Stored-Program organization** is the important property of a general-purpose computer.

Instruction code

- ✓ An Instruction code is a group of bits that instructs the computer to perform a specific operation (sequence of microoperations). It is divided into parts (basic part is the operation part and address of operand)
- ✓ The operation code of an instruction is a group of bits that defines certain operations such as add, subtract, shift, and complement
- ✓ The number of bits required for the operation code depends on the total number of operations available in the computer
- ✓ Consist of at least n bits for a given 2n(or less) distinct operations
- ✓ 2^n (or little less) distinct operations $\rightarrow$ n bit operation code

Relation between computer operation and microoperation is:

From Computer Memory-> reads instruction-> Control Unit-> interprets-> Operation Code-> issues Control signals->generates Microoperations->on registers.

- ✓ Operation code is sometimes called a macrooperation because it specifies a set of microoperations.
- ✓ An operation must be performed on some data stored in processor registers or in memory.

Address part

- ✓ An instruction code must therefore specify not only the operation, but also the location of the operands (in registers or in the memory), and where the result will be stored (registers/memory)
- ✓ 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.
- ✓ Each computer has its own particular instruction code format.
- ✓ Instruction code formats are conceived by computer designers who specify the architecture of the computer.

1.2.1.1 Stored Program Organization

- The simplest way to organize a computer is to have one processor register (accumulator AC) and an instruction code format with two parts (op code, address)
- An instruction code is usually divided into *operation code, operand address, addressing mode*, etc.
- The memory address tells the control where to find an operand in memory.

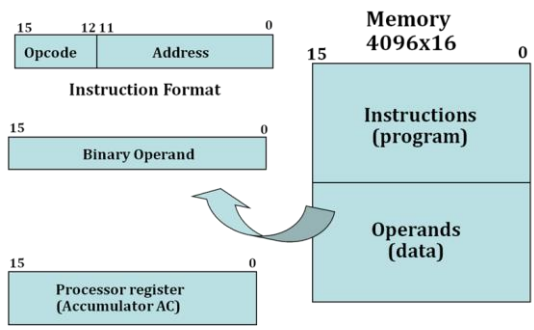


Figure: Stored Program Organization

4-bits for Opcode then :: 24=16 operations possible.
12-bits for address then:: 212=4096 words in memory each of 16-bit words

The simplest way to organize a computer:

- ✓ One processor register: AC (Accumulator).
- ✓ The operation is performed with the memory operand and the content of AC.
- ✓ Instruction code format with two parts: Op. Code + Address.
- ✓ Op. Code: specify 16 possible operations (4 bit). Example Clear AC, Increment AC, Complement AC...
- ✓ Address: specify the address of an operand (12 bit).
- ✓ If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction (address field) can be used for other purpose (e.g. clear AC. increment AC).

- ✓ Memory: 12 bit = 4096 words (Instruction and Data are stored).
- ✓ Store each instruction code (program) and operand (data) in 16-bit memory word.
- ✓ When the second part of the instruction code specifies an operand, the instruction is said to have an immediate operand.
- ✓ Addressing Mode (The mode bit I is 0 for direct address and 1 for an indirect address).

1.2.1.2 Indirect Address

- ✓ When the second part of the instruction specifies
  - an operand – immediate operand(immediate address)
  - the address of an operand – direct address
  - the address of a memory word where the address of the operation can be found – indirect address
- ✓ The indirect address instruction needs two references to memory to fetch an operand
- ✓ Effective address –the address of the operand
- ✓ One bit of the instruction code can be used to distinguish between direct & indirect addresses.

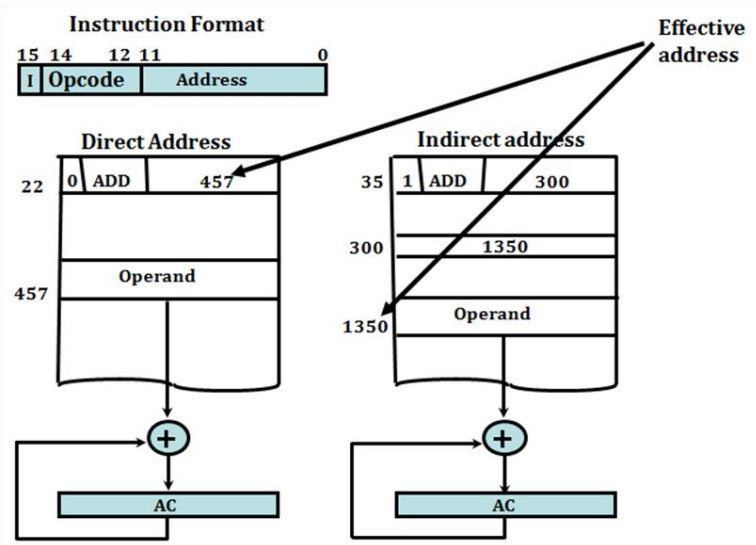
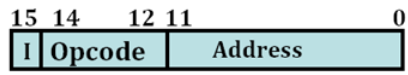


Figure: Demonstration of direct and indirect address

- Effective address: the address of the operand in a computation-type instruction or the target address in a branch-type instruction
- The pointer can be placed in a processor register instead of memory as done in commercial computers

Instruction Format



In an instruction format:

– First 12 bits (0-11) specify an address.

– Next 3 bits(12-14) specify operation code (opcode), or type of operation.

– Left most bit(15) specify the addressing mode I

- I = 0 for direct address
- I = 1 for indirect address

1.2.2 Computer Registers

- Computer instructions are normally stored in consecutive memory locations and executed sequentially one at a time.
- The control reads an instruction from a specific address in memory and executes it, and so on.
- This type of 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.

Table: List of Registers for Basic Computer

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

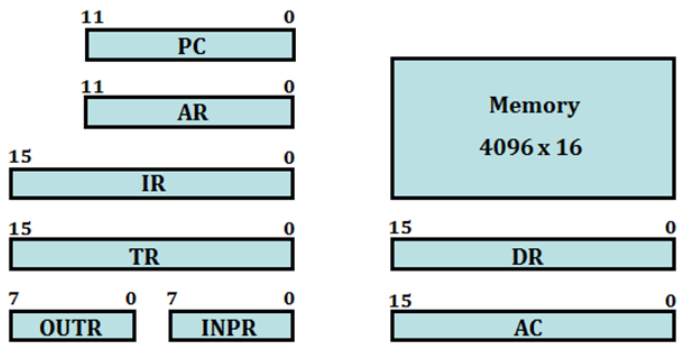


Figure: Basic Computer Registers and Memory

- A processor has many registers to hold instructions, addresses, data, etc..
- The processor has a register, the Program Counter (PC) that holds the memory address of the next instruction to get.
  - Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits.
- In a direct or indirect addressing, the processor needs to keep track of what locations in memory it is addressing: The Address Register (AR) is used for this.
  - The AR is a 12 bit register in the Basic Computer.
- When an operand is found, using either direct or indirect addressing, it is placed in the Data Register (DR). The processor then uses this value as data for its operation.
- The instruction read from memory is placed in instruction register(IR).
- The Basic Computer has a single general purpose register – the Accumulator (AC).
- The significance of a general purpose register is that it can be referred to in instructions.
  - e.g. load AC with the contents of a specific memory location; store the contents of AC into a specified memory location.
- Often a processor will need a scratch register to store intermediate results or other temporary data; in the Basic Computer this is the Temporary Register (TR).
- The Basic Computer uses a very simple model of input/output (I/O) operations
  - Input devices are considered to send 8 bits of character data to the processor.
  - The processor can send 8 bits of character data to output devices.
- The Input Register (INPR) holds an 8 bit character gotten from an input device.
- The Output Register (OUTR) holds an 8 bit character to be send to an output device.

1.2.2.1 Common Bus System

- ✓ The basic computer has 8 registers, a memory unit, and a control unit.
- ✓ Paths must be provided to transfer information from one register to another and between memory and registers.
- ✓ A more efficient scheme for transferring information in a system with many registers is to use a common bus.
- ✓ The connection of the registers and memory of the basic computer to a common bus system are shown in below Figure.

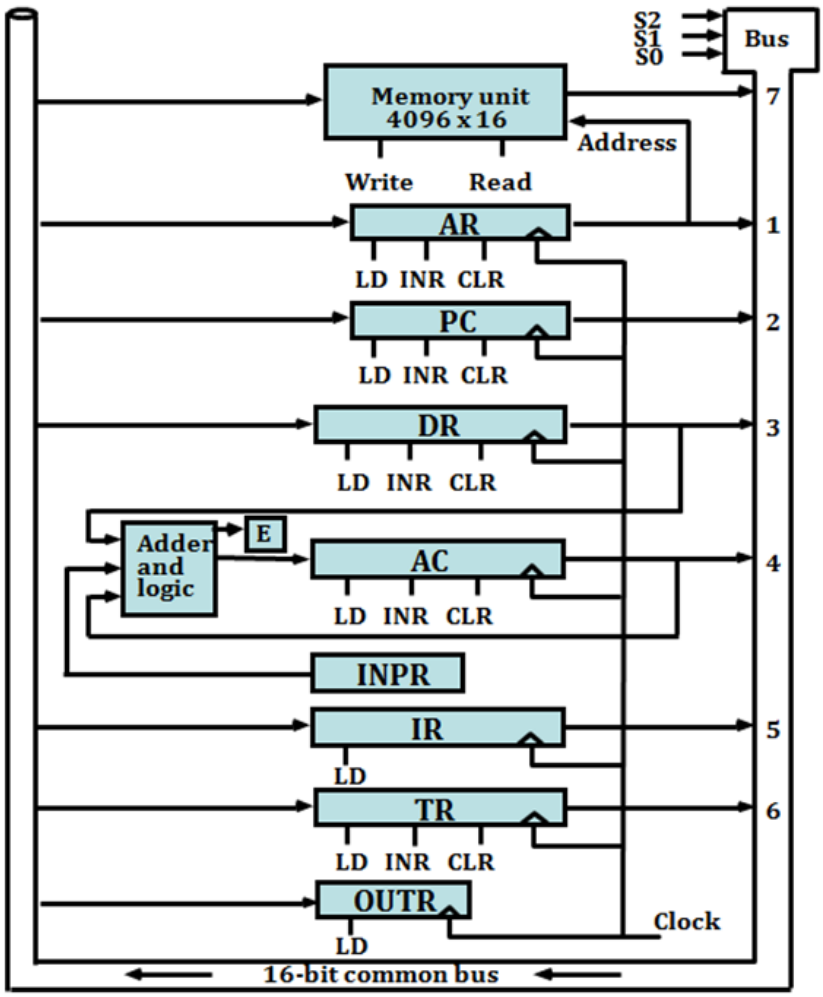


Figure: Common Bus system of computer

1. The outputs of six registers and memory are connected to the common bus.
2. **Selection variables:** The specific output is selected by a MUX (S0, S1, S2). Used to specify a register whose output is connected to the common bus at any given time. To select one register out of 8, we need 3 select variables. For example, if S2S1S0 = 011, the output of DR is directed to the common bus.
3. **Control variables:** LD, INC, CLR, Write, and Read. Various control variables are used to select the paths of information and the operation of the registers.
   Load input (LD): Enables the input of a register connected to the common bus. When LD = 1 for a register, the data on the common bus is read into the register during the next clock pulse transition.
   Increment input (INR): Increments the content of a register.
   Clear input (CLR): Clear the content of a register to zero.

4. E (extended AC bit): flip-flop holds the carry
   DR, AC, IR, and TR: have 16 bits each
   AR and PC: have 12 bits each since they hold a memory address
5. When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to zeros
   When AR or PC receives information from the bus, only the 12 least significant bits are transferred into the register
6. INPR and OUTR: communicate with the eight least significant bits in the bus
   INPR: Receives a character from the input device (keyboard,…etc) which is then transferred to AC
   OUTR: Receives a character from AC and delivers it to an output device (say a Monitor)
   Five registers have three control inputs: LD (load), INR (increment), and CLR (clear)
   Register º binary counter with parallel load and synchronous clear.

Memory Address

7. 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
8. By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise.
   – Register → Memory: Write operation
   – Memory → Register: Read operation (note that AC cannot directly read from memory!!)
9. 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 transition at the end of the cycle transfers the content of the bus into the destination register, and the output of the adder and logic circuit into the AC.
   - For example, the two microoperations
     DR←AC and AC←DR (Exchange) can be executed at the same time
   - This is done by:
   - 1- place the contents of AC on the bus (S2S1S0=100)
   - 2- enabling the LD (load) input of DR
   - 3- Transferring the contents of the DR through the adder and logic circuit into AC
   - 4- 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.

1.2.3 Computer Instructions

✓ The basic computer has three instruction code formats, as shown in below Figure.

✓ Each format has 16bits. 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 Instructions (Opcode = 000 ~ 110, I=0/1)

(b). Register-Reference Instructions (Opcode = 111, *I = 0*)

(c). Input-Output Instructions (Opcode =111, *I = 1*)

Figure: Basic Computer Instruction code format

On the basis of opcode and addressing mode, the basic computer has three 16-bit instruction code formats:

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

1. Memory Reference Instructions.

- First 12 bits (0-11) specify an address.
- bits of opcode are used to specify the types of instruction.
- Value of opcode ranges from 000 to 110.
- Opcode = 000 to 110 (I=0: 0xxx to 6xxx, I=1: 8xxx to Exxx).
- If I=0, it is direct addressing mode and if I=1, it is indirect addressing mode.

2. Register Reference Instructions.

- First 12 bits (0-11) specify the register operation.
- The next three bits equals to 111 specify opcode.

- The last mode bit of the instruction is 0 for register reference instruction.
- Therefore, left most 4 bits are always 0111 which is equal to hexadecimal 7.
- 7xxx (7800 to 7001)

3. Input / Output Instructions.

- First 12 bits (0-11) specify the I/O operation.
- The next three bits equals to 111 specify opcode.
- The last mode bit of the instruction is 1.
- Therefore, left most 4 bits are always 1111 which is equal to hexadecimal F.
- Fxxx (F800 to F040)

✓ The total number of instructions chosen for the basic computer is equal to 25.

✓ The instructions for the computer are listed in below table. The symbolic notation is three-letter word and represents an abbreviation.

Table: 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 Reference 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	

1.2.3.1 Instruction Set Completeness

The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

1. **Arithmetic, logical, and shift instructions**
   - ADD, CMA (Complement AC), INC (increment AC)
   - CIR, CIL
   - AND, CMA, CLA (clear AC)
2. **Instructions for moving information to and from memory and processor registers**
   - LDA (Load AC), STA (Store AC)
3. **Program control instructions together with instructions that check status conditions**
   - BUN (Branch Unconditional), BSA (Branch and Save return address),
   - ISZ (Increment and skip if zero) with 4 skip instructions (SPA, SNA, SZA, SZE)
4. **Input and output**
   - INP, OUT

1.2.4 Timing & Control

- ✓ 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 (i.e., Load).
- ✓ 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 for the accumulator.

Control units are implemented using one of the two organizations:

1. Hardwired control
2. Microprogrammed control

1. **Hardwired control**
   - The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
   - Fast mode of operation.
   - Difficult to change control logic.

2. **Microprogrammed control**
   - The control information is stored in a control memory.
   - Control memory is programmed to initiate the required sequence of microoperations.
   - Easy to change.

1.2.4.1 Control Unit of Basic Computer (Hardwired)

The block diagram of the control unit is shown in below figure:

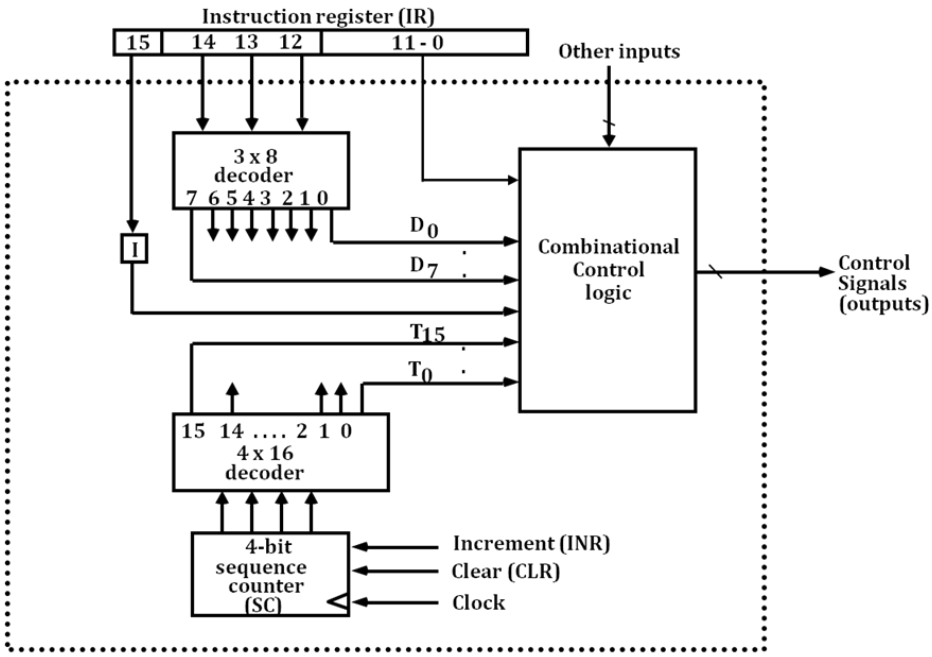


Figure: Control unit of Basic Computer(hardwired)

Hardwired Control Unit Control unit consist of:

- Instruction Register
- Number of Control Logic Gates,
- Two Decoders
- 4-bit Sequence Counter

✓ An instruction read from memory is placed in the instruction register (IR).

✓ The instruction register is divided into three parts: the I bit, operation code, and address part.

✓ First 12-bits (0-11) to specify an address, next 3-bits specify the operation code (opcode) field of the instruction and last left most bit specify the addressing mode I.

- I = 0 for direct address
- I = 1 for indirect address

✓ First 12-bits (0-11) are applied to the control logic gates.

- ✓ The operation code bits (12 – 14) are decoded with a 3 x 8 decoder.
- ✓ The eight outputs (D0 through D7) from a decoder goes to the control logic gates to perform specific operation.
- ✓ Last bit 15 is transferred to a I flip-flop designated by symbol I.
- ✓ The 4-bit sequence counter SC can count in binary from 0 through 15.
- ✓ The counter output is decoded into 16 timing pulses T0 through T15.
- ✓ The sequence counter can be incremented by INR input or clear by CLR input synchronously.

Timing signals

- ✓ The sequence counter SC can be incremented or cleared synchronously
  Eg: D3T4 : SC ← 0
  1. D3 becomes active
  2. D3T4 becomes active
  3. This signal is applied to the CLR
  4. On T4, SC is cleared to 0
  5. T0 becomes active instead of T5

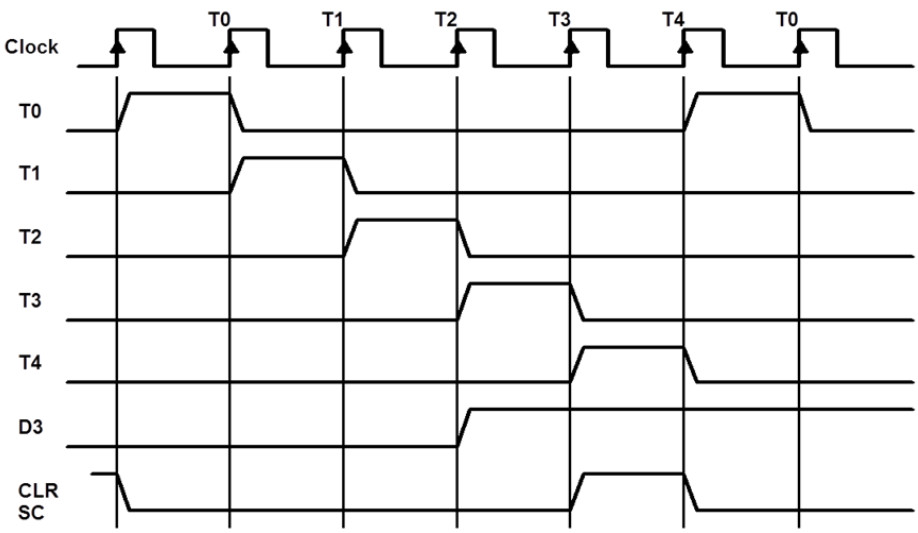


Figure: Example of Control timing signals

- A memory read or write cycle will be initiated with the rising edge of a timing signal.
- Assume: memory cycle time < clock cycle time!
- So, a memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its positive edge
- The clock transition will then be used to load the memory word into a register.
- The memory cycle time is usually longer than the processor clock cycle à wait cycles.

- ✓ Relation between the clock transition and timing signal

  Eg: **T0 : AR ← PC**
  - During the time T0 is active, PC is loaded onto the bus and LD of AR is enabled. The actual transfer occurs at the end of the clock cycle when clock goes through a positive transition.
  - Transfers the content of PC into AR if timing signal T0 is active
  - T0 is active during an entire clock cycle interval.
  - During this time, the content of PC is placed onto the bus (with $S_2S_1S_0$=010) and the LD (load) input of AR is enabled.
  - The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition.
  - This same positive clock transition increments the sequence counter SC from 0000 to 0001.
  - The next clock cycle has T1 active and T0 inactive.

1.2.5 Instruction Cycle

- ✓ A program is a sequence of instructions stored in memory.
- ✓ The program is executed in the computer by going through a cycle for each instruction (in most cases).
- ✓ Each instruction in turn is subdivided into a sequence of sub-cycles or phases.
- ✓ Instruction Cycle 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
- ✓ This cycle repeats indefinitely unless a HALT instruction is encountered.

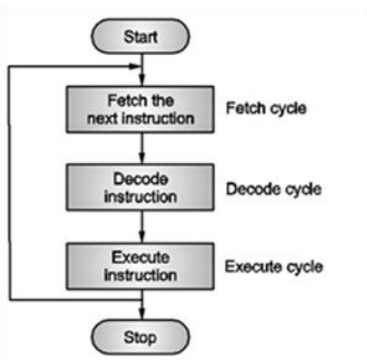
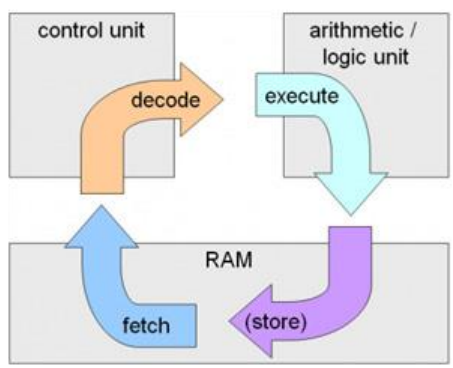


Figure: Basic Instruction Cycle



Fetch and Decode

- ✓ Initially, the PC register is loaded with the address of the first instruction in the program.
- ✓ The sequence counter SC is cleared to 0, providing a timing signal T0. After each clock T0, T1, T2, and so on.
- ✓ The micro operations for the fetch and decode phases can be specified by the following register transfer statements.
  - T_0 : $A_R \leftarrow P_C$ (this is essential!!)
  - The address of the instruction is moved to AR.
  - T_1 : IR←M[AR], PC←PC+1
  - The instruction is fetched from the memory to IR , and the PC is incremented.
  - T_2: D_0…, D_7← Decode IR(12-14), AR←IR(0-11), I←IR(15)

- ✓ T0: Since only AR is connected to the address inputs of memory, the address of instruction is transferred from PC to AR.
  1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 = 010.
  2. Transfer the content of the bus to AR by enabling the LD input of AR

     AR ← PC

- ✓ T1: The instruction read from memory is then placed in the instruction register IR. At the same time, PC is incremented to prepare for the address of the next instruction.
  1. Enable the read input of the memory.
  2. Place the content of memory onto the bus by making the bus selection inputs S2S1S0 = 111. (Note that the address lines are always connected to AR, and we have already placed the next instruction address in AR.)
  3. Transfer the content of the bus to IR by enabling the LD input of IR

     IR ← M[AR],
  4. Increment PC by enabling the INR input of PC (PC←PC+1)

- ✓ T2: The operation code in IR is decoded; the indirect bit is transferred to I, and the address part of the instruction is transferred to AR.
- ✓ Below Figure shows the register transfers for the fetch phase at T0 and T1. Similar circuits are used to realize the microoperations at T2.

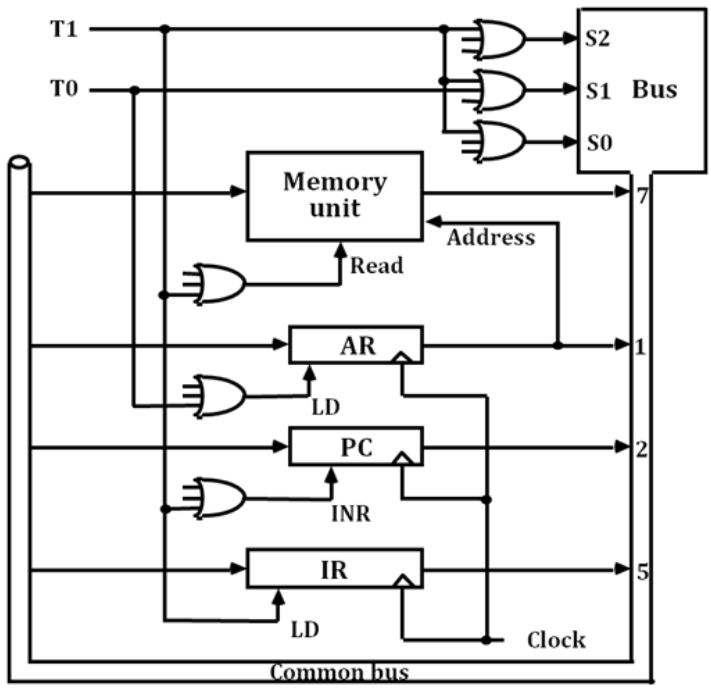
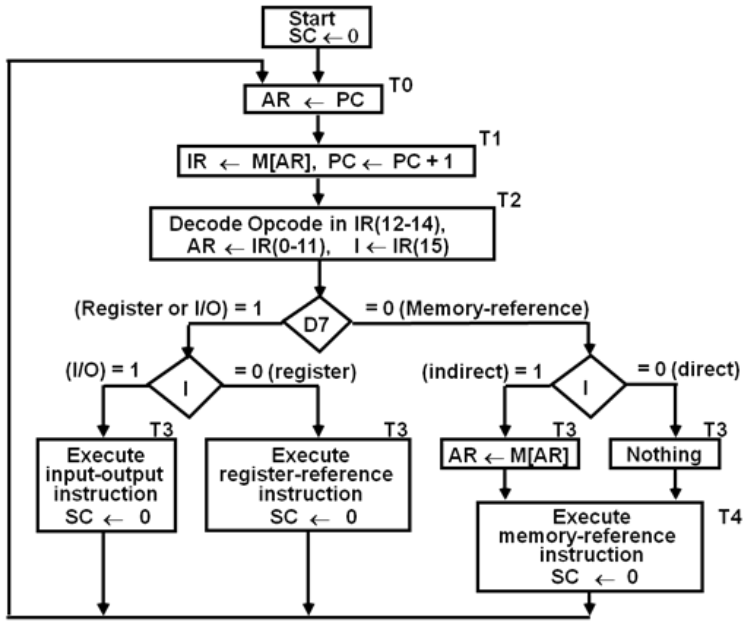


Figure: Register Transfers for the fetch phase

Determine the type of Instruction:

- ✓ At T3, microoperations which take place depend on the type of instruction read from memory.
- ✓ Below Figure presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after decoding.
- ✓ The four different paths are symbolized as follows, where the control functions must be connected to the proper inputs to activate the desired microoperations.
  D7' I T3: AR ← M[AR], indirect memory transfer.
  D7' I' T3: Nothing, direct memory transfer.
  D7 I' T3: Execute a register-reference instruction.
  D7 I T3: Execute an I/O instruction.
- ✓ The sequence counter SC must be incremented when D7'T3 = 1, so that the execution of the memory-reference instruction takes place with the next timing variable T4, after which SC is cleared to 0 and control returns to fetch phase with T0 = 1.



D'7IT3: AR ← M[AR]
D'7I'T3: Nothing
D7I'T3: Execute a register-reference instr.
D7IT3: Execute an input-output instr.

Figure: Flowchart for instruction cycle(initialConfiguration)

1.2.5.1 Register-Reference Instructions

- ✓ The 12 register-reference instructions are recognized by I = 0 and D7 = 1 (IR(12-14) = 111).
- ✓ These instructions use bits IR(0-11) of the instruction code to specify one of 12 instructions.
- ✓ Each operation is designated by the presence of 1 in one of the bits in IR(0-11). Therefore D7 I' T3 = r is common to all register-transfer instructions.
- ✓ The control function for CLA can be written as D7I'T3B11= rB11.

Table: Execution of Register-Reference Instructions

$r = D_7 I'T_3$ => Register Reference Instruction
$B_i = IR(i)$, i=0,1,2,...,11

	r:	SC ← 0	Clear SC
CLA	rB_{11}:	AC ← 0	Clear AC
CLE	rB_{10}:	E ← 0	Clear E
CMA	rB_9:	AC ← AC'	Complement AC
CME	rB_8:	E ← 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 zero
SZE	rB_1:	if (E = 0) then (PC ← PC+1)	Skip if E zero
HLT	rB_0:	S ← 0 (S is a start-stop flip-flop)	Halt Computer

1.2.6 Memory-Reference Instructions

- ✓ Opcode (000 - 110) or the decoded output Di (i = 0, ..., 6) are used to select one of the memory-reference instructions.
- ✓ The effective address of the instruction is in AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1
- ✓ Memory cycle is assumed to be short enough to be completed in a CPU cycle.
- ✓ The execution of these instructions starts with timing signal T4.
- ✓ Below table shows seven memory-reference instructions.

Table: Memory Reference Instructions

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

- – The AND instruction performs a bit-wise logical AND :: AC and Memory Word.
- – The ADD instruction performs a regular addition :: AC and Memory Word.
- – The LDA and STA are regular load and store instructions :: Memory Word to/ from AC .
- – The BUN (branch unconditional) is a jump instruction.
- – The BSA is used to call subroutines in the basic computer.
- – The ISZ instruction is used for program loops.

1. AND to AC

- ✓ Performs the AND operation on pairs of bits in AC and the memory word specified by effective address.
- ✓ Microoperations for this instruction:

D_0T_4 :	DR ← M[AR]	Read operand
D_0T_5 :	AC ← AC Ù DR, SC ← 0	AND with AC

2. ADD to AC

- ✓ Performs the ADD operation on pairs of bits in AC and the contents in memory word specified by effective address.
- ✓ Microoperations for this instruction:

D_1T_4 :	DR ← M[AR]	Read operand
D_1T_5 :	AC ← AC + DR, E ← C_{out}, SC ← 0	Add to AC and stores carry in E

3. LDA: Load to AC

- ✓ Transfers the memory word specified by the effective address to AC.
- ✓ Microoperations for this instruction:

D_2T_4: DR ← M[AR] Read operand

D_2T_5: AC ← DR, SC ¬ 0 Load into DR

4. STA: Store AC

- ✓ Stores the content of AC into the memory word specified by effective address.
- ✓ Microoperation for this instruction:

D_3T_4: M[AR] ← AC, SC ← 0

5. BUN: Branch Unconditionally

- ✓ Transfers the program control to the instruction specified by the effective address.
- ✓ Allows specify instruction out of sequence.
- ✓ Program branches (jumps) unconditionally.
- ✓ Microoperation for this instruction:

D_4T_4: PC ← AR, SC ← 0

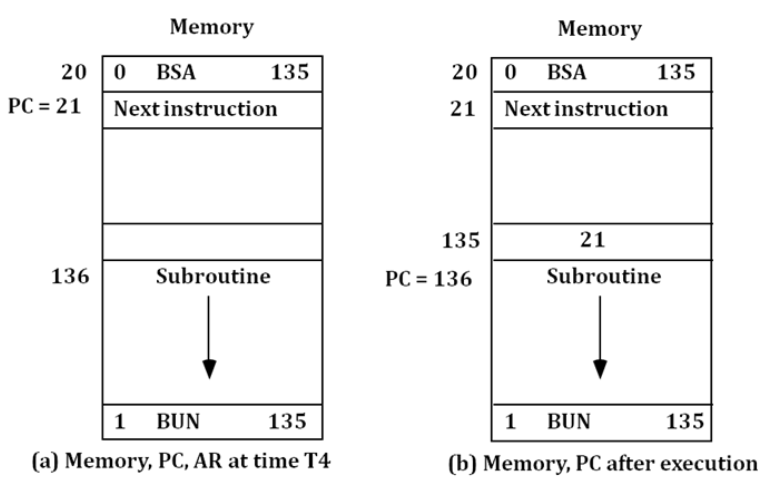
6. BSA: Branch and Save Return Address

- ✓ Useful for branching to a portion of the program called a subroutine or procedure.
- ✓ When executed, BSA does the following operation.

M[AR] ← PC, PC ← AR + 1 R holds effective address

A numerical example that demonstrates how this instruction is used with subroutine is shown below:

Figure: Example of BSA instruction execution



(a) Memory, PC, AR at time T4 (b) Memory, PC after execution

Subroutine Call

- ✓ The BSA instruction performs the operation / function referred to as a subroutine call.
- ✓ The indirect BUN instruction at the end of the subroutine performs the function referred to as subroutine return.
- ✓ BSA: executed in a sequence of two micro-operations:

D_5T_4 : M[AR] ← PC, AR ← AR + 1

D_5T_5 : PC ← AR, SC ← 0

7. ISZ: Increment and Skip-if-Zero

- ✓ Increments the word specified by the effective address, and if the incremented value is equal to zero, PC is incremented by 1.
- ✓ Negative numbers are stored in 2's complement form.
- ✓ Microoperations for this instruction:

D_6T_4 : DR ← M[AR]

D_6T_5 : DR ← DR + 1

D_6T_6 : M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

Control Flowchart

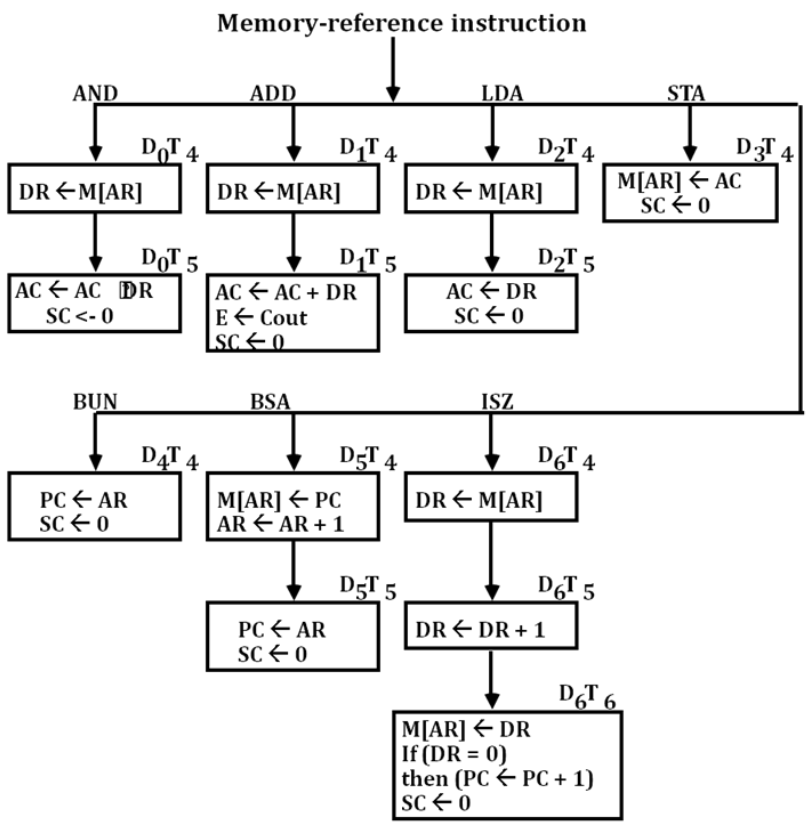


Figure: Flow chart for Memory-reference instruction

1.2.7 Input-Output and Interrupt

- ✓ Instructions and data stored in memory must come from some input device.
- ✓ Computational results must be transmitted to the user through some output device.

1.2.7.1 Input-Output Configuration / Programmed Control Transfer

- ✓ For the system to communicate with an input device, serial information is shifted into the input register INPR.
- ✓ To output information, it is stored in the output register OUTR.

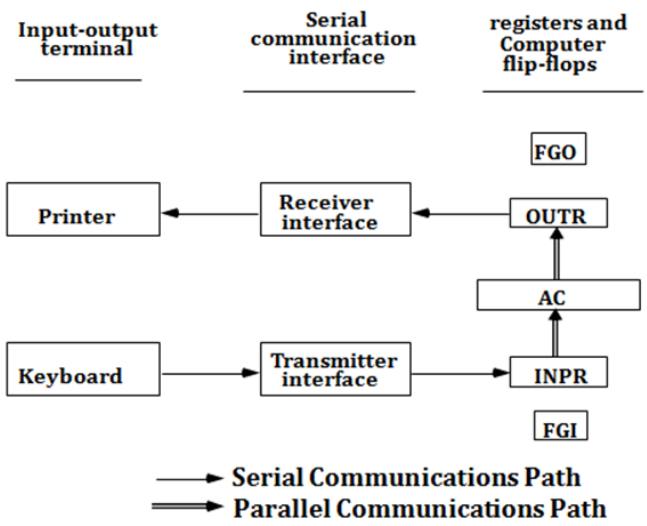


Figure: Input-Output Configuration

INPR Input register - 8 bits
OUTR Output register - 8 bits
FGI Input flag - 1 bit
FGO Output flag - 1 bit
IEN Interrupt enable - 1 bit

- INPR and OUTR communicate with a communication interface serially and with the AC in parallel. They hold an 8-bit alphanumeric information.
- I/O devices are slower than a computer system à we need to synchronize the timing rate difference between the input/output device and the computer.
- FGI: 1-bit input flag (Flip-Flop) aimed to control the input operation.
- FGI is set to 1 when a new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- FGO: 1-bit output flag used as a control flip-flop to control the output operation.

- If FGO is set to 1, then this means that the computer can send out the information from AC. If it is 0, then the output device is busy and the computer has to wait.
- The process of input information transfer:
  - Initially, FGI is cleared to 0.
  - An 8-bit alphanumeric code is shifted into INPR (Keyboard key strike) and the input flag FGI is set to 1.
  - As long as the flag is set, the information in INPR cannot be changed by another data entry.
  - 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 the input device (striking another key)
- The process of outputting information:
  - 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 accepts the coded information (prints the corresponding character).
  - When the operation is completed, the output device sets FGO back to 1.
  - The computer does not load a new data information into OUTR when FGO is 0 because this condition indicates that the output device is busy to receive another information at the moment.

1.2.7.2 Input-Output Instructions

- ✓ Needed for:
  - Transferring information to and from AC register
  - Checking the flag bits
  - Controlling the interrupt facility
- ✓ The control unit recognize it when D7=1 and I = 1.
- ✓ The remaining bits of the instruction specify the particular operation.
- ✓ Executed with the clock transition associated with timing signal T3.
- ✓ Input-Output instructions are shown in below table.

Table: Input-output Instructions

$D_7IT_3 = p$ (Common to all input-output instructions)
$IR(i) = B_i$, i = 6, ..., 11 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

Programmed control transfer

- The difference of information flow rate between the computer and that of the I/O device makes this type of transfer inefficient.
- Consider a computer that can go through an instruction cycle in 1μs.
- The I/O device can transfer information at maximum rate of 10 character per second ≡ one character every 100,000 μs.
- 2 instructions are executed when the computer checks the flag bit and decides not to transferat the maximum rate, the computer will check the flag 50,000 times between each transfer.
- Computer is wasting time

1.2.7.3 Program Interrupt

- The process of communication just described above is referred to as **Programmed Control Transfer**.
- The computer keeps checking the flag bit, and when it finds it set, it initiates an information transform (this is sometimes called **Polling**).
- This type of transfer is in-efficient due to the difference of information flow rate between the computer and the I/O device.
- The computer is wasting time while checking the flag instead of doing some other useful processing task.
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer. This type of transfer uses the **interrupt facility**.
- While the computer is running a program, it does not check the flags
- Instead:
  - When a flag is set, the computer is immediately interrupted from proceeding with the current program.
  - The computer stops what it is doing to take care of the input or output transfer

- Then, it returns to the current program to continue what it was doing before the interrupt

- The interrupt facility can be enabled or disabled via a flip-flop called IEN.
- The interrupt enable flip-flop IEN can be set and cleared with two instructions (IOF, ION):
  - IOF: IEN ← 0 (the computer cannot be interrupted)
  - ION: IEN ← 1 (the computer can be interrupted)
- The way that the interrupt is handled by the computer can be explained by the following flow chart.

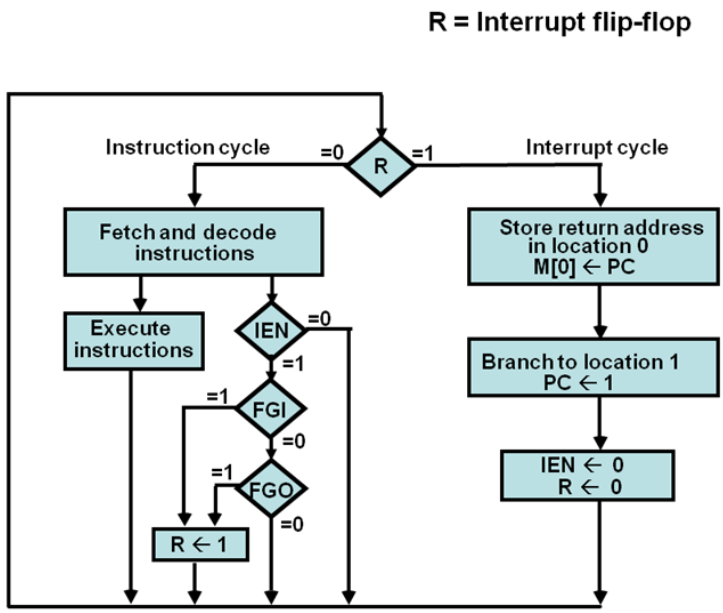


Figure: Flowchart for interrupt cycle

- Another flip-flop (called the interrupt flip-flop R) is used in the computer's interrupt facility to decide when to go through the interrupt cycle.
- FGI and FGO are different here compared to the way they acted in an earlier discussion.
- So, the computer is either in an Instruction Cycle or in an Interrupt Cycle

Interrupt Cycle

- The interrupt cycle is a hardware implementation of a branch and save return address operation (BSA).
- 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.
- For our computer, we choose the memory location at address 0 as a place for storing the return address.
- Control then inserts address 1 into PC: this means that the first instruction of the interrupt service routine should be stored in memory at address 1, or, the programmer must store a branch instruction that sends the control to an interrupt service routine.
- IEN, R ← 0: no more interruptions can occur until the interrupt request from the flag has been serviced.
- The service routine must end with an instruction that re-enables the interrupt (IEN ← 1) and an instruction to return to the instruction at which the interrupt occurred.
- The instruction that returns the control to the original program is "indirect BUN 0".
- Example: the computer is interrupted during execution of the instruction at address 255

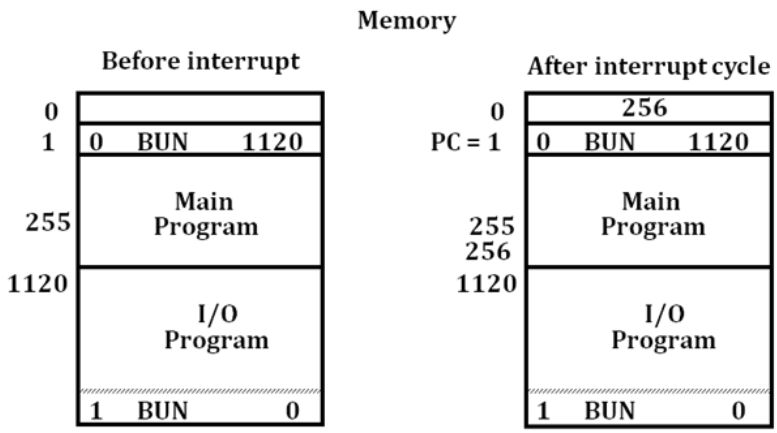


Figure: Demonstration of the Interrupt cycle

1.2.7.4 Interrupt Cycle

Register Transfer Statements for Interrupt Cycle

- R ← 1 if IEN (FGI + FGO)$T_0'T_1'T_2'$
  $\Leftrightarrow T_0'T_1'T_2'$ (IEN)(FGI + FGO): R ← 1
- The fetch and decode phases of the instruction cycle must be modified
  Replace T_0, T_1, T_2 with $R'T_0, R'T_1, R'T_2$
- Fetch and decode phases occur at the instruction cycle when R = 0
- The interrupt cycle :

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

- Further Questions:
  - How can the CPU recognize the device requesting an interrupt?
  - Since different devices are likely to require different interrupt service routines, how can the CPU obtain the starting address of the appropriate routine in each case?
  - Should any device be allowed to interrupt the CPU while another interrupt is being serviced?
  - How can the situation be handled when two or more interrupt requests occur simultaneously?

1.2.8 Complete Computer Description

The final flowchart for computer operation:

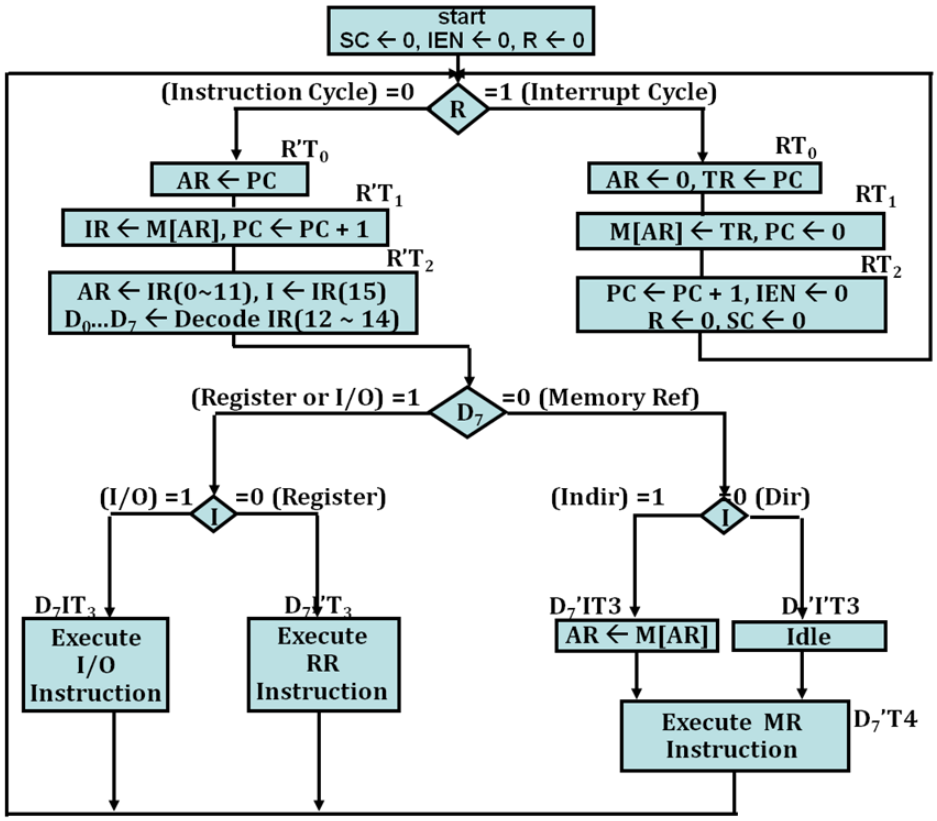


Figure: Flowchart for Computer operation

The control functions and microoperations for the basic computer: